VIRUS BULLETIN INTERNATIONAL CONFERENCE

Boston Park Plaza Hotel and Towers
20 - 22 September

95

Symantec 1026
Symantec v. Finjan
IPR2015-01892

VIRUS BULLETIN

INTERNATIONAL CONFERENCE

Proceedings of

The Fifth International Virus Bulletin Conference

Virus Bulletin Ltd

21 The Quadrant, Abingdon, OX14 3YS, England

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use of operation of any methods, products, instructions of ideas contained in the material herein.

FOREWORD

CONTENTS

DAY 1

Corporate stream

Technical stream 1

Technical stream 2

Late additions

DAY 2

Corporate stream

Technical stream 1

Technical stream 2

Late additions

THE SPEAKERS

WES AMES

Wes Ames is a Senior Principal Scientist in Personal Computer Hardware and Operating Systems for the *Boeing Company* in Seattle, Washington. He is responsible for specific hardware and O/S standards and support in the *Boeing Company*. Over the past six years, Ames has managed the anti-virus activities for *Boeing*, which has responsibility for over seventy thousand personal computers worldwide. These activities range from policy determination to technical support training.

Ames has created the *Boeing* corporate policies necessary for reducing the risk from computer viruses, and is responsible for their implementation and update. He teaches anti-virus classes to technical support analysts, and consults with corporate customers on virus methodologies. He has lectured and led anti-virus discussion groups at the Law Enforcement Conference for Computer Security, and the Society for Information Management.

DAVID AUBREY-JONES

David Aubrey-Jones has been a prolific figure in the computer industry since 1980 and is an authority on viruses and anti-virus warfare. Aubrey-Jones has a PhD from *Leeds University*. In 1988, he started his own company, *Speedlock*, which specialised in copy protection, and soon became a market leader. It was through copy protection that Aubrey-Jones became involved with *Reflex Magnetics,* finally joining as Technical Director in 1991.

Aubrey-Jones is the author of disknet™, *Reflex's* multi-layered computer security solution, which currently protects over 750,000 PCs in multi-nationals, government institutions and other blue-chip organisations worldwide.

JIM BATES

Jim Bates has been involved in electronics all his working life. After service as an Air Radar Engineer in the *Royal Air Force*, he worked as an Electronic Service Engineer on early computers and tabulators. When computer viruses appeared, he was the first in the UK to disassemble them. In 1989, he broke the code encryption and analysed the infamous AIDS Information Disk. This marked the start of his connection with the *Computer Crime Unit* at *New Scotland Yard*: he is now regularly consulted by them and other national and international law enforcement agencies. He runs his own company, *Computer Forensics Ltd*, and he is the designer of the DIBS copying system.

As well as being a respected member of *Virus Bulletin's* advisory board, Bates also belongs to the Computer Security Specialist Group of the *British Computer Society*. He holds a degree in Electronic Engineering, and was elected a Fellow of the *Institute of Analysts and Programmers* in 1987. He was appointed President of the ruling council of the *IAP* in 1993, and is an active member of the *Forensic Science Society*.

NEVILLE BULSARA

Neville Bulsara first began programming in assembler at the age of 18. In 1988, at the age of 21, he pioneered the anti-virus movement in India by being the first in the country to take apart the Brain virus and write an antidote for the same. Since then, he has been at the forefront of the battle against viruses in India.

Since 1989, Bulsara has served as a consultant on the field to the Government of India, some of the largest corporations in the country and various defence establishments. He considers the lack of user-awareness as the greatest factor contributing to the virus menace, and so is to be regularly found lecturing on the subject at user-group meetings and computer shows.

LUCIJAN CARIC

Lucijan Caric gained his LL.B. at the *University of Zagreb*. When he started to work in the area of computer security and anti-virus measures in 1991, he already had extensive experience in the computing. He joined *United Nations Peace Forces* (*UNPF*) in the former Yugoslavia in 1993 and is currently the Special Projects Coordinator in the Information Technology Services Section. Caric is responsible for computer security projects and the development of major projects conducted by the section.

In an effort to improve standards of computer security and anti-virus protection in Croatia, Caric is also acting as a contributing editor to a leading Croatian computer magazine, *Bug*, and has taken part in a series of broadcasts about computer viruses on the metropolitan TV station.

GLENN COATES

Glenn Coates is 23 years old and has recently completed a BSc (Hons) degree in Computing Science at *Staffordshire University*. For his final year project, he developed a prototype Virus Description Language (VDL) upon which his paper at *VB '94* was based.

He is now working at *Security Information Systems Ltd* (*SISL*) as a trainee security evaluator. His other computing interests include operating systems design, compiler writing, neural networks and human computer interaction. His hobbies include fitness training, parachuting and socialising.

PAUL DUCKLIN

Paul Ducklin's involvement in the anti-virus field started in 1989 in South Africa, at the time that computer viruses first began to appear there. He spent five years as the head of the Computer Virus Lab at the South African *Council for Scientific and Industrial Research* in Pretoria, before moving to England earlier this year to join the anti-virus team at *Sophos Plc*, the producers of SWEEP. Though a recent arrival in the EC, he has attempted to make his mark as a true European by eating British cheese, driving a French car, and riding an Italian motorcycle.

ALLAN DYER

Allan Dyer first studied biological viruses, graduating in Microbiology from *University College*, London, in 1984. He switched fields, gaining a Master's in Control Engineering from *Sheffield University* in 1987 and combined his skills programming in a haematology research laboratory. He first met computer viruses in 1988 while a Systems Programmer at the *London School of Hygiene and Tropical Medicine*, and joined the team controlling them in a number of London colleges. He moved to Hong Kong in 1993, and now manages F-PROT Professional for *Yui Kee Co. Ltd.*

JUDY EDWARDS

Judy Edwards is a Microcomputer Software Specialist at *Illinois State University*, where she is a member of the WWW Team and an ftp site administrator. She provides Internet training and help desk support to faculty, staff and instructional computer labs, and also does independent Internet consulting.

Edwards holds a Master's degree in Instructional Systems Technology from *Indiana University's* College of Education, and a Personal Computer Coordinators certificate from the *University of Southern Maine.*

RICHARD FORD

Richard Ford obtained a BA in Physics from Queen's College, Oxford, in 1989, and went on to study for a D. Phil. in Semiconductor Physics. His interest in computer viruses began during the course of his research, when the computer he was using became infected with the Spanish Telecom virus. The virus triggered, nearly destroying six months' worth of results, but rather than turning to anti-virus software for the answer, Ford analysed the virus himself.

In the following year, he wrote various articles for *Virus Bulletin* and became editor in January 1993. He has since lectured and talked world-wide on the problems posed by malicious software. In April of this year he joined the *National Computer Security Association* (*NCSA)* in the US as Director of Research.

SARAH GORDON

Sarah Gordon, Security Analyst for *Command Software Systems, Inc*, has been an invited speaker at such diverse conferences as those sponsored by *The American Association for the Advancement of Science, EICAR, DEFCON,* and *Virus Bulletin.* A frequent contributor to security industry technical publications, she is also the winner of the Sec 94 IFIP TC11 award for her research on social and ethical implications of technology: 'Technologically Enabled Crime: Shifting Paradigms for the Year 2000'.

She has recently completed research projects at *Indiana University* in Unix Security and in Computer Ethics. Current projects include 'Development of Information Security Education in Developing Countries', and 'Anti-Virus Product Certification Methodologies'. Sarah can be heard at the upcoming *National Computer Security Conference* in Baltimore, in October and also at *Compsec* in London, in November.

SCOTT GORDON

Scott Gordon is the Product Manager for *McAfee Associates*' security solutions. Acting as the focal point relating to product development, positioning, delivery and support, he is a company spokesperson and among resident experts on issues relating to computer viruses, enterprise date quality assurance, and security. Gordon's background in the computer industry spans a broad range of experience; from retail, channel and corporate sales to consulting and product marketing / development manager for network security and management.

Prior to joining *McAfee*, Gordon was the product development manager for network security and management products at *Cheyenne Software.* Before this, he was responsible for product marketing and technical sales functions with *Computer Associates International* and, previously, was a product manager with a network technology seminar company and an independent systems consultant. He has an MBA from the *University of Phoenix* and a BBS from *Hofstra University.*

IGOR GREBERT

Igor Grebert studied in Paris and graduated in 1989 from *Ecole Centrale de Paris*, with a major in bio-technology. He worked as a post-doctorate researcher at Stanford designing neural networks applied to target tracking, image analysis and automatic pilots. In 1990, he started to use his pattern matching expertise to help detect the growing number of computer viruses. In 1993, he headed the redesign of *McAfee's* VirusScan product line as manager of the research and development team. Today he applies his skills to designing enhanced anti-virus systems, integrating his years of experience in the field.

DMITRY GRYAZNOV

Dmitry Gryaznov was born in 1961 in Frunze, Kirghyzskaya SSR, USSR. He was educated at the *Moscow Skills Improvement Institute* and the *Moscow Physics and Technology Institute* (*MPhTI*), where he gained a Unix Operating System programmer and administrator certificate and an MSc in Computer Science in 1984.

Since graduating, Gryaznov has held various positions in the *Program Systems Institute* at the *Russian Academy of Sciences*. Earlier this year he moved to the UK, and now works as a senior virus research analyst with *S&S International Plc.*

HAROLD HIGHLAND

Dr. Harold Joseph Highland, FICS, FACM, is the only Fellow of both the *Irish Computer Society* and the *US Association for Computer Machinery*. The career of this 'elder statesman' of computing spans over 57 years with experience in the military, industry and academia.

His professinal life started when he was designated as Honor Graduate of his military class and commissioned on his college graduation in 1938. He served as Provost Marshall and was seconded to cryptographic analysis and later to intelligence analysis. In addition to working for *The New York Times* and other newspapers, Highland was a research statistician, an economist, a management consultant, a methods engineer, a magazine editor and publisher, a television producer and even MCed one of his programs. He also owned an advertising/public relations organization, was a dean of a graduate school, associate dean of a liberal arts college, director of various computer centers, a consultant, and a classroom teacher. Likewise, he has worked with various government agencies and even today serves as computer security consultant to the Beijing government.

Prior to his retirement in 1981, Highland planned a new international journal, *Computers & Security*, the first issue of which appeared in 1982: he was Editor-in-Chief. In 1984, his publication became the official journal of *International Federation for Information Processing's Technical Committee 11* on information security [IFIP/TC11]. Dr. Highland is a prolific author, who has written 27 books in the past 35 years. He has also published and/or presented over 200 technical papers in various areas of computing at regional, national and international conferences, as well as in professional journals.

JEAN HITCHINGS

Jean Hitchings obtained a BSc in Computer Science from the *University of Westminster* in 1982 and went on to study for an MSc in the same subject at the *University of London.* Since January 1992 Jean has been a lecturer in Information Technology at the *University of Nottingham.* She has recently gained a PhD in Computer Science from the *University of East Anglia.*

JAN HRUSKA

Jan Hruska is the Technical Director of *Sophos Plc* in Oxford. A graduate of Downing College, Cambridge, he gained his doctorate at Magdalen College, Oxford. In April 1980, he formed *Sophos* with Dr. Peter Lammer as a computer design partnership: the company was incorporated in 1987 and specialises in data security. He is a co-author (with Dr Keith Jackson) of 'The PC Security Guide', published by *Elsevier*, and 'Computer Security Solutions', published by *Blackwells*. He is the author of 'Computer Viruses and Anti-Virus Warfare', published by *Ellis-Horwood*. Hruska regularly speaks at computer security conferences and consults on a number of security aspects, including virus outbreaks. His extra-curricular interests include flying, skiing, scuba-diving and piano-playing and he is an ex-member of *Mensa*.

JAKUB KAMINSKI

Jakub Kaminski graduated and received an MSc in Electronics from *Warsaw Technical University* in 1986. He went on to work for the *Institute of Fundamental Technological Research*, at the *Polish Academy of Sciences*, spending most of his time doing system programming and working in different assemblers.

In 1992, Kaminski moved to Australia and started working for *CYBEC*. He disassembles new viruses and incorporates them into *CYBEC*'s VET. In June 1995, he joined the *Virus Bulletin* team as Technical Editor.

ROBIN KINNEY

Robin Kinney has dedicated nearly his entire career to devices used for treating cancer. The last nine years he has spent in software management of *Varian Oncology Systems,* where he has managed both departments and projects.

Kinney is an advocate of software process improvement. He led the effort within *Varian Oncology Systems* for ISO-9001 certification, and has spent more than a year as chair of the Software Process Improvement Committee within that organization.

PAVEL LAMACKA

Pavel Lamacka graduated in 1971 and, in 1983, gained a degree in computer science from the *Slovak Technical University* in Bratislava. He has worked at several research institutions, mainly in software engineering, taking part in work on software - including a real-time operating system for the first Slovak control computer. He was a member of the team which designed and implemented the BPS programming system based on a MODULA-2-like programming language (this system was used for years on *IBM* mainframes and *DEC* minicomputers) and has also worked on a perspective block-building programming system.

In Spring 1988, he encountered and disassembled his first virus and gave his first lecture on viruses and other computer infiltration means. Since then, he has been active in computer security, initially as a consultant, later as an author of computer security products. Lamacka is currently the Head of the *Computer Accidents Research Center*, which he formed in 1992 and which is based in *HTC*, a large private computer company.

MIKE LAMBERT

Mike Lambert is Electronic Security Manager at *Frontier Corporation*. He has been involved with computer viruses since 1988; doing analyses, testing products and assisting in cases of virus infections. He has written other papers on Disaster Recovery Disks for the PC and fatal DOS vulnerability.

FERENC LEITOLD

Ferenc Leitold graduated from the Department of Informatics at *Budapest Technical University*. Having completed a three year post-graduate course at the university, he is now in the process of doing a PhD on the mathematical modelling of operating systems and computer viruses.

Leitold joined the fight against computer viruses in 1988, with the appearance of the first virus in Hungary (Cascade). He is a founding member of the *Hungarian VirusBuster Team*, operating under the aegis of *Hunix Ltd.* Their anti-virus product, VirusBuster for DOS and NetWare, is sold throughout Hungary.

JONATHAN LETTVIN

Jonathan Lettvin has been *Lotus*' anti-virus principal investigator for six years. He developed all of the company's anti-virus policies and procedures. He also created the *Lotus* 'Release Engineering Anti-virus Laboratory' (REAL): this is responsible for examining all *Lotus* products for viruses before shipping. REAL has an unblemished record of preventing viruses being shipped in *Lotus* products.

Lettvin has been programming for many years, and views his anti-virus work at *Lotus* as strongly influenced by his training at *MIT*, medical school and *Bell Labs*, as well as some beneficial professional partnerships.

OverByte was incorporated to develop and market products based on Lettvin's experience fighting viruses in the *Lotus* corporate environment. *Lotus* has been generous in granting all commercial rights for DisQuick/ViRemove to *OverByte* and is its first and best customer.

DMITRY MOSTOVOY

Dmitry Mostovoy was born in 1962, in Moscow. He graduated from the *Moscow Aviation Institute*, specializing in Space Science, and then worked at the *Keldysh Institute of Applied Mathematics* at the *Russian Academy of Sciences*, on the dynamics of the re-entry of space vehicles. Mostovoy participated in the Russian orbiter 'Buran' project.

Mostovoy obtained a PhD degree in theoretical mechanics and, in 1989, became interested in the computer virus problem. Since 1991, he has been a leading anti-virus designer at *DialogueScience Inc*, and the author of one of the renowned Russian anti-virus utilities, ADinf, a data integrity checker. Mostovoy is also an active yachtsman.

IGOR MUTTIK

Muttik Igor was born in Moscow in 1962. He graduated from the Physics Department of *Moscow State University* in 1985, where he subsequently worked on low temperature physics and used computers in physics experiments. In 1989, he received a PhD in physics and mathematics from *Moscow University*. He then worked on the use of computers in education and experiments, and published more than 50 scientific articles in various Russian and international magazines. In 1988, he became interested in computer viruses, although this anti-virus activity was just a hobby.

A programmer and a researcher, Muttik has developed an interest in the fundamental investigation of viruses. He is especially engaged in complex polymorphic, armored and multi-partite viruses. In 1994 he joined *CARO*. In August 1995 he was appointed Virus Research Analyst at the Virus Laboratory of *S&S International Plc*, in Aylesbury, UK.

PETER RADATTI

Pete Radatti is the founder and President of *CyberSoft, Inc*, manufacturers of VFind, the antivirus software product which executes on Unix systems. VFind simultaneously scans for Unix, MS-DOS, Macintosh and Amiga destructive software while providing cryptographic integrity to filesystems.

ROGER RIORDAN

Roger Riordan graduated in Electrical Engineering from *Melbourne University* in 1954. After two years with *English Electric* in the UK, and some years with *CSIRO*, he set up *CYBEC Electronics* in 1973. At *CYBEC*, he designed a wide range of scientific and industrial equipment. He joined *Chisholm Institute of Technology* as a lecturer in Electronics in 1983, and became involved with computer viruses in 1989, when the PC labs were paralysed by an outbreak of the Stoned virus. He wrote the first version of VET to counter it, and gave it to the students as shareware.

Riordan has attended a number of international conferences, and published several papers on his work related to virus research. He is a member of *CARO*, the international anti-virus research organisation.

PAUL ROBINSON

Paul Robinson, Editor of *SECURE Computing* magazine, has a long track record writing about security issues and related solutions. Prior to assuming the editorship, he wrote for many of the top UK computer and business magazines. *SECURE Computing* is an international security journal with one of the largest circulations of a publication in its field.

FRIDRIK SKULASON

Fridrik Skulason received a BSc from the *University of Iceland*. In 1987, he started his own software company in Reykjavik, specialising in programs tailored for Icelandic needs. Skulason became involved in computer viruses in early 1989, when they first appeared in Iceland. He is the author of F-Prot anti-virus software and is a former Technical Editor of *Virus Bulletin*.

DAVID STANG

David Stang has been involved in computer security for several years, and is currently the Chief Technical Officer for *Norman Data Defense Systems, Inc.* He was the founder of the *National Computer Security Association* (*NCSA)* and also founder and chairman of the *International Computer Security Association* (*ICSA*), the umbrella organization for the *NCSA*. He is the author of several books on computer security including *Norman*'s 'Computer Virus Handbook', and co-author (with Syliva Moon) of 'Network Security Secrets'. Stang edited *Virus News and Reviews* (*VNR*), a journal which was published monthly throughout 1992. He is also a member of the editorial board and columnist for *InfoSecurity News,* and has contributed over 160 articles to the computer trade press.

Stang holds a PhD from *Syracuse University*, an MS from the *University of Toronto* and a BS from *Cornell University*.

MORTON SWIMMER

Morton Swimmer was born in New York City, USA. After moving to Germany, he studied first in England and then at the *University of Hamburg*, Germany. He is currently close to completing his Master's degree in Computer Science (Informatik).

Swimmer has been a member of the Virus Test Center at the *University of Hamburg* since its inception in 1988. He has also managed *S&S International (Deutschland) GmbH* as well as working in the Virus Lab at *S&S International Plc*, UK. His research interests are in computer and network security, in particular computer viruses and worms.

IAN WHALLEY

Ian Whalley has been Editor of *Virus Bulletin* since April 1995; before that he worked at *Sophos Plc* developing an anti-virus solution for Windows NT. He is a graduate of *Manchester University* (1994), where he studied Physics and Computer Science, and it was here that he first became interested in the field of computer security. He maintains a keen interest in viruses on the new generation of PC operating systems, not least Windows NT.

STEVE WHITE

Steve White received a PhD from *UCSD* in theoretical physics in 1982, and since then has been at the *IBM Thomas J. Watson Research Center.* He has had articles published on a variety of subjects, including condensed matter physics; optimization by simulated annealing; software protection; computer security and computer viruses. White holds several patents in security-related fields. He organized and now manages the High Integrity Computing Laboratory at *IBM Research,* where he is responsible for the research and development of *IBM* anti-virus products. His research interests include the long-term implications of computer viruses and other self-replicating programs in distributed systems.

RIGHARD ZWIENENBERG

Righard Zwienenberg is the Research & Development Manager of *Computer Security Engineers Ltd.* He started dealing with computer viruses in 1988 after encountering the first virus problem on a system at the *Technical University of Delft.* His interest thus kindled, Zwienenberg has studied virus behaviour and presented solutions and detection schemes ever since - initially as an independent consultant and later, in 1991, with *CSE.* His interests have now broadened to include general security issues, such as network protection and internet firewalls.

THE DELEGATES

(as of 29/08/95)

A

Emmanuel Areola	EBA Communications	UK

B

Sqn Ldr Mark Baker	RAF High Wycombe	UK
Philip Bancroft	Digital Equipment Corporation	USA
Pavel Baudis	Alwil Software	Czech Republic
Ken Bauman	Computer Security Consultants Inc	USA
Richard Beard	State Street Bank	USA
Juergen Benz	Deutsche Telekom AG	Germany
Joseph Bernfeld	Merrill Lynch	USA
Daphne Bertrand	United Parcel Service	USA
Derril Bibby	Texaco Group Inc	USA
Robin Bijland	ESaSS GmbH	The Netherlands
Pat Bitten	S&S International	UK
Jim Blackwell	US Department of Agriculture	USA
Peter Bohm	NoVIR Data	Germany
Vesselin Bontchev	Frisk Software International	Iceland
Kevin Bosworth	British Telecom	UK
Adrienne Botti	Department of the Navy	USA
Donald Boyd	New York Times	USA
Carl Bretteville	Norman Data Defense Systems	Norway
James Brown	Fidelity Investments	USA
Charles Brown	Keiretsu Institute	USA
Torri Buchwald	Pratt & Whitney	USA
John Butler	The Automobile Association	UK

C

Bob Cartwright	Chevron	USA
Banda Casella	Royal Bank of Canada	Canada
Alex Chen	Cheyenne Software Inc	USA
Sing Bin Chew	TAS Inc	USA
Graham Cluley	S&S International	UK
Gary Cornelius	Command Software Systems	USA

D

Martin Damen	Ministry of Defence	UK
Helen Dawe	Sophos Plc	UK
Chris Debracy	Command Software Systems	USA
Paul Docherty	Portcullis Computer Security	UK
Moti Dover	Eliashim Microcomputers Inc	USA
Dyan Dyer	Command Software Systems	USA

F

Wesley Fagan	US Army	USA
Tom Farrell	Alternative Computer Technology	USA
Cheryl Flerk	Detroit Edison	USA
Thomas Le Fleur	S&S International	UK
David Flury	BT Payphones	UK
Dan Fox	Defense Logistics Agency	USA
Susan Franco	American Airlines	USA

G

Blase Gaude	Sandia National Laboratories	USA
Mr Donny Gilor	Iris Software	Israel
Ray Glath	RG Software Systems	USA
Eint Goedhart	Ministry of Defense	The Netherlands
Albert Gorter	NATO / NAPMA	The Netherlands
James Gosler	Sandia National Laboratories	USA
Paul Graham	Bureau of Reclamation	USA
Jeremy Gumbley	Command Software Systems	USA
Pege Gustafsson	Telia AB	Sweden

H

Ronald Halbgewachs	Sandia National Laboratories	USA
Neil Halliday	Sophos Plc	UK
Cynthia Hanlon	Fidelity Investments	USA
Philip Harris	Fidelity Investments	USA
Mike Hills	Ministry of Defence (Army)	UK
Robert Hinten	Environmental Protection Agency	USA
Richard Ho	IBM Corporation	USA
Steve Holstein	Virus Bulletin	USA
Tony Hopkins	Kingston University	UK
Zoltan Hornak	Technical University of Budapest	Hungary
Frank Horwitz	Reflex Inc	USA
Tore Hoyem	Norske Shell A/S	Norway
Mikko Hypponen	Data Fellows Ltd	Finland

J

Craig Jackson	Datawatch Corporation	USA
Portia Jackson	Department of Veterans Affairs	USA
Richard Jacobs	Sophos Plc	UK
Ken Jaworski	Detroit Edison	USA
Joy Johnson	Intel	USA
Martin Jones	Computacenter	UK

K

Natalya Kasperskaya	KAMI	Russia
Norkio Kato	Jade Corporation Ltd	Japan
Tapio Keihanen	MikroPC Magazine	Finland
Greg Kendig	AMP Incorporated	USA
Jeffrey Kephart	IBM	USA
Michal Kovacic	Alwil Software	Czech Republic
Eduard Kucera	Alwil Software	Czech Republic

L

Lawrence LaBella	Merrill Lynch	USA
Paul Lawrence	S&S International	UK
Orlton Lawrence	Toronto Dominion Bank	Canada
Dave Leigh	Staffordshire University	UK

Darren Leonard	Alternative Computer Solutions	USA
Sharon Lettvin	OverByte Corporation	USA
Myron Lewis	Norman Data Defense Systems Inc.	USA
Vince Lombardi	Fleet & Industrial Supply Center	USA
Tina Lombardi	McAfee Associates	USA
Greg Lutz	OverByte Corporation	USA
Merrill Lynch	Chevron	USA
Sherry Lynch	Detroit Edison	USA

M

Jafar Muhammad Mamun	El-Mamun Ent.	Ghana
Henry Matos	The Segal Co.	USA
Jack McAulay	University of Eninburgh	UK
Tom McCllough	IBM Corporation	USA
Svein Meland	Allianse Informasjonssystemer	Norway
Alison Millar	Standard Life Assurance Co	UK
Mahesh Moorthy	IBM Corporation	USA
Seiji Murakami	Jade Corporation Ltd	Japan
Akihiko Muranaka	Jade Corporation Ltd	Japan

N

Sean Nabeau	ESaSS GmbH	The Netherlands
Carey Nachenberg	Symantec	USA
Kurt Natvig	Norman Data Defense Systems	Norway
Senthil Nathan	IBM	USA
Kevin Norris	Allied Irish Bank Group	Ireland

O

Martin Odvarko	Alwil Software	Czech Republic
Jorgen Olsen	DOU, Odense University	Denmark
Ernst Oud	Crypsys	The Netherlands
Ian Overton	GPT Ltd	UK

P

Therese Padilla	Command Software Systems	USA
Charles Parker	IBM	USA
Keith A Peer	Central Command Inc.	USA

Chen Yi-Pen	Trend Micro Devices Inc	Taiwan
Manfred Philipowsky	Deutsche Telekom AG	Germany
Donald Phipps	The Clorox Company	USA
Andre Pitkowski	Compusul	Brazil
Kirstin Police	Alternative Computer Solutions	USA
Alastair Port	KPMG	UK
Edward Pring	IBM	USA
Judy Pruitt	Alternative Computer Technology	USA
Richard Ramza	Deere & Company	USA

R

Francisco Ramos	Microasist	Mexico
Carole Reid	Defense Commissary Agency	USA
Charles Renert	Symantec	USA
Sally Riordan	Cybec Pty Ltd	Australia
Eduardo Rios		Mexico
Frank Roache	BBC World Service	UK
Jake Roddy	Defense Contracts Audit Agency	USA
Rhonda Rosenbaum	IBM	USA
Marvin Ruppert	National Futures Association	USA

S

Alla Segal	IBM	USA
Marek Sell	Apexim Co S.A	Poland
James Shaeffer	Reflex Inc	USA
Lee Jieh-Sheng	Trend Micro Devices Inc	Taiwan
Risto Siilasmaa	Data Fellows Ltd	Finland
Lester Simons	Lloyd's Register	UK
George Sneddon	Sophos Plc	UK
Alan Solomon	S&S International	UK
Susan Solomon	S&S International	UK
David Stanley	Royal Air Force	UK
Philip Statham	CESG/GCHQ	UK
Ken Stieers	Ontrack Computer Systems	USA
Howard Stone	BBC World Service	UK
Tom Stormer	Alternative Computer Solutions	USA

Robert Stroud	Cybec Pty Ltd	Australia
Mary Sullivan	Alternative Computer Technology	USA

T

Gabriel Takami	S&S International	Mexico
Allen Taylor	Customs Service	USA
Tony Tedridge	Minsitry of Defence (Army)	UK
Howard Thaw	ESaSS GmbH	The Netherlands
Peter Theobald	Quantum System Software	India
Andrew Tilling	Royal Military Police Info Systems	UK
Shelagh Todd	NationsBank Services Inc	USA
Ruben Tovar	S&S International	Mexico
Howard Townsend	Virginia Housing Development Auth.	USA
Alan Tremblay	Statistics Canada	Canada

V

Gert van de Nadort	Crypsys	The Netherlands
Chan Vo	The New York Times Company	USA

W

Reed Ward	Ontrack Computer Systems	USA
Simon Webber	Defense Research Agency	UK
Joe Wells	IBM	USA
Ian West	Royal Air Force	UK
Simon John Williams	Bass Brewers Ltd	UK
Denis Woods	Renaissance Contingency Services	Ireland
Simon Woolley	Sophos Plc	UK

EXHIBITORS

Alwil Software

Command Software Systems

Computer Security Consultants Inc

Cybec Pty Ltd

Eliashim Microcomputers Inc

ESaSS GmbH

IBM Corporation

McAfee Associates

Norman Data Defense Systems

Ontrack Computer Systems

Reflex Inc

RG Software Systems

S&S International

Sophos Plc

Virus Bulletin Ltd

ANTI-VIRUS PRODUCT DEVELOPERS

The following is a list of anti-virus product developers known to Virus Bulletin in August 1995. Its accuracy and currency are not attested and cannot be guaranteed.

Alwil Software, Prubezna 76, CS-100 00 Prague 10, Czech Republic
Tel +42 278 22050, Fax +42 278 22553

Product: *Avast!*

Carmel Software Engineering, PO Box 25055, Hamachshev Ltd Hahistradrut Av 20, Haifa, Israel.
Tel +972 4 416976, Fax +972 4 416979

Product: *TNT Anti-virus, Carmel Anti-virus*

Cheyenne Software Inc., 3 Expressway Plaza, Roslyn Heights, New York 11577, USA.
Tel +1 516 629 4459, Fax +1 516 484 3446

Product: *Inoculan*

Command Software, Suite 500, 1061 E Indian Town Road, Jupiter, FL 33477, USA.
Tel: +1 407 575 3200, Fax +1 407 575 3026

Product: *F-PROT*

Computer Security Engineers Ltd, Postbus 85 502, 2508 CE Den Haag, The Netherlands.
Tel +31 70 362 2269, Fax +31 70 365 2286

Product: *PC Vaccine Professional*

Cybec Pty. Ltd., 133 Alexander Street, Crows Nest, NSW 2065, Australia.
Tel +61 2 9965 7216, Fax +61 3 2438 2335

Product: *VET*

Cybersoft, 1508 Butler Pike, Conshohocken, PA 19428, USA.
Tel: +1 610 825 4748, Fax: +1 610 825 6785

Product: *V-FIND*

Data Fellows Ltd, Paivantaite 8, 02210 Espoo, Finland.
Tel: +358 0 478 444 , Fax: +358 0 478 44599

Product: *F-PROT*

Datawatch Corporation, 234 Ballardvale Street, Wilmington, MA 01887, USA.
Tel: +1 508 988 9700, Fax: +1 508 988 0105

Product: *Virus XScreen Link, Virex for Macintosh*

Diamond Chip Computers CC, 2nd Floor, 4 Susmann Avenue, Blairgowrie, Randburg, Johannesburg, South Africa.
Tel: +27 11 886 3131, Fax: +27 11 86 3331

Product: *ViruGuard*

EliaShim Microcomputers Ltd., PO Box 9195, Haifa 31091, Israel.
Tel: +972 4 516111, Fax: +972 4 528613

Product: *Virusafe*

EMD Enterprises, 6 Cardinal Drive, Glenrock, PA 17327, USA.
Tel: +1 717 235 4261, Fax: +1 717 235 1456

Product: *EMD Armor Plus*

ESaSS BV, Saltshof 10-18, 6604 EA Wijchen, The Netherlands.
Tel: +31 8894 22282, Fax: +31 8894 50899

Product: *Thunderbyte*

Frisk Software International, PO Box 7180, 127 Reykjavik, Iceland.
Tel: +354 1 617273, Fax: +354 1 617274

Product: *F-PROT*

H+BEDV Datentechnik GmbH, Olgastrasse 4, D-88069 Tettnang, Germany.
Tel: +49 7542 93040, Fax: +49 7542 52510

Product: *AVScan*

Hunix Ltd, Budafoki ut. 57/A, 1111 Budapest, Hungary.
Tel: +36 1 186 7408, Fax: +36 1 186 7408

Product: *Virus Buster*

IBM, TJ Watson Research Centre, PO Box 218, Route 134, Yorktown Heights, NY 10598, USA.
Tel: +1 914 945 3000, Fax: +1 914 945 2141

Product: *IBM AV*

Information Security Services Inc., 1211 Distribution Way, Beltsville, MA 20705, USA.
Tel: +1 301 470 2500, Fax: +1 301 470 2507

Product: *Detect Plus*

Intel Corp., 5200 N E Elam Young Parkway, Hillsborough, OR 97124, USA.
Tel: +1 503 629 7354, Fax: +1 503 629 7580

Product: *LAN Desk Virus Protect*

Iris Software, 6 Hamavo Street, Givataim 53303, Israel.
Tel: +972 3 571 5319, Fax: +972 3 318 731

Product: *Iris AV*

Leprechaun Software Pty. Ltd., 75 Redland Bay Road, Capalaba, Queensland 4157, Australia.
Tel: +61 7 823 1300, Fax: +61 7 823 1233

Product: *Virus Buster*

McAfee Associates, 2710 Walsh Avenue, Suite 200, Santa Clara, CA 95051-0963, USA.
Tel: +1 408 988 3832, Fax: +1 408 970 9727

Product: *Virus-Scan, V-Shield*

Norman Data Defense Systems, Suite 201, 3028 Javier Road, Fairfax, VA 22031, USA.
Tel: +1 703 573 8802, Fax: +1 703 573 3919

Product: *Norman Virus Control*

Panda Systems, 801 Wilson Road, Wilmington, DE 19803, USA.
Tel: +1 302 764 4722, Fax: +1 302 764 6186

Product: *Panda Pro, Bear Lock, Dr Panda Utilities*

Peter Hoffmann Service GmbH, Friedrichsplatz 12, 68165 Mannheim, Germany.
Tel: +49 621 4311 901, Fax: +49 621 444 273

Product: *PC Safe*

Reflex Magnetics Ltd, 31-33 Priory Park Road, Kilburn, London, NW6 7OP, UK.
Tel: +44 71 372 6666, Fax: +44 71 372 2507

Product: *DiskNet*

RG Software Systems, 6900 East Camelback, Suite 630, Scottsdale, AZ 85251, USA.
Tel: +1 602 423 8000, Fax: +1 602 423 8389

Product: *Vi-spy*

Safetynet Inc., 140 Mountain Avenue, Springfield, NJ 07081, USA.
Tel: +1 908 851 0188, Fax: +1 908 276 6575

Product: *VirusNet-Pro*

SA Software, 28 Denbigh Road, London, W13 8NH, UK.
Tel: +44 81 998 2351, Fax: +44 81 998 7507

Product: *PC Immunize II*

S&S International Plc, Alton House, Gatehouse Way, Aylesbury, Bucks HP19 3XU, UK.
Tel: +44 296 318700, Fax: +44 296 318777

Product: *Dr Solomon's Toolkit*

SikkerhedsRadgiverne ApS, Knabrostraede 20, 4th Floor, DK-1210 Copenhagen K, Denmark.
Tel: +45 3332 3537, Fax: +45 3332 3547

Product: *ASP Integrity Toolkit*

Softcraft AG, Niederwiesstrasse 8, CH-5417 Untersiggenthal, Switzerland.
Tel: +41 56 28 11 16, Fax: +41 56 28 11 16

Product: *VIP*

Sophos Plc, 21 The Quadrant, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YS, UK.
Tel: +44 1235 559933, Fax: +44 1235 559935

Product: *Sweep, Vaccine*

Symantec Corporation, 10201 Torre Avenue, Cupertino, CA 95014, USA.
Tel: +1 408 725 2762, Fax: +1 408 253 4992

Product: *Norton Anti-virus, CPAV*

The Davidson Group, 20 Exchange Place, 27th Floor, New York, NY 10005, USA.
Tel: +1 212 480 1050, Fax: +1 212 422 1953

Product: *Vaccine*

Thompson Security Software, PO Box 669306, Marietta, GA 30066, USA.
Tel: 0101 404 971 8900, Fax: +1 404 971 8828

Product: *Doctor*

Trend Micro Devices Inc., 1F #28 Li-Shui Street, Taiwan, Republic of China.
Tel: +886 2 312 0191, Fax: +886 2 341 2137

Product: *PC-Cillin, Stationlock*

Visionsoft, Unit C7, Enterprise Way, Five Lane Ends, Idle, Bradford, West Yorkshire BD10 8EW, UK.
Tel: +44 274 610503, Fax: +44 274 616010

Product: *SmartScan, Immunizer*

THE ANTI-VIRUS STRATEGY SYSTEM

Sarah Gordon

Command Software Systems, Inc, 1061 E. Indiantown Road, Suite 500, Jupiter, FL 33477, USA
sarah@dockmaster.ncsc.mil

ABSTRACT

Anti-virus protection is, or should be, an integral part of any Information Systems operation, be it personal or professional. However, our observation shows that the design of the actual anti-virus system, as well as its implementation and maintenance, can range from haphazard and sketchy to almost totally nonfunctional.

While systems theory in sociological disciplines has come under much attack, it has much to offer in the management of integration of technological applications into daily operations. We will examine the 'anti-virus' strategy (Policy, Procedure, Software [selection, implementation, maintenance]), focusing on areas where the 'system' can fail. We will address this interaction from a business, rather than a personal computing, point of view.

The Anti-Virus Strategy System will examine anti-virus strategies from a Holistic General Systems Theory perspective. By this, we mean that we will concern ourselves with the individual parts of the system, their functionality, and their interaction. We will draw from various IT models specifically designed to provide a holistic, forward-thinking approach to the problem, and show that for our strategy to flourish, we must concern ourselves with the system as a whole, not merely with its individual components.

1 INTRODUCTION

Computer virus. System failure. These words bring to mind a computer system brought to its knees - data corrupted and time wasted. Is this an accurate picture? We hear arguments against investing in virus protection: 'Viruses are mythical. Your chances of getting hit by one are pretty rare.' Others tell us anti-virus software is a necessity: 'Viruses can cost your company a lot of money. Better safe than sorry.' What are we to believe?

Let's assume that you don't have any anti-virus software. If you are 'hit' by a virus, the cost will be proportional to the value of your data and the value of your time. Independent studies [1] have shown that this cost can be quite high, depending on these factors as well as environmental factors such as how many computers you have (Note: If your data is of little or no value, and if your time is worthless, then you can well afford not to have an anti-virus strategy).

We will assume here that your data is worth something to your company, and that your time also has a significant value. In this case, you will want to protect your computer system from viruses. We will concede for the purists among us that not all viruses are intentionally harmful, but stipulate that intentional harm is not requisite for actual harm. For our purposes, allocating disk space and CPU time and/or modification of files without knowledge and consent (implied or otherwise) constitutes damage, as do deliberate or unintentional disruption of work, corruption of data and the lost time mentioned earlier. Basically, we are saying viruses are bad and we want to protect against them (there may be some wonderful new virus out there in development that can help us, but that is beyond the scope of this paper).

Fortunately, we are in luck. The very thing we need already exists: software, which will detect 100 percent of viruses listed by the Wildlist [2] as being known to be in the wild. In tests run against a library matched with the Wildlist, several programs were capable of detecting all such viruses. The necessity of detection of 'lab' viruses is another matter, and will not be covered at this time, although it is addressed in [3].

Since we have such software, we should have no problems. However, there are problems. Something is wrong. Before examining the sources of the problem, a few comments on definitions we will be using are in order.

2 DEFINITIONS

The definitions used here are pretty generic, and are adapted for use in an interdisciplinary approach to the problems addressed. Some among us would argue that the systems movement was born out of science's failures [4], but in this paper, we take the view that General System theory is a child of successful science, and as most children, it sees things through optimistic eyes. We have specifically avoided in-depth discussion of categorical schemes, generalizations, and other commonly used 'tools' of General Systems thought, and have focused instead on the simplest of the simple. The ideas in this paper are drawn heavily from very basic works in systems theory. They are not new ideas, but it is our hope that their application to the management of security and computer viruses will help us identify some of the problems we may be overlooking.

2.1 GENERAL SYSTEMS THEORY

A system is a set, or group, of related elements existing in an environment and forming a whole. Systems can be made up of objects (computers), subjects (your employees) and concepts (language and communication); they can be made up of any one or more of these elements. There are 'real systems' (those which exist independent of an observer), and 'conceptual systems' (those which are symbolic constructs). Our system, 'The anti-virus strategy system', is not so different from many others, in that it is composed of all three elements: computers (objects), people (subjects) and concepts (policies and ideas). Each of these systems has its own subsystems. For example, your system of networked computers consists of individual computers. These computers are comprised of yet more subsystems; microprocessors, resistors, disk drives, etc. Our system consists of both real and conceptual subsystems. A system can also be said to be a way of looking at the world, or a point of view [5].

Concepts, laws, and models often appear in widely different fields [6] based upon totally different facts. This appears to be at least in part due to problems of organization, phenomena which cannot be resolved into local events, and dynamic interactions manifested in the difference of behaviour of parts when isolated or in higher configurations. The result is, of course, a system which is not understandable by investigating their respective parts in isolation. One reason these identical principles have been discovered in entirely different fields is because people are unaware of what those in other disciplines are doing. General Systems theory attempts to avoid this overlap in research efforts.

There are two main methodologies of General Systems research; the empirico-intuitive and the deductive theory. The first is empirical, drawing upon the things which regularly exist in a set of systems. It can be illustrated fairly easily, but lacks mathematical precision and can appear to the 'scientist' to be naïve. However, the main principles which have been offered by this method include differentiation, competition, closed and open systems, and wholeness – hardly naïve or worthless principles. The second method, basically, can be described as 'the machine with input', defined by a set 'S' of internal states, a set 'I' of input and a mapping '*f*' of the product I x S into S (organisation is defined by specifying states and conditions). Self-organising systems (those progressing from lower to higher states of complexity, as in many social organisations) are not well suited to this approach, as their change comes from an outside agent. Our anti-virus strategy system is such a system and for this reason we will use the empirico-intuitive methodology.

Classical system theory uses classical mathematics to define principles which apply to systems in general or to subclasses. General System theory can be called the doctrine of principles applying to defined classes of systems. It is our hope that we can stimulate thought on how already-known principles can help us in managing our anti-virus protection by examining the system as a whole.

2.2 HOLISM

Our definition of holism, drawing where appropriate from the medical profession, is health-oriented, and focuses on maintaining and improving the existing health of the system. It does not focus on disease and illness. It is interesting to note that, while we have many terms that relate to compromised and infected systems, we do not seem to have many terms relating to 'well' computers. Holism operates under the assumption that the open system possesses an innate organising principle, with the interdependence of the parts having an effect on the total system health. Holism views symptoms of distress as signalling disharmonic conditions, from which we can learn how to adjust the system (feedback); it is open to a variety of approaches for attaining balance. The focus of holism is heavily slanted toward the correction of causal factors, not symptomatic relief. Thus, the role of the holistic practitioner is to facilitate the potential for healing [7].

3 ANTI-VIRUS STRATEGY SYSTEMS

Where do our anti-virus strategy systems fit in this picture? We hope to explore some answers to that question by first examining the components of our model system. Keep in mind, however, that the goal of this paper is not to provide you with answers, but rather to stimulate new ways of thinking about the problems we face daily.

3.1 COMPONENTS

Each of the components in Diagram 1 contributes to the overall health of the system. Conversely, each can contribute to the illness of the system. For instance, our computer can contribute to the health of the system by functioning properly. If the hard drive crashes, a disharmonic condition is introduced. Our managers contribute to the overall well-being of the system, as long as they perform correctly. However, if one of them intentionally or unintentionally infects a computer with a virus, he or she contributes to the illness of the system. Our software contributes to the wellness by keeping employees reassured, and by keeping viruses out. If it is disabled by an employee desirous of more speed upon boot, or if it does not do its job in virus detection, it contributes to the illness or chaos in the system. There are other factors not shown, as the anti-virus strategy system model does not stop at the boundary of the company. The model includes your Internet service provider, virus writers, makers of electronic mail front-ends, anti-virus product tech support people and more. For the purposes of this paper, we must draw an artificial boundary. We mention the rest to give you food for thought, and to illustrate that boundaries are not static.

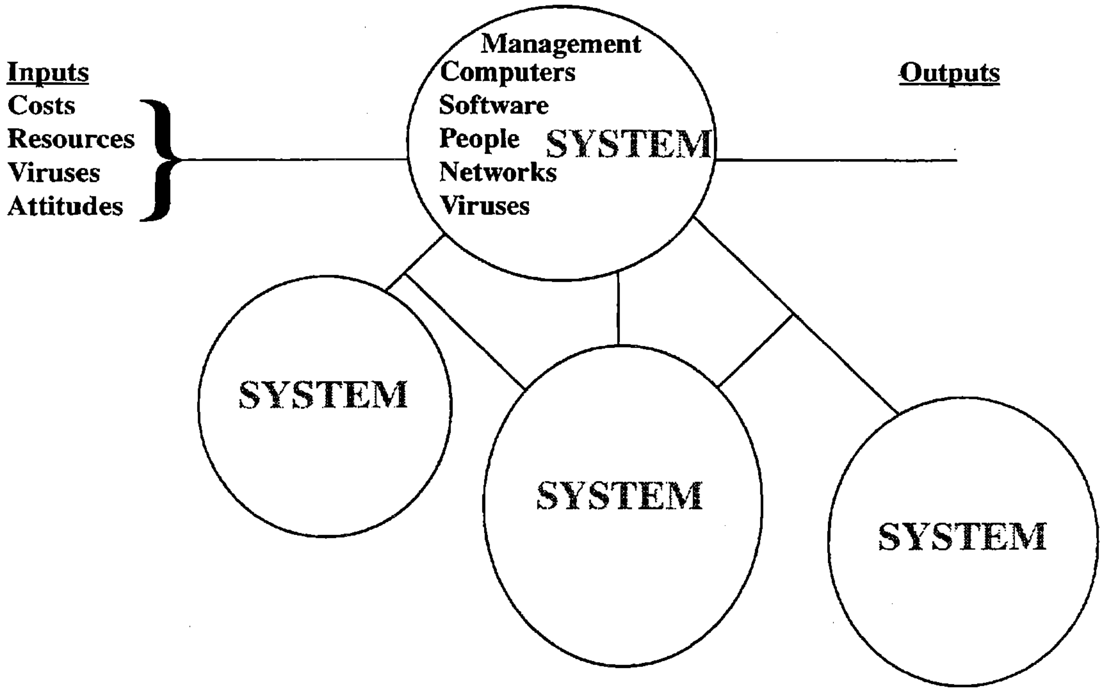


Figure 1. Anti-virus Strategy System - The Environment

3.2 PROGRAMS POLICY AND PROCEDURES *(SELECTION, IMPLEMENTATION AND MAINTENANCE)*

Where do we begin in examining the interaction of our chosen system elements? Let's start with the software selection. Anti-virus software is selected based on a wide number of criteria (8). While some of these criteria are beneficial, several are counterproductive at best (9). We need to be aware of exactly how our company's software is being chosen, and not leave this vital aspect of software selection up to people who do not have the experience or expertise to make a selection that will maximize your organisation's protection against viruses.

Does your anti-virus software detect all of the viruses which are a real threat to your organisation? Before you glibly answer yes, you should recognise that all products are far from created equal, and that even the best products will not achieve this goal if not properly maintained. Consider the following:

'When asked what happens to two blocks of copper initially at different temperatures left alone together in an insulated container, students will reply that the blocks will come to the same temperature. Of course, if asked how they know, they usually say "Because it is a law of nature"...the opposite is true...it is a law of nature because it happens. [10].'

Apply this to your anti-virus software. Does it catch viruses because it is anti-virus software? If so, you can depend on it, as its name defines what it is. But, if you even loosely apply this concept, you will see that it is anti-virus software because it catches viruses – and if it does not, then what does that make it?

Remember the following quote:

'If you call a tail a leg, how many legs has a dog?'

'Five?'

'No, Four. Calling a tail a leg doesn't make it a leg' [11]

Maintenance of your software is another critical issue. Maintenance refers not to the upgrade, but to the maintaining of the software on a daily basis. What does it require to run? Are you supplying what it needs to live? Or is it merely surviving? Does it have adequate memory, power, disk space to run optimally and lessen the chance your employees will disable it? Is it in an environment free from other programs which may hinder its performance? If you cannot answer yes to these questions, you are not providing an environment for this element of your strategy system which will allow it to remain viable. It will not survive. Like living systems, the anti-virus strategy system requires a favorable environment, else the system will adapt. Unfortunately, in the case of this system, adaptation can mean software becoming disabled by the user component of the system, or overridden by a competing software component. All this, and we have not even added viruses which by design cause a problem to the system by the introduction of instability.

Even if you have the best anti-virus software, and are running it optimally, there can still be problems. Software is just one part of the strategy system. Policies and procedures play an important role in the overall strategy. Even the viruses we mentioned earlier play a part in this system. Then there are the least predictable aspects of the system, the human beings. How complex is this system? How much should we expect the people involved to understand?

Ackoff defines an abstract system as one in which all of the elements are concepts, whereas a concrete system is one in which at least two of the elements are objects [12]. As you can see, our system is concrete. It is also by design an open system, one into which new components may be introduced. Some of these components are by nature 'unknown' (i.e. actions of people, how software may react, viruses which may appear).

When these components are introduced, we have to consider first how they behave on their own. Next, we have to consider how they would behave in combination with any and/or all of the other elements. Finally, we have to consider how 'things' in general will be if neither of the objects are present. In its most simple form, a two-part system would require four equations, but of course, you can see that as the number of elements increases, the number of interactive equations grows by leaps and bounds [Table 1].

	Linear Equations			Nonlinear Equations		
Equation	One Equation	Several Equations	Many Equations	One Equation	Several Equations	Many Equations
Algebraic	Trivial	Easy	Essentially impossible	Very difficult	Very difficult	Impossible
Ordinary differential	Easy	Difficult	Essentially impossible	Very difficult	Impossible	Impossible
Partial differential	Difficult	Essentially impossible	Impossible	Impossible	Impossible	Impossible

Table 1. [From [5]] - Introduction of Elements

One of the systems theory approaches we can draw from here to help illustrate the problem comes from what is sometimes called the Square Law of Computation. This means basically that unless you can introduce some simplifications, the amount of computation involved in figuring something out will increase at least as fast as the square of the number of equations. Consider all of the interactions between humans, computers, and software, and you will see why it is impossible to precisely calculate what the results of all of those interactions will be. We cannot even measure them. In other words, you cannot possibly anticipate all of the problems you will encounter in trying to keep your company's data safe from viruses, because you cannot possibly calculate the interactions which will occur once you begin trying to formulate a strategy. Needless to say, these interactions create 'problems'.

If we examine our anti-virus strategy in various ways, we may be able to see things more clearly. Another helpful way in which we can view our system is as an expression, such as the terms of a set. For instance, the notation:

Let x stand for marriage

Let y stand for carriage

Let z stand for bicycle

The set [x,y,z] is simple enough for anyone to understand. Using names in sets takes us to the more complex:

[The look on your face when you saw your first child, a proof that Vesselin Bontchev is not the Dark Avenger, an atom of plutonium]; wherein the first no longer exists (or possibly never did); the second has not yet existed, and the third is out of reach of the common man.

If you were to be asked for the meaning of the ... in the set [Alan, Dmitry, Fridrik...] would you say the ... represented men's names? Names of programmers? Names of programmers who make anti-virus software? Names of people not from the United States?

What is the rule for determining the meaning of what is unstated? Is there some unwritten heuristic of which your employees are not aware? What is the meaning of the three dots in our set?

This has a particular application to policy. Users can easily understand, 'Do not turn the computer off if you find a virus'. Can they as easily understand, 'Do not reset the computer if you find a virus'? Can they understand, 'In the event of a suspected virus, call the administrator or take appropriate action'? What *is* a suspected virus? Is it any time the computer system seems to act strangely? Is it only when the letters fall off? After all, that's what viruses do, right? What is appropriate action? [Turn off the computer, Call your supervisor, Reboot the computer, ...] What is the meaning of the ... in this set?

4 VARIATIONS ON A THEME

How well are our strategies doing? As pointed out early on, not very well. Why not? To help answer that question, next we will examine the problems of our strategy using the concept of variation. We recognise the duality of variables as they relate to information processing; the significant values which variables acquire at the two extremes of their respective spectra. Specifically, in order for a system to continue to thrive, information must be processed. Disorder, uncertainty, variety – all must shift from high to low [Table 2].

Disorder, Uncertainty and Variety: Entropy and the Amount of Information Processed

High	Disorder	Low
High	Uncertainty	Low
High	Variety	Low
Large	Number of Alternatives	Small
Small	Probability of an Event	Large
Low	Regulation and Control	High

Table 2 - Predictable Output

The probability of particular events follows by decreasing from small to large. The amount of regulation and control increases from low to high. We become increasingly sure of the output of our systems [13]. However, viruses introduce a form of disorder with which the human components of our systems are not intimately familiar. While the probability of infection can be calculated mathematically [14], we are unable to calculate the probability of other events related to viral infections[15]. In what ways does this introduced unfamiliarity manifest itself? One manifestation is the appearance of problems.

We typically try to solve most of these problems deductively, to determine the reason for a variation between design and operation or design and implementation. This approach is doomed to failure because it places the blame on the subsystems. We attempt to 'restore to normal' instead of redesigning our system. We formulate plans based on incorrect, incomplete or obsolete assumptions. We neglect to factor in spillover effect, that is, the unwanted effect which actions in one system can have in another. Improving an isolated system may seem the epitome of system integrity. You can have your pure clean computer. Of course, it is virtually useless, unconnected to the rest of the world. Or, perhaps it is the solution. Isolated perfect machines. This would probably create a dissatisfied workforce, however, which would ultimately impact business negatively. In the case of anti-virus strategy, 'spillover' takes on many new dimensions – as many as the human beings with which our machines interface. Can you control all of the aspects of this system? You cannot.

Another factor to consider is the size and extent of our system. Further insight may be gained by considering what is sometimes referred to as the generalised thermodynamic law, which states that the probable state is more likely to be observed than the less probable. While this may incite the physicists among us, it has two parts which correspond to the first and second law of thermodynamics. The first law is hardly worth mentioning (physical reason), but the second is of interest to us. We should be concerned with the limited power of observers when viewing large systems. In other words, we cannot expect our managers to be in every place at once, knowing what is going on with every system, every employee. The concept of boundaries can be used to help solve this problem, but their definition is beyond the scope of this paper [16].

4.1 SYSTEM FAILURE AND MEASUREMENT

We say the system is failing for three reasons. It is not performing as intended. It is producing results other than expected. It is not meeting its goal. The objective is **NO VIRUSES**. However, in addition to often neglecting to define what 'no viruses' actually means, we are frequently unaware of how 'no viruses' can mean different things to different people. Not performing as intended could mean it finds some viruses but not all, or it finds all but only removes some. Unexpected results could mean it crashes 1 out of every 6000 machines, or produces system degradation you did not anticipate (if this is the case, does the fault really lie with the product for producing the degradation or you for not anticipating?) Not meeting its goal most likely means failing to keep out viruses. However, to some people, this is a different goal from 'no viruses'.

How is this possible? Isn't 'no viruses' a simple concept? In a word, no. When there is a malfunction, i.e. a virus is found, the natural tendency is to look for the cause within the system. We tend to blame the problem on the variation of the system from its 'desired' behaviour. It could be the fault of the program, the employee, the policy. We tend to blame the program as it is the part of the system most closely identified with the failure as immediately perceived. However, consider for a moment that, to your employee, 'no viruses' means simply that. No viruses are found. Following that line of thought, finding 'no viruses' would be a system success – that is, until it brought your operation to a halt. You see, to some people, 'no viruses' means that none are seen or observed, and not that none are actually operational in the system. We plan grandiose policies and procedures around finding a virus and make no space for 'no viruses' as a possible failed variation. If you find 'no virus', you need to be very sure it is not due to your employees disabling your software, or your software not finding the virus.

Many system 'improvements' are possible which in reality doom the system. Faulty assumptions and goals are often at the root of this problem. For instance, it is obvious that all of your computer workers must, under dire penalty, refrain from bringing disks from home into your office. You implement this policy. You assume they will comply. Your goal is compliance, not 'no viruses'. If the goal was 'no viruses', you would be forced to be more realistic.

Consider the following two statements:

'We have clean, working computers and by not bringing in software, we can keep them that way. It will save us all a lot of time, and effort!'

'If you bring in disks, you will probably infect our office computers. It will cost us all a lot of money.'

In the first instance, the focus is on the well machine. Everyone wants well machines. People like to be part of winning teams, and participate in things that are nice.

In the second, the focus is on the sick machine. None of your people would have viruses on their home computers. So, this must not apply to them. And if they do break the rule, you have already set them up to be afraid to tell you. After all, they don't want to cost you a lot of money and they certainly don't want to be known as the culprit for infecting the office computers.

How do we measure the performance of our anti-virus strategy system? Not very well. If we find some viruses, we say it's working. If we don't find any viruses, we say it's working. In some cases, you can apply 'we say it's not working' to these same sentences. There is no standard way in which we measure the success of the entire system. Only in the act of being out of control will the system be able to detect and bring back the control.

5 CONCLUSION

The systems approach proposed here is a 'whole system' optimization. Think of it as the configuration of a system which will facilitate optimal performance. There exists, of course, a dilemma, in that at some time suboptimization may be necessary, or even the only possible approach. An approximation which is used may be a great deal better than an exact solution which is not [17]. Nevertheless, our model will attempt to show ways to optimize system performance.

Models are how we express things we want to understand and possibly change, designed in terms of something we think we already understand. Models sometimes present problems when you try to translate them into real world activities. With this in mind, I would like to suggest a simple model which may help us begin to find ways to find a solution to the problem of designing a workable anti-virus strategy.

'Models should not so much explain and predict as to polarize thinking and pose sharp questions.' [18]

Using a holistically modelled approach, we would strive to maintain the existing health of the system. This assumes we have a healthy system to begin with. This requires you not depend on your belief that your software is correctly installed and operational, and that your employees know how to use it and are using it, and that your equipment is functional, and that your policies are correct and being followed... It requires that you actually take it upon yourselves to designate people to ensure that your system is optimal to begin with. If you are not willing to do this, you cannot expect to restore the system to health. The focus should shift from 'blame' to 'responsibility'. This may require investment on your part. You may need to update equipment. You may need to train employees. You may need to purchase software. You may need to subscribe to publications which can keep your employees up to date on trends in virus and security matters.

You will need to monitor feedback between various aspects of your anti-virus strategy system. We have not discussed feedback at any great length in this paper, due to the number of elements of the system and the complexity of the feedback. However, using the empirico-intuitive General Systems theoretical approach defined earlier in this paper, you should be able to determine the sorts of feedback which are required to keep your system functioning optimally. If there is NO feedback, you can rest assured your system will fail. Lack of feedback produces entropy. In simple terms, entropy can be called the steady degradation or disorganization of a society or a system. This is not what you want for your system. You want to move the system into organisation and order, high rates of probability and certainty. As we discussed earlier, this happens when information is processed. The information can be communication of any type between any elements of the system.

Our current focus seems to be on the existing illnesses in our systems. If open systems indeed, as suggested, possess an innate organising principle, perhaps we should be paying more attention to what the elements of our systems are telling us. We could learn the sorts of information required to maintain organised reliability. We could learn the amount and types of feedback required to process information optimally, and to keep the system both desirably adaptive and from adapting negatively. We must examine our systems as a whole, including all of the parts, as best we can, to determine what the elements and the system are telling us. In the case of our anti-virus strategy systems, we have yet to determine what that message is. Many of us have not even yet defined the elements of the system, the system boundaries, or the goal of the system.

It is clear that there are disharmonic conditions in the 'Anti-virus strategy systems' of most companies; if there were not, no one would be attending this conference or reading this paper. It is also clear that the way we traditionally approach these problems is not working. We have been using these approaches for a long time, and the problems are not going away. Drawing from the holism model, one thing we can do is examine causal factors, instead of focusing on symptomatic relief. We need to examine more closely the interdependence of the parts of our system, and as security professionals, should facilitate the potential for healing our systems. It is hoped that some of the ideas mentioned in this paper can provide a starting point for this.

The author would like to thank Louise Yngstrom, University of Stockholm, for late night chats on System Theory, above and beyond the call of even academic duty.

BIBLIOGRAPHY

[1] 'Virus Encounters, 1995: Cost to the World Population'. Testimony, House Subcommittee on Telecommunications and Finance, Tippett, Peter, June 1993.

[2] 'The Wildlist'. Maintained by Joe Wells.

[3] 'Real World Anti-Virus Product Reviews and Evaluation'. Gordon, Sarah and Ford, Richard, *Proceedings of Security on the I-Way*, NCSA, 1995.

[4] 'An Introduction to General Systems Thinking', p.3, Weinberg, Gerald. John Wiley and Sons, 1975.

[5] 'An Introduction to General Systems Thinking', p.51, Weinberg, Gerald. John Wiley and Sons, 1975.

[6] 'General Systems Theory: Foundations, Development, Applications', pp.xix-xx, Revised Edition, von Bertalanffy, Ludwig. George Braziller, Inc, 1980.

[7] 'Health Promotion Throughout the Lifespan', Edelman, Carole and Mandle, Carole. Mosby, 1994.

[8] 'Guide to the Selection of Anti-Virus Tools and Techniques'. Polk, T. and Bassham, L. NIST Special Publication 800-5. NIST, December, 1992.

[9] 'Real World Anti-Virus Product Reviews and Evaluation', Gordon, Sarah and Ford, Richard. *Proceedings of Security on the I-Way*. NCSA, 1995.

[10] 'Semantics, Operationalism and the Molecular-Statistical Model in Thermodynamics', Dixon, John and Emery, Alden. American Scientist, 53, 1965.

[11] Quote attributed to Abraham Lincoln.

[12] 'Applied General Systems Theory', p.39, Van Gigch. John P. Harper and Row, 1974.

[13] 'Applied General Systems Theory', Figure 2.2, Van Gigch. John P. Harper and Row, 1974.

[14] 'Directed Graph Epidemiological Models of Computer Viruses', Kephart, Jeffrey O. and White, Steve, R., *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, 1991.

[15] 'The Viability and Cost Effectiveness of an 'In the Wild' virus scanner in a Corporate Environment', Gordon, Sarah, 1995.

[16] 'Applied General Systems Theory', p.25, Van Gigch. John P. Harper and Row, 1974.

[17] 'The Development of Operations Research as a Science', pp.59-60, as cited in [4]. Ackoff, Russell. *Scientific Decision Making in Business.*

[18] 'Some Mathematical Models in Science', Kac, Mark. Science, 166 No. 3906 695, 1969.

BLESSINGS IN DISGUISE: BUILDING OUT OF DISASTER

Paul Ducklin

Sophos Plc, 21 The Quadrant, Abingdon, OX14 3YS, UK
Tel +44 1235 544037 · Fax +44 1235 559935 · Email duck@sophos.com

Imagine, for a moment, that there is such a thing as the 'average anti-virus expert'. Take him or her aside briefly, and start talking gently and generally about computer viruses. The chances are good, even if you are skilled in keeping a conversation running along lines of your choice, that the subject matter will veer rapidly towards the technical. You should not be surprised if the expert makes a sudden subroutine call to highly technical matters; you should be even less surprised if you find that the subroutine stack becomes lost, so that a return to the original topic of discussion is impossible.

Likewise, much of the literature published in the short lifetime of the anti-virus field is largely technical in content. Even documents which are supposed to be corporate anti-virus policies, written in 'plainspeak' and signed by Topmost Management, sometimes manage to lose themselves in a tangle of 'technobabble'. This can be hard to avoid if you are trying to describe the best way to navigate through a nightmare world infested with armoured, tunnelling, full-stealth, highly polymorphic, multi-partite, fast-infecting malware objects.

Sometimes, though, the problems which emerge from the technological computer virus battleground are decidedly plain. Often, the 'obvious' attack (the unsubtle, widely telegraphed, low-tech viral broadside) succeeds where deviously clever programming fails. It may be an old bromide [1], but we can hardly blame technology for the ongoing prevalence of viruses such as Form and Stoned [2]. In July 1994, another technically unremarkable virus succeeded globally, making a sudden appearance worldwide: Kaos4.

This paper is a case study of an attack by this virus on a large South African company. Managers, network administrators, and users alike were all surprised by the sudden appearance of the virus; they were even more surprised when the virus reappeared just as suddenly three months later. As you will probably guess, these were not innocent, defenceless victims – especially as the lessons learned after the first attack should at best have prevented the second, and at worst allowed it to be handled with ease.

However, as an internal survey has shown, this organisation's corporate anti-virus awareness has improved as a direct result of the Kaos4 incidents, and the risk of viral disaster in the future has been addressed, and reduced. It would probably be going too far to describe Kaos4 as a blessing in disguise for this company (and it would offer the virus itself a function and legitimacy it does not possess), but they have certainly managed to learn from their mistakes. As will become obvious, though, they did not learn quite as swiftly as they might have done; others will want to aim to do better, faster.

To maintain the privacy of the company studied here, we shall refer to them as 'The Company We're Studying, Limited', and abbreviate this name as 'TCWS'. And, before you smirk at their story, ask yourself if you are absolutely certain that it couldn't happen to you.

THE INTERNET SPEAKS

According to the Internet, this is what happened (messages have been edited to remove personal or commercial identifying information; errors are reproduced as they originally appeared):

```
Date:    Thu, 28 Jul 94 07:58:11 -0400
From:    ABC Anti-Virus Company <abc@def.ghi>
Subject: Virus warning (PC)

We have discovered an infected file which has been spread on Usenet
in the group 'alt.binaries.pictures.erotica'. The virus is called
Chaos4/ kohntark 697, and is a com/exe infector. No current scanners
seem to be able to detect it yet. A detector/disinfector routine is
available in the file 'abcdefg.zip', which has been uploaded to
several sites (wuarchive.wustl.edu, ftp.funet.fi,
ftp.informatik.uni-hamburg.de etc.)

Sincerely,
S. O. Meone
ABC Anti-Virus Company

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Date:    Tue, 30 Aug 94 01:04:37 +0400
From:    XYZ <xyz@uvw.rst>
Subject: Re; [News] KAOS? (PC)

Hi !

somebody@somewhere.com (Some Body) writes:

> I have been hearing about a new (?) virus called KAOS that
> has been transferred over the internet. Does any one have
> any info on it?

Any name of this virus is 'Kaos4'.

So far, verified reports or samples of this virus have been received
from the US, Austria, Norway and Finland.

It seems that the virus was distributed over Usenet, possibly in one
of the alt. groups.

The virus is not very remarkable - it is a 697 byte non-resident
COM/EXE infector, which contains the string 'KODE4 / Kohntark' (The
'o' has 2 dots above it). This string is not encrypted and can be
```

```
found with any text search utility.
The virus does not seem to have any specially interesting functions,
and does not contain any destructive code, so the problem is not as
serious as it might have been, but the virus might have non-
intentional side-effects, such as preventing a machine from booting
if it infects IBMBIO.COM/IBMDOS.COM on a machine running IBM DOS.

-XYZ

- - - - - - - - - - - - - - - -

Date:     Sat, 01 Oct 94 11:20:46 -0400
From:     Another Person <ap@pqr.stu>
Subject: Re: Re; [News] KAOS? (PC)

XYZ (xyz@uvw.rst) wrote:

> Any name of this virus is 'Kaos4'.
> So far, verified reports or samples of this virus have
> been received from the US, Austria, Norway and Finland.

and South Africa...

Cheers, Another Person
```

Ignore for the moment the obvious inconsistencies in the above messages; the bulk of the information explains what happened, and how the virus came to make momentary global headlines. From a viral point of view, Kaos4 was interesting because of a novel combination of factors:

- the virus was uploaded openly to the Internet, posted into a newsgroup, which acted as a vehicle to spread the virus rapidly across the globe
- the uploaded infected object was a commercial shareware program
- this program was a game
- the upload destination on the Internet was an unmoderated newsgroup better known for disseminating pornographic pictures
- the uploader made no attempt to conceal himself, acknowledged the upload and claimed that he had taken appropriate precautions, but had been let down by an anti-virus scanner which gave his upload a false bill of health.

MYTHS CONSOLIDATED

The nature of the Kaos4 distribution gave the virus multiple angles of importance in the media, and in tearooms across the corporate world. Unfortunately, the immediate lessons learned from it, judging from informal conversations with victims at TCWS during my consultancy immediately following their first attack, were not particularly useful. In some cases, they served only to perpetuate those inaccurate myths which arose years ago, when the computer virus first began to become a problem [3].

Computer games have long been singled out as 'dangerous' software, and many companies have banned games because of the viral risk they pose – rather than because their employees are usually taken on to perform tasks other than playing games at work. In fact, games may even serve a valuable purpose in business (indeed, I learned to type fast playing the curiously-named 'Lobster Sea Adventure', designed specifically to teach keyboard speed and accuracy). Worse still, suggesting that there is a specific type of 'dangerous' computer program serves to suggest that there are programs which are inherently 'safe', and therefore immune to viral infection. Since many file viruses will infect just about any executable file which comes their way, this is an unsafe myth.

Many companies also retain shareware and freeware on their list of 'dangerous' programs. Clearly, it is the means by which shareware is sometimes distributed that needs consideration, rather than the shareware itself. After all, a careless or unscrupulous dealer selling shrink-wrapped commercial software might easily use, infect, and re-package that software (and such cases are well-documented) – so singling out the shareware concept as virally risky simply serves to deflect attention from the facts.

The burying of Kaos4 in a pornographic newsgroup also obscured corporate risks, with managers and users who steered clear of such newsgroups tending to believe that this would keep them and their workgroups safe from the virus – in the same way, perhaps, that those who are not sexually promiscuous reduce their risk of HIV infection to negligible levels.

Unfortunately, the obvious analogy here is not the correct one; a more useful analogy would be the observation that because you and I are not thieves does not mean that our houses are, *ipso facto*, safe from burglars. Furthermore, once a virus is in the wild, its initial mode of distribution becomes relatively unimportant – once Kaos4 was out there, companies not even on the Internet were at risk from infected floppies, just as they would be with any other virus.

The fact that Kaos4 was connected with not one but **four** arenas often proscribed to corporate users – games, shareware, bulletin boards (or the Internet, seen by some as a giant global BBS) and pornography – has led some observers to conclude that the uploader of the virus deliberately sought out a multiply-prohibited way in which to introduce the virus. This, presumably, would delay initial reports of the virus inside an organisation, with users possibly quadruply afraid of recriminations. You can imagine that the average user, complaining to Technical Support about a new-found problem with his or her PC, would be unlikely to offer a report such as, 'I downloaded a shareware porno game file called SEXY.EXE from the Internet, installed it on my work PC and ran it, whereupon the network fell apart'.

SOMEONE ELSE'S FAULT

Although access to the Internet is increasingly popular, and increasingly important to business, it is still often seen as an unconquerably dangerous vehicle, and incidents such as Kaos4 serve to fuel fear and ignorance about the safety and useability of the Internet. Many corporate computer network and security policies deal with the Internet in simple, total fashion – interconnectivity with the 'Net' is prohibited. In the wake of Kaos4, the Internet was certainly a convenient hook on which to hang blame.

Within TCWS, who had (and have) partial Internet connectivity, one of the outcomes of Kaos4 was to concentrate security attention on the interconnection point. Whilst network firewalls do indeed require expert attention, TCWS's firewall was configured quite restrictively, and was already fairly carefully managed. Their internal networks, on the other hand, were not. If the organisation's internal policymakers had stepped back before the Kaos4 incident, they would doubtless have prevented the first attack with ease. If they had resisted the temptation to zoom in afterwards, they would have prevented the second attack.

BATTLE STATIONS

The first TCWS engagement appears to have unfolded like this:

- The actual infected game, as uploaded to the Internet, was brought into the company. Whether it was downloaded directly from the Internet or transferred to an employee by someone outside is uncertain, although it seems that the latter was the case.
- The game was tried on a handful of PCs on the corporate network, which all became infected.
- Infection spread from one of those PCs to one of the company's NetWare servers, and executable objects which were part of the corporate mail system were infected.
- Other servers on the network became infected when control workstations attached to both the mail server and one or more of the user file servers executed infected mail software.
- Workstations were rapidly hit as infected programs on infected servers were accessed by users.

None of the anti-virus scanners in use at TCWS detected Kaos4, which is unsurprising: presumably, the author went to some trouble to ensure that this would be the case, and relied on the virus travelling so far, so fast, that some victims would be hit before their scanner updates arrived.

Scanner detractors will be quick to point out that this is an unacceptable and ongoing weakness in scanner-based protection, and that integrity checkers would have noticed this viral attack as soon as it started. Thus, TCWS's problem was that they had elected to use the wrong anti-viral tools. Scanner versus integrity-checker arguments have become, in some quarters, the 'Holy War' of the anti-virus community; fortunately, there is little need for us to consider this debate here, because TCWS's problem was rather different, and much broader than this.

Firstly, although several brands of anti-virus software were in use within the company, TCWS had a corporate licence for one specific product. This licence permitted them to install the software on all user workstations, where the built-in integrity checker of the software would have identified the spread of the virus early in the day. Additionally, their vendor was consulted by telephone, and immediately provided a virus database update which allowed TCWS to detect Kaos4.

Some departments of TCWS swung into action quickly, and removed the virus from PCs in their corridors. Others had never got round to installing anti-virus software on their systems in the first place (though the company had bought and paid for corporate protection some time before). They were much slower to react, because they had never carried out a dry run – they left the anti-virus learning process until they were faced with the 'Real Thing', whereupon fear and panic added themselves to the equation and frustrated their attempts to do things correctly and efficiently.

One or two departments took the extreme approach of closing down all their PCs, disconnecting from the network, and laboriously cleaning every PC before going live again. Sadly, the fact that the corporate clean-up was neither co-operatively performed nor centrally run meant that they beat some of the more lax departments to it. Furthermore, they merely removed the virus, and paid little attention to their current network configuration. So, when they brought everything back up again, it was back to square one. Just as the virus had entered and propagated across their server and workstations before, so it did again.

THE ENEMY WITHOUT

From the scope of the attack, you might conjecture that Kaos4 is a 'difficult' virus, with tricks such as fast infection and stealth to help it spread far, fast, and unobserved. Actually, it is a very plain little virus. Kaos4 is a direct action (non-memory-resident) file infector; it has no stealth capabilities, making no effort to disguise itself; and it is entirely unencrypted, so that the text string 'KAOS4' (not 'KODE4' as stated in the Internet posting above) is clearly visible in every infected file. It is, in a word, obvious.

Because Kaos4 is a non-resident, non-stealth virus, it is the kind of file virus that a network administrator, faced with a compulsory viral infection, would be well advised to choose. Indeed, even the 'Golden Rule' of virus hunting – boot from a known, clean, write-protected system diskette – can be ignored in the face of Kaos4. Even without any anti-virus software, a network administrator would need very little to get his network back on its feet again: minimal programming skill; a BASIC compiler (or even a simple programmer's tool such as GREP); and a network operating system which supports login scripting would be enough to do the job.

It is interesting to note that, in my wanderings around TCWS whilst on contract to investigate the nature and extent of their problem, I came across a small workgroup who had decided to take on Kaos4 themselves. Although most computer scientists who have ever been involved in technical support might shudder at the thought of a band of 'Have-A-Go-Henries' within a community of computer users, their approach demonstrates that careful thinking, combined with the use of obvious precautions, can often produce a high-speed one-off solution to an apparently large problem.

They noticed that program files on their workstations were growing in size, and surmised that they had a virus. One of the team set about an immediate backup of the workgroup's PCs; two others began to examine the altered files. Although they knew very little about viral replication, they compared infected objects with fresh originals, and deduced enough about the relationship between the two to guess how to convert infected files into clean ones.

One of the two had done a little Pascal programming, and whipped up a utility to apply the conversion scheme they had deduced. A little testing, both of the hypothesis and their utility, and they were ready to try it out. By this time the backup was complete, so they had little to lose – instead of waiting for their network administrator to visit their machines personally, they were the first on the block to be up, running, and clean. In some companies, you could probably get sacked for that sort of behaviour – but these three 'Wild West' problem solvers got away with it. Their utility worked; they simply neglected to mention it to their administrator, and quietly allowed him to take credit for cleaning their workplace. And they did make a proper backup first, so their experiments were relatively risk-free.

THE ENEMY WITHIN

If someone tosses a lighted match through your letterbox, burning down the house, they would be guilty of arson. They ought to have foreseen that your house might burn down, and should be punished accordingly. At the same time, if you knew such an attack was likely, it would be a wise move to buy a fire extinguisher, and to learn to use it. It would also be prudent to give up the habit of storing uncovered buckets of aviation fuel inside your front door.

In the same way, the ultimate responsibility for the attack on TCWS lies with the author of the virus – Köhntark, as he seems to call himself (there was, at the time of the first appearance of the virus, an attempt to establish some sort of forensic link between the person who uploaded the virus to the Net, and the author, though no connection was ever proved).

In terms of self-protection, however, some of the TCWS internal networks and procedures were veritable buckets of petrol, and this served to help the attack succeed. Most of the mistakes are fairly obvious from the battle chronology listed above. Their details are as follows:

- Untrusted software – even if its origin appears more benign than an unmoderated Internet newsgroup – ought not to have been used directly on any machine on the main network. Untrusted software should be track-tested on an auxiliary system first.

- The central mail system executables, although write-protected on the server, became infected during a SUPERVISOR login from an infected workstation. Because super-user logins grant unlimited power, they ought never to be allowed except from secure, trusted workstations. Additionally, in this case, the super-user login was used for a mundane task out of habit, not out of necessity.
- The virus spread from the central mail server to other servers, thanks to inappropriate access control configurations on those other servers. Programs had been unnecessarily deposited in a world-writeable data directory due to a misunderstanding about the requirements of the software.
- Workstations for which anti-virus software had been ordered, received and paid for were unprotected. Even though the software would not have detected Kaos4 directly, its integrity checking component would have provided rapid indirect notification of infection, if it had only been installed.

These configurational and procedural errors were confirmed rapidly during my investigation, and steps were taken to educate the company's network administrators about what was wrong, and what they could do at once to reduce the risk of reinfection. To what I thought was the extreme credit of the IT managers at TCWS, I was never asked to present them with information that could be used in a witch-hunt. The brief of my work was simple: find out what went wrong this time; help set things up to prevent it happening again.

The wisdom of TCWS in resisting a knee-jerk reaction, such as insisting on finding someone to punish, cannot be understated. In this case, it was recognised that numerous mistakes had been made, and that these mistakes had worked together to leave the corporate network insecure. There had obviously been no deliberate sabotage attempt; instead, the virus attack was seen as a 'total corporate quality' failure. With this in mind, an in-house seminar, open to all staff and paid for out of a central corporate budget, was scheduled and duly held.

Sadly, there was a repeat attack of Kaos4 at TCWS about three months later. Initially, when I was contacted again to look into the circumstances of this attack and to assist in cleanup, I felt a sense of personal failure. It was not as though I had been contracted to help reduce the number of virus attacks by 7%, or some such nebulous score. There had been one attack, and the new target was zero, which left little margin for error.

REPEAT PERFORMANCE

Cleanup the second time around was straightforward, because most of those involved had previous experience with this very virus. Additionally, I observed a number of things that made me feel much less personally concerned about the repeat attack. Consultants are often despised as those who talk about solving problems because they are incapable of actually solving them; by the same token, their role, especially in large corporates, is usually defined to stop short of implementing any solutions they devise. Consultants usually do not need to say 'I told you so', because that is what they were employed to do in the first place – and this is how it was at TCWS.

The repeat infections would have been prevented if the simple changes recommended three months earlier had been carried out. This time, there was no Internet to blame, as the reinfection started completely internally. Amongst the things which had happened or not happened since my previous involvement, were:

- Reinfected networks on which the purchased anti-virus software (now fully Kaos4-aware) had still not been installed three months later, despite the protestations of users.
- Reinfected networks on which infection had again been spread by shared programs on the server that were world-writeable.

- On-going routine use of the SUPERVISOR account from arbitrary user workstations, due to its convenience.

Clearly, the lesson to be learned here is that where changes are required, they must be seen to have been carried out. Network configuration can be monitored with auditing tools; the output of such tools can be cross-checked. Basic informal examination of network security by users, as well as by trusted outsiders, is a simple task. In a multi-departmental company such as TCWS, with departmental networks, administrators can easily assist one another with basic security audits. The idea here is not to watch for malicious internal breaches of security (that is a separate issue), but to prevent easily-avoided lapses which could prove costly.

A YEAR ON

One of the things which TCWS pushed most strongly after the first Kaos4 incident was the education of end users. My own hope, as the person contracted to run the first major seminar, was that users would be keen to attend. Before the TCWS incident, my experience had been of corporate anti-virus seminars being restricted by top management, in order to keep both direct and indirect costs down. TCWS, on the other hand, made every effort to remove this barrier, providing central funds and encouraging all staff to attend. Their belief was that the cost of the seminar in lost working hours would easily be recovered in hours saved handling virus problems in the future.

The users proved relatively uninterested, despite considerable publicity given to the event via corporate e-mail and through network administrators. In the end, less than 5% of potential delegates attended, although preliminary estimates suggested a turnout of over 12% was likely. I should like to be able to say why attendance was so poor in order to help other organisations avoid similar disappointments; sadly, the reasons were never clear.

It was clear, however, from a survey carried out amongst TCWS users one year after the original Kaos4 attack, that users continue to consider in-house anti-virus seminars unimportant. Users were asked:

```
Rate the following in terms of their importance to virus protection
inside the organisation (use the digits 1 to 5, with 5 for 'most
important', down to 1 for 'least important').

( 4 ) Formal corporate anti-virus policy
( 5 ) Anti-virus software
( 3 ) Network administration and configuration
( 2 ) Seminars, information sharing and awareness campaigns
( 1 ) General attention to 'total corporate quality'.
```

Unsurprisingly, anti-virus software was overwhelmingly voted most important. Seminars and awareness campaigns, however, were rated second last, just above 'total corporate quality'. Clearly, TCWS users do not see the anti-virus issue as their management do: whilst users put a formal corporate anti-virus policy in second place, they seem relatively unconcerned about getting themselves into a position to understand how they might build this policy into their own computing regimen. Whatever TCWS users may think, I agree with their managers, and rate anti-virus protection as a total corporate quality issue.

Nevertheless, TCWS users have a healthy understanding of their own importance in the corporate anti-virus battle. They were presented with:

Rate these people or groups in order of their importance to controlling viruses inside the organisation (use the digits 1 to 5, with 5 for 'most important', down to 1 for 'least important').

(1) Top management
(5) Network administrators
(3) Computer maintenance contractors
(2) Everyone else
(4) Me.

Although they chose to place the bulk of the responsibility on someone else (they picked network administrators as most important), they voted 'Me' into close second, which is a good sign. Top management were placed in a very distant last place. Strangely, however, despite the importance associated with network administrators, it was unclear what TCWS users thought these administrators would be doing in dealing with viruses:

Rate the following items or activities in order of the viral risk they pose to the organisation (use the digits 1 to 5, with 5 for 'highest risk', down to 1 for 'lowest risk').

(4) Exchange of disks with outside companies
(2) Use of disks to move information between work and home
(1) Incorrect network administration and configuration
(3) Software taken from bulletin boards or the Internet
(5) Illicit copying of software from other people.

Incorrect network administration and configuration was felt to pose the lowest risk to the organisation. Although the inextricability of the link between virus protection and network security was stressed at the TCWS user anti-virus seminar, it would seem that there were not enough users there to hear the message that was preached that day. Piracy was rated most risky, with exchange of disks with other companies voted into second place; the Internet got off more lightly than I had expected, rated in third place.

WHAT NEXT?

The organisational culture lessons from this case study are clear, and somewhat surprising: even though your users may recognise the significance of their role in keeping the organisation virus-free, they may yet have a certain reluctance to learn. Their virus awareness may improve, but not as much as you might wish it to:

How would you describe your computer virus awareness of a year ago?

[7%] Excellent. Understood the technical and organisational issues
[30%] Good - confident I knew enough to handle one if I got hit
[46%] Fair - heard of them, and had some idea of how they spread
[17%] Poor - heard of them, but they were 'someone else's problem'
[~0%] Zero - never even knew that viruses existed.

And how do you describe that awareness now?

[7%] Excellent. Understand the technical and organisational issues

```
[40%] Good - confident I know enough to handle one if I get hit
[45%] Fair - heard of them, and have some idea of how they spread
[ 8%] Poor - heard of them, but they're 'someone else's problem'
[ 0%] Zero - never even knew that viruses existed until right now.
```

Satisfactorily, 23% of respondents claimed their knowledge had increased over the last year; 3%, surprisingly, said their knowledge had gone down. Nevertheless, even after a year during which computer viruses received a high profile inside the organisation, more than half the respondents effectively rated their own knowledge as insufficient to deal with a virus should they get hit. And hit they were:

```
Did you get hit by the Kaos4 virus in the past year?

[ 5%] Yes, more than once
[21%] Yes, once only
[74%] No.

Have you had a virus *other than Kaos4* in the last year?
[ 2%] Yes - more than one
[13%] Yes - one only
[85%] No.
```

This is a high virus incidence rate, and the administrative lessons here are obvious, and not especially novel: run your networks properly; use your anti-virus software; and make sure that when network reconfiguration is necessary, that it actually gets carried out. At TCWS, the IT administrators managed to make the same mistake twice. Your goal, of course, will be to make no mistakes at all..

REFERENCES

[1] Ducklin, P: 'Anti-Virus Education: Have We Missed the Boat?'; *Proceedings of the 1994 Virus Bulletin Conference*, September 1994.

[2] Whalley, I (ed.): 'Virus Prevalence Table', *Virus Bulletin*, June 1995.

[3] Greenberg, R & Rosenberger R: 'Computer Virus Myths', October 1993.

[4] alt.comp.virus: Usenet newsgroup - various postings, various dates.

[5] comp.virus: Usenet newsgroup - various postings, various dates.

HUMAN DIMENSION OF COMPUTER VIRUSES

Jean Hitchings

ICL Institute of Information Technology, University of Nottingham, Nottingham, NG8 1HL, UK
Tel +44 115 951 3356 · Fax +44 115 951 3353 · E-mail jean.hitchings@nottingham.ac.uk

ABSTRACT

This research considers the human issues when designing an information system that is resistant to computer viruses. Most information security considers technical factors but often ignores human issues. This paper begins by looking at the development of systems analysis and the compares it with information security. This is followed by a summary of current literature which indicates the importance of human factors.

Finally, there is a case study of a large multinational organisation where a computer virus infected the computer systems. The situation is analysed in context to current literature and the developments which are occurring in systems analysis.

KEY WORDS

Computer virus, information security, Virtual Methodology, Soft systems analysis, Human issues and computing.

INTRODUCTION

It is possible to compare information security to general systems analysis as both involve analysing an information system in order to determine requirements followed by a design phase. While methods to implement information security have remained relatively static, the last decade has seen the traditional approach to systems analysis (also known as hard systems thinking) questioned as to its suitability to information systems. A major problem with the traditional method is that it ignores the human factor. Information systems are considered in the same light as machines, assuming that they behave logically and as instructed. People are the main component of any information system and it is generally understood that they are not totally logical.

The human issues are manyfold and include the objectives of personnel (which may conflict with those of the organisation); the cultures of the people involved; and attitudes which can be influenced negatively by low morale or positively by a good *esprit de corps*. It would not be feasible to expect users to cooperate in designing a system which is going to make them redundant or cause them to carry out a considerable number of extra tasks in their day to day work.

The new approach to systems analysis is called the soft systems methodology [1,2]. It intends to include human issues in the analysis and design phases. In addition, the new methodology considers organisational issues, such as policies. If management policy is to praise staff only on visible output then it is quite feasible that a junior will concentrate on jobs which will help him to obtain recognition and ultimately promotion. Other tasks which may be more important, such as security procedures, could be totally ignored. This was certainly a major contributing factor to a large multinational organisation where outsiders were able to hack into the networked system. There were no obvious results to show management if time was spent on learning and implementing network security. The hackers were able to roam freely through the system with super user privilege, because this was the default. There is obviously a need to reconsider our approach to information security in order to avoid such situations in the future.

Another factor that should not be ignored is the environment within which the system and organisation operate. The environment includes for example, customers, competitors, and legislation. Competitors can affect a company in many ways. An organisation may be forced to produce an extra product or service, because competitors have introduced one and it is proving popular. Customers can affect organisations by only buying a certain product causing others to be discontinued.

The human issue can have a powerful effect on organisations and at last its significance has been considered by systems designers. However, this important issue is being ignored by those implementing information security and the traditional approach is still being used. The information security designer should be even more concerned with human behaviour as after all it is people that commit crime not computers.

Most information security breaches are committed by employees who are opportunists, have seen an opening in procedures and have taken advantage of this [3]. Now that information technology has moved into the open office there is even more opportunity to tempt employees. Coupled with this is the fact that managers in general appear to be unaware that the main threat is from within. Such managers are concerned with procedures that prevent outsiders from entering their systems and are much more lax with internal procedures. This was the situation in the case study described below where a virus infected an organisation's computers.

CURRENT LITERATURE

For some time it has been suggested that information security is not just a technology problem, but that it also concerns people. Davis and Price [4] state that security is a people related issue and give a number of reasons. Firstly, the system is designed by people and the original controls are dependent on their ability to understand the problems and the relevant solutions. The integrity of the system is also dependent on the people who build the system as well as the people who undertake the day to day maintenance. Once operational, the system is reliant on people to run it. They must carry out the security related procedures adequately, if the system is to remain secure. Finally, there is always someone whose level of control of the system is high. This person is necessary in most systems. They may be a senior manager with the authority to transfer large sums of money or perhaps a systems administrator who potentially has the ability to access any data or programs in the system and is responsible for the allocation of passwords to authorised users.

Morrie Gasser [5], devotes a section in his book to 'The Problem is People, Not Computers'. In this he explains that computer crime is usually concerned with breakdowns in procedural or personnel controls, rather than exploiting a weakness in internal controls. He concludes that so long as relatively easy non-technical methods exist to commit a crime, technical controls can largely be regarded as superfluous.

Often the case is put forward that computer systems are not particularly useful for detecting or preventing computer related crime because the perpetrators are usually employees that do not violate internal controls. Instead they tend to misuse the information or privileges which they are authorised to access or use. However, Gasser continues that, on reflection, it is often the case that people gain access to more information than they need. This may be because the security restraints have not been implemented or that it has been too costly or inconvenient to include them.

Wong and Watt [3]) devote a large chapter to people, 'People - Asset or Liability'. In this, they state that many cases of sabotage to computer equipment, data and systems are committed by employees. Also, most computer related fraud has been undertaken by trusted staff in organisations, sometimes colluding with outsiders.

A number of cases are described to support their claim, followed by a section on fraud prevention or reduction measures. These include traditional controls, such as, separation of duties, job rotation, and split knowledge or dual controls. Controls on inputs, outputs and amendments are discussed, along with structured walk-throughs of the system design and good documentation. Finally, there is a section on password management.

It can be seen that although the people aspect of information security has been discussed as a problem by several authors, it does not appear to receive the same attention in industry. This means that either information security personnel in industry are not aware of the issue, (which seems unlikely) or they consider the issue to be too difficult or too costly to implement.

The answer may be as indicated by Gasser, who gives the explanation that while it is relatively easy to detect a single bug in a system which can be exploited for individual gain or to the detriment of the organisation, it is much more difficult to totally eradicate bugs from the system.

THE ORGANISATIONAL DIMENSION

At the time of writing, only two publications have been identified which attempt to tackle security issues using a soft systems approach. Richard Baskerville [6] states that discussion of information security is restricted by the narrow influence of technology as the only solution. He feels this has prevented the field from expanding and keeping pace with developments within the area of computing.

Baskerville looks at the design of information systems security in the light of modern system analysis and design. He states that by discarding traditional information security approaches, it is possible to consider security as a variation of normal information system design.

Hitchings [7] develops an information security methodology called the Virtual Methodology (VM), which considers organisational, contextual and human issues as well as offering technical solutions.

It is clear that there is a genuine problem and current literature indicates that information security is a management issue which involves people. Over the last decade computing has developed rapidly, with advances in technology and in methodologies for systems analysis and design. In the area of information security great progress has been made in the technology being used, but there have been no real advances in the management of these techniques and the understanding of the role of human factors.

The methods for implementing security are outdated and a new methodology is required that takes into account the people problem. This methodology should ideally follow the soft systems approach, in keeping with current trends in systems analysis. By considering the human issue, an organisation may be better equipped to tackle the problem of information security and deter the introduction of computer viruses.

There follows a description of a case study where a virus was introduced into the organisation's computer system by an application programmer. It was allowed to happen because of organisational policy and carelessness on behalf of the employee.

THE CASE STUDY

1) Introduction

This case study refers to the accidental virus infection of a personal computer caused by unauthorised use of software. The personal computer was one of many in use in a large confectionery organisation providing the possibility of massive cross infection.

2) The organisation

The organisation is a major international confectionery manufacturer based in the UK with a significant world presence. Their strength in this market has been increased by a merger with another large company that also manufactures confectionery.

At the time of the incident the organisation employed more than 10,000 people in Europe and the UK with a turnover of around £1 billion.

3) Systems description

Information technology plays a major role in company business. Computer based systems cover all the commercial areas of the business, process control manufacturing, environmental controls, research and site security.

4) Technical details

The organisation is a medium to large IBM site and has a DEC based distributed network. Personal computers are used extensively throughout the company. Both physical and logical security access controls are employed.

5) Context description

The departments involved in the incident were the Finance Department of one of the factory sites, remote from head office, their local Information Technology Department and the Central Information Technology Department at head office.

The Finance Department was responsible to the factory Chief Accountant, who in turn was responsible to the Factory General Manager, who was responsible to the company Managing Director.

The personnel involved were the local Information Technology Coordinator, a local Finance Department Section Head and a Central Information Technology Analyst/Programmer. The Analyst/Programmer was temporarily responsible to the local Finance Manager and his role was to advise and develop a local specialist accounting system based on personal computers.

The local Information Technology Coordinator was also the local Computer Security Administrator and was therefore responsible not only to the Chief Accountant but also to the Information Technology Security Manager.

6) Details of the security lapse

The security breach involved the implanting of a computer virus into the personal computer that was being used to develop an accounting application.

The responsibility for personal computer security rests with the local departmental managers and the local Computer Security Administrator. Standards and guidelines for the use of personal computers in terms of physical, data and software security had been distributed to all personal computer users. However, there were no software controls in place to prevent or detect viruses.

7) Discovery of the problem

The Information Technology Security Manager at Head Office received a report from the Information Technology Manager at one of the other factory sites saying that unusual characters kept appearing on one of the personal computer screens. Following discussions, it was discerned that the personal computer had most probably been infected by a 'friendly' virus.

The personal computer was dispatched to head office and the origin of the virus traced as far as possible. It was decided that the source had been at one of the other factory sites.

At this point the incident was reported to Senior Management and it was agreed that the Information Technology Security Manager should investigate further and report his findings.

8) Organisational analysis

The size and geographical distribution of the organisation makes it difficult to check that each of the personal computer users are adhering to the organisation's distributed standards and guidelines.

As previously stated, the responsibility for personal computers rests with local departmental management and local Security Administrators. It is not practical to check on every user to ensure that standards are being followed. The main duty of the management is to ensure that each employee is aware of the need for information security and to provide the information as necessary to observe company standards.

Personal computer software was usually obtained from standard, reputable suppliers, but some arrived unsolicited through mailshots. Sometimes games were brought from home and loaded onto a company personal computer to play during lunch breaks.

The effects of a malicious virus spreading through the organisation's personal computers could have a significant effect on the computer based application systems. The biggest impact would probably be the time it would take to locate the spread of the virus, remove it and then recreate the data afterwards. The cost of such an operation is considered to be significant.

There have been no changes with regard to the roles and duties of personnel in so far as they relate to information security, however, positive steps (e.g. the installation of virus detection software on each PC) have been taken to reduce the likelihood of known personal computer viruses infecting the organisation's personal computers.

9) Reflections on the organisation

A memo from the company Managing Director was issued to every employee stating categorically that the playing and storing of computer games on company computers was now banned and anyone found to be doing this would be dealt with at a senior level.

The organisation has installed virus checkers on all personal computers. It has also established a 'quarantine area' for scanning any unsolicited software that may arrive through the post. Even though this service is well publicised, it relies on the active involvement of the recipient.

The organisation also accepts that there is a minor risk of being infected by a virus from a standard supplier but feels that the suppliers checks are adequate enough for this to be ignored.

10) What happened to the people concerned

An employee who was an Analyst/Programmer on secondment to the finance department was suspected. He was severely reprimanded, removed from the project he was working on and returned to head office.

The Analyst/Programmer left the organisation shortly after this incident to set himself up in business writing software for personal computers.

Members of management and those involved in IT security in the organisation were considered to have behaved correctly and in accordance with organisational procedures.

11) Conclusions from the case study

Viruses are considered by the organisation to be a real threat, not just to large companies but also to standalone applications in small enterprises and therefore should be taken seriously.

It was felt that the creation of standards and guidelines was vital and that it was essential to check for their compliance, even if it was only in a cursory way.

The organisation thought that awareness of the need for information security has to come from the top and each layer must be seen to support the policy.

Organisationally, information technology plays an important role in this company. In addition to business computing, it is used in other areas such as process control manufacturing and site security. Through experience, the company has a well-developed ethos of security towards its computing resources. However, its geographically dispersed nature has led to a fairly complex organisational structure with some duplication and some gaps in managerial responsibility.

In particular, the growth of PC based end-user computing has provided some weaknesses in overall organisational control. This is because the security ethos was based on centralised large scale computer systems. Consequently, there was a lack of sufficient security measures and procedures in this area.

The management of end-user computing resources is by its nature a difficult task. Its main purpose is to harness the creative talents of employees by arming them with powerful tools, but inevitably this can cause control problems from an overall organisational perspective. Managerially, it requires a level of trust combined with publicity and an educational policy that creates awareness of the organisational risks. This is largely because the disparate nature of the computing resources makes it very difficult to regulate each individual user.

Unfamiliarity with the culture of the PC can, as in this case, present problems. The ease of access and widespread use of PCs (many people own one at home) encourages the exchange and swapping of software, (especially games) and experience. This is completely in contrast to the bureaucratic, centralised and heavily controlled culture of centralised computing. The capabilities of many modern PCs easily outstrip mainframe computers of a few years ago and the sense of power accorded to end-users may lead to almost fanatical extremes.

The problems in the case arose out of a common blind spot exhibited by many computing professionals whose experience is based on centralised systems. Despite the existence of seemingly secure systems, they were unable to anticipate one of the major problems of PCs - virus infection.

The perpetrator of the misuse was clearly a PC enthusiast who was almost certainly using company resources for his own interests. His subsequent occupation as a games author would seem to confirm this.

An approach to secure systems such as that offered by the Virtual Methodology [Hitchings,1995] would not only reveal the organisational and managerial issues but it would expose this kind of weakness. It would also, for example, highlight the bureaucratic or political nature of control exercised by centralised computing departments and the clash with the ethos of end-user computing.

CONCLUDING REMARKS

The prevention of computer viruses and information security in general is usually considered a technical problem, however, it has become clear that organisational issues have been a major factor. The size and geographical distribution of an organisation can make information security weaker, especially if the company has become too devolved and there is no centralised checking or controls.

The ethos of organisations must be reconsidered with information security in mind. Too much trust is not desirable, it must be balanced with adequate controls. Allowing staff to run their own software on the company's machines may at first seem harmless. However, on closer inspection, it is obvious that a virus can enter a networked system in this way and spread throughout an organisation's computers.

Managerial issues are also of importance. Management skills should be improved so that a managers understand their staff and are aware of what they are doing.

It is now known that most threats to information security are from insiders. This highlights the fact that trust in employees must be balanced by adequate controls. It also shows that the myth of the lone hacker attempting to disrupt systems should not be the major concern of organisations.

By using a methodology like VM, information security can be dramatically improved because the entire organisation is considered and not just one area. Organisational ethos and policy are re-evaluated with security-in mind, as are managerial issues. It only takes one breach in security for a problem to occur and by looking at the organisation as a whole, as well as local issues, a successful information security policy is more likely.

REFERENCES

[1] Checkland, P., 'Systems thinking, systems practice', Chichester: Wiley, 1981

[2] Checkland, P. and Scholes, J., 'Soft systems methodology in action', Chichester: Wiley, 1990

[3] Wong, K. and Watt, S., 'Managing information security', Oxford: Elsevier Advanced Technology, 1990.

[4] Davis, D.W. and Price W.L., 'Security for computer networks', Chichester: John Wiley and Sons, 1987.

[5] Gasser, M., 'Building a secure computer system', New York: Van Nostrand Reinhold, 1988.

[6] Baskerville, R., 'Designing information system security', Chichester: Wiley, 1988.

[7] Hitchings, J. 'Achieving an integrated design: the way forward for information security', *Proceedings of the Eleventh International Security Conference IFIP SEC '95*, Cape Town, South Africa, 9 - 12 May 1995, Chapman & Hall.

FULLY AUTOMATED RESPONSE FOR IN THE WILD VIRUSES (FAR - ITW)

Mike Lambert

Frontier Corporation, 61 Coventry Avenue, Rochester, NY 14610, USA
Tel +1 716 777 4761 · Fax +1 716 423 9853 · Email mlambert@rochgte.fidonet.org

1 INTRODUCTION

I recently looked at some anti-virus (AV) products as a buyer and found that the state-of-the-art in virus response has moved forward, but only slightly, toward what we need in an enterprise environment. One product will auto-disinfect floppy disk boot sectors on presentation without user intervention. Another product will auto-recover from BS/MBR virus infections but still requires operator intervention. There is also a product that will install on infected systems and clean the system during the installation. There is a product that has been capturing virus specimens for years. Many products will notify someone over a network, but incident accounting seems to be missing altogether.

The reason for the slow progress may be because there is no vision of what kind of product we should be moving toward, or maybe the AV development community is just more resistant to change than other development communities. We can do something about the former; that is, tell the AV development community what we want. This paper is just that, the 'Enterprise Wish-list for AV product developers'. I hope that there will be other papers that will correct my errors or include my omissions when and if they are identified. Such work is good and will fill the need for the lack of visionary direction it seems that we need so desperately.

Generally speaking, we are still working with the philosophy of non-automated response to virus exposure and infection response. Basically it's 'product sees it, someone cleans it'. This is fine for the single user at home, but not much good for an enterprise environment. ***What we need is a Fully Automated Response (FAR) for virus exposures and infections produced by the viruses from which we are most at risk, the In The Wild (ITW) virus.*** I would like to see an automated response with no user intervention and automatic sample gathering and reporting for ITW virus infections and exposures. I think this is extremely important in our environments which are directly exposed to the same in-the-wild viruses on a daily basis.

FAR is a response philosophy for the known risk; it is not a substitute for unknown risk mitigation. FAR handles the 99% you know, not the 1% risk you have yet to experience. Once experienced, the unknown risk becomes the known risk and is included in the known risk handling (FAR).

2 SPECIFICATIONS AND DEFINITIONS

The target audience is the Corporate Security Manager or the Network Administrator responsible for Anti-Virus capabilities. This is not meant to be a feasibility study, a technical justification, or a technical description of implementation. This is intended to present the idea of FAR to the target audience, highlight a few problems which will surface, and describe what FAR might look like from the functional point of view.

This paper is concerned with only a subset of all known DOS viruses. That subset consists of the 'in-the-wild' (ITW) viruses. There is no provision for MAC viruses, or for viruses not found in the wild. The reader is charged to keep this in mind throughout the paper, because what is stated here and applies here applies to 'in-the-wild DOS viruses' only. What I am asserting may not be applicable in the theoretical realm which includes response for all known and future viruses, and it isn't meant to be.

In-the-wild viruses are those viruses which are actually cruising computers in homes and organizations. These are the viruses that are likely to visit your organization (99% or more of the time). There are zoos containing thousands of viruses. These zoos are passed around to professionals, non-professionals, and the curious. Just because a virus is in a zoo, it does not mean that you are likely to see it in your organization. 'Zoo specimens' should not be confused with 'in the wild' viruses. I will refer to the 'in the wild' as ITW.

The current document which attempts to identify ITW viruses and appears to be accepted by all is Joe Wells' *Wildlist.* It is a good starting place, but one should keep in mind that there are viruses in the wild which are not on Joe's Wildlist, and a few problems with the list itself. For instance, because of naming variations, the same virus appears more than once. Since there is not a repository for actual ITW samples, some of the listed viruses are too vague to point to a specific strain. I think using this list is the most acceptable place to start, and that the list will overcome its current problems as it is used more.

I use the term 'virus exposure' as the general definition of a clean system in which a virus is present. Examples of this are: 1) a clean system which has an infected floppy disk in the drive; and, 2) a clean system which has an object (file infector) with a virus in it that, if executed, can infect the system.

I use the term 'virus infection' as the general definition of a system which has a memory-resident virus active and capable of reproducing (non-resident viruses not included).

I use the terms 'enterprise', 'organization', and 'site' to indicate a networked system with multiple DOS computers connected to it. This could be a small company, a school or university, or a multi-national corporation.

FAR is Fully Automated Response. This is an automated identification, disinfection, and reporting response to those viruses for which it is most appropriate. No user intervention is required and no user notification is delivered. Most BS/MBR, multipartite, and file viruses can be dealt with by restoring infected objects, even with the virus present. Some viruses cannot or should not be included in an automated response; these viruses are termed *non-FAR viruses*. Non-FAR viruses include those which destroy the executable or which require the virus to be present to access data.

A *FAR-ITW virus* is an 'in the wild' virus that can be disinfected without a special disinfection procedure (i.e.. can be totally contained in a software solution). All ITW boot sector (BS), master boot record (MBR), companion, directory, appending, and prepending viruses that do not produce 'virus resident dependent' problems should be included in a fully automated restoration capability. Exceptional ITW viruses requiring special disinfection procedures need not be included in the FAR requirement and are noted as non-FAR-ITW viruses. EXEbug would be a FAR-ITW virus, One-half may be a non_FAR-ITW virus (depends on the expertise of the AV developer). Some hardware implementations may dictate some virus infections as non-FAR.

A ‘workplace interruption’ is any unnecessary interruption of a user’s productivity. This includes manually cleaning floppy disks, scanning systems, cleaning systems, etc. A workplace interruption is not limited to a single worker; it often extends to other workers nearby. Workplace interruptions can be short or extended. The organization loses productivity with each incident which interrupts the worker. This productivity loss is often much more expensive than the actual cost of the technical response to the incident. In larger organizations, with employees located across large geographical areas, virus incident support often requires the user, increasing the magnitude of the interruption. An interruption need only occur when the ITW virus infection is non-FAR.

Non-ITW viruses can be included in the FAR but if it is not ITW there is little reason to include them. Doing this work ahead of time will certainly be necessary if the virus ever attains the ITW status. If a non-ITW virus is subsequently found in the wild, it can be included at that time.

No specific network is required for this model; communication requirements are stated generically. The specific implementation will vary from network to network, feature to feature.

‘NLMs’ and FAR have almost nothing to do with one another. File and multi-partite infections may reside on servers, but it is workstations that ‘get infected’ and ‘spread infections’. An NLM does not FAR; don’t confuse an NLM scanning executables delivered from a server with a Fully Automated Response to a virus exposure or infection. The closest an NLM comes to FAR is to ‘move’ or copy a suspected file to a protected directory. This is not FAR. You may need an NLM if you don’t have any decent network security (i.e. someone using the network can infect an object residing on the network), but FAR is a different philosophy.

3 IN THE WILD VIRUSES

So just how many of the thousands of viruses are we talking about? It is not near as many as you think! Joe Wells’ list of July 1, 1994 lists a total of 152. January 1, 1995 shows a total of 197. The current list of June 1, 1995 gives us a total of 235 worldwide! I know that there are more ITW viruses than those on Joe’s list. It is the requirement of list participation which causes this problem. Even if we add 30% to Joe’s list we still only reach just over 300 viruses!

Of the 6,000 or more viruses in existence, it is most probably a mere 300 that we are talking about for FAR! This is just 5% of the world’s virus population.

Let’s say that 35% of the ITW are BS/MBR viruses. A conservative figure is that 65% of the virus incidents in the an organization are ITW BS/MBR viruses. This means that more than 65% of the problem is caused by a mere one and one quarter percent (1.25%) of the viruses! In many organizations, BS/MBR infections account for 90+% of the number of virus incidents.

I think that when we look at the viruses which we see every day; they are virtually all ITW viruses.

4 THE FULLY AUTOMATED RESPONSE (FAR) OVERVIEW

Fully Automated Response is:

A. Detect the ITW virus

B. Remove the ITW virus automatically and without any user notification or intervention

C. Report the incident to technical support

The difference between current solutions and FAR is that software does all the work automatically and quietly. We make all the decisions up front.

Decision 1 - always remove the ITW virus.

Decision 2 - always report incidents and include a sample of the virus.

Decision 3 - always compile statistics.

The event, as an exposure or an infection, must be closed by the conclusion of the response. After-action decisions, such as notification, are handled outside of the response. *It is this 'total response to the event' that we currently lack.* We augment our current responses with customer service manpower, technical support manpower, user manpower, and training manpower. All of this manpower is a waste of our organization's resources. We've just got to do things smarter!

FAR should be considered a 'medium security' solution. (Those needing a 'high security' solution must add appropriate technology.)

Note that FAR lacks the 'tell the user and make a big deal of it' philosophy. I have found in asking users that the last thing they want to deal with is a virus; 'I just want to do my work' is the most common phrase I hear when talking to enterprise users. I agree with the user; let them work. Let virus administrators notify users when necessary.

For those concerned with 'user interfaces', what better user interface could there be than FAR? *Absolutely no user input is required.* In non-FAR situations, user response is via a message directing the user to call the Help Desk (or whatever is desired). Administrator interface (to the FAR product administration) is a different issue, and keep in mind that system administrators generally are more computer literate than most end users.

The reasons we need FAR are:

- *It costs too much* to 'open a ticket and dispatch a tech' for individual virus incidents
- The resulting (and expensive) *workplace interruption is not necessary and too costly*
- We can *cut down the necessary virus training* for technicians and employees
- We *need the automated reporting,* so that we can justify re-licensing the product next year.

If your virus problem consists solely of infections by FAR-ITW viruses, it is easy to calculate the cost savings. Just add all the costs of the Help Desk calls, plus the PC technician's time, plus the lost worker productivity. This is the amount you would save with FAR for ITW viruses.

FAR need only handle our biggest risk, the ITW virus. There is no need to include every obscure research virus in the FAR concept. New ITW will viruses appear, and need to be included in the FAR. Most products have demonstrated that they can supply scanners and cleaners for new ITW viruses very quickly (sometimes in mere hours), so including them in FAR products is trivial.

The idea behind FAR is to use an automated response to a FAR-ITW virus infection or exposure whenever possible. *This includes product installation.* There are no technical reasons for why a FAR-ITW virus cannot be dealt with when installing a FAR product on an infected system. If generic solutions such as a generic MBR restoration instead of a real MBR restoration are necessary, these are acceptable.

'False Positives' are non-existent in the FAR model. Since we are talking about working with known viruses, both memory and object infections can be positively identified.

FAR is not a replacement for all current AV products. FAR is the distillation of the best technologies into the exact product that will be of the most value in fighting viruses on the front lines. FAR's technology comes from the multitude of other products, all FAR necessary technology is already in existence in different products. Research and Development of current products must continue. There is no reason for

FAR to be the only technology an organization utilizes. All organizations must select the technology which best fulfills their security philosophy.

5 PROBLEMS WE FACE TO FAR-ITW

We face four basic problems to implement FAR-ITW:

- No ITW base from which to start from
- Current 'zoo' certifications
- Industry resistance
- Ourselves.

NO ITW BASE

Unfortunately, while Joe's Wildlist is a starting place, it is not a base to define the ITW. There is no actual 'ITW zoo' from which to specify 'these are the ITW viruses'. The only public ITW attempt is currently made by *Virus Bulletin* and it does not resemble Joe's list to any great degree. I have 90% of the ITW viruses on Joe's list and it is easy to argue that they are not the actual ITW viruses which are reported on the list. This is due to the variety of names and inexact identification by some products. Still, it is from this list that we can at least start an interim solution.

We need two solutions to this problem.

The *first* is an interim solution which will comprise a consensus of what AV developers and professionals will concede that makes a reasonable ITW zoo. This work is in progress by Richard Ford at the *NCSA*. Richard is working with the Anti Virus Product Developer group to define an initial ITW specification. The agreed-upon samples will form the first ITW sample base.

The *second*, long-term solution is to 'start from scratch'. We must assemble a 'new' ITW list directly supported by an ITW sample base. These samples would be from actual ITW infections. This would provide the indisputable base from which to launch a definite FAR-ITW implementation. Information compiled must include generic site information and specific virus information *for each infection and exposure reported*. This will give us more information than ever, virus location and prevalence. All incidents need to be reported to get an accurate picture of the true ITW virus.

We, as corporations, organizations, sites and private users, must assist in the assembly of this new base by reporting ITW infections and exposures, complete with samples. The new ITW sample base will identify a real sample of the actual ITW threat to computer systems that all can agree on. This must be a world-wide effort. I am confident that the AV developer, professional, independent, and the user communities will support such an effort.

To facilitate such a massive effort requires central clearing houses from which to assemble reports and samples to define the actual ITW virus. Existing and new reporting lines will be able to support and represent all of us. Joe Wells has a reporting structure in place. Joe will collect samples. *Virus Bulletin* has been reporting ITW infections and can collect samples. The *NCSA* has a large number of members and being a security organization seems the most likely place to house the ITW sample base. All ITW samples must be made available to all AV developers (a situation that does not now exist). Other professionals and independents in organizations can help by forwarding reports and samples to another central location. This could be the first joint venture for all to unite to solve our biggest single problem.

The actual details of creating a working solution from which we may all come together is in the making. It looks like there will be three or four different places to report infections and exposures and send samples. More details will be forthcoming.

ZOO CERTIFICATIONS

While there have been no ITW sample bases to test products with, there are a lot of 'virus zoos' out there. Using these zoos to 'test scanners' has been the definition of AV product testing. 'Zoo scanning' requires a known control base and proper interpretation of results, two things in short supply. What has taken the place of representing our actual threat has been this 'zoo testing'. The scanner is pitted against one or many private collections of a myriad of viruses, Trojans, joke programs, simulator samples, and just plain junk files. We are 'given results' and told that this certifies some product as good for use in our ITW environments. Few things are further than the truth.

Zoo certifications are really a sort of 'what if the research virus made it to the wild?'. Occasionally this happens, but frequently the new ITW virus is new, so 'zoo scanning' doesn't help. Proper interpretation of this sort of 'scanner testing' may be valuable, but just isn't directly transferable to solutions to our ITW problems. Worse still is that these certifications give one a false sense of security because they are very inadequate tests. Just because a scanner can identify something in a zoo scanning test does not mean that it will find the virus in an infected environment (this is a common occurrence).

We *must* de-emphasize 'zoo certifications' and emphasize 'ITW certifications' using a known ITW sample base. Casual magazine-type of 'AV testing' without professional guidance must be avoided.

INDUSTRY RESISTANCE

The AV development industry has just come out of a war with the virus creation community. The virus creation community provided, the AV development community included, regardless. This has little to do with our ITW problems except that some AV developers got so caught up in the war, they forgot the civilians. The fact is, until recently when ITW virus testing became something which could not be ignored, it was ignored by most. There has never been a concerted effort by the AV community to deal with ITW viruses. AV developers must engage 'full disclosure' of ITW viruses. Current exchange restrictions should only apply to research viruses. Current systems and philosophies will take time to change.

The AV industry is sometimes inflexible in what we need, but inclined to what they think we need. Consumers bear a fair amount of the blame for picking products which identified ever-increasing numbers of viruses (zoo certifications) and insisting on a scanner solution. We have refused to listen to sage advice for the different level of security solutions, insisting that there should be just one, the scanner.

The industry is geared to our marketing weaknesses and will resist the direction in which we really need to go. We must change our requirements to reflect our real needs if we wish to move the AV industry to provide the solutions we need.

OURSELVES

We as a consumer community are going to have to look at the information that we see and make some intelligent, objective analysis. We have to learn the difference between an article and an advertisement. We are considered by the marketing community to be drones waiting to be told what to do. Don't blame anyone else; it is our fault. We are much more educated in other areas of purchase, and naïve in AV product selection.

We need to get reliable information from security organizations rather than advertisements. We need to support activities which are directed at ITW viruses and the reliable, appropriate handling of them. If your security organization does not have sound, measurable efforts in these areas, we need to demand them.

6 THE ANATOMY OF FAR

FAR is comprised of six tasks:

1. Identify the ITW virus
2. Collect a sample
3. Clean the ITW virus
4. Make a report
5. Send report to virus administrator
6. Keep all incident statistics for the virus administrator

and must operate equally well in:

1. The clean environment
2. The infected environment

Under clean conditions, FAR requires that the virus be cleaned without user notification or intervention. Notification of the exposure should be a virus sample and report to the virus administrator. Leave the user notification to the virus administrator. Installation should be automatic, no user intervention required, and should save the MBR, BS, system files, and command processor (depending on product design philosophy). These objects can be used later, in infection recovery instances.

Under FAR-ITW infected conditions, FAR requires that the virus be handled exactly as under clean conditions (i.e. identify, remove, report). Installation should recover the system and save the same objects as under clean installation.

Under non-FAR-ITW infected conditions, FAR requires that the user be notified to contact his local technical support personnel. Under these conditions, the situation is hazardous enough to warrant the workplace interruption. All of the sample gathering and reporting should be done, but in the non-FAR-ITW infected condition, the virus is not automatically cleaned.

All of the techniques necessary to accomplish the related tasks necessary for FAR have been in use in one product or another at some time. I note that some products really excel in some of these areas. What we have never seen is those techniques combined to accomplish FAR. It's not that FAR is impossible; it just hasn't been a priority worth pursuing...yet. If the 'best identification', the best 'working in an infected environment', and the 'best restoration' were all to join together with the 'best accounting', we could have a superior FAR product!

7 FAR IN OPERATION

To be complete, I must describe FAR response enough to facilitate its development. Some managers may not be concerned with some of the specifics. Skip this section if you wish.

A THE INFECTED FLOPPY VIRUS EXPOSURE

This is the easiest response to implement. The object in question is not executed and needs to be replaced with a known object.

FAR says that, when a user inserts an infected floppy disk in the system, the following happens:

- Identify the ITW virus

- Save a sample of the BS
- Clean the floppy if not write-protected
- Create an infection report
- Send the report and sample to the virus administrator when possible
- Integrate the infection report into the virus administrator's summary.

Several things happen and don't happen in this instance:

- The user continues working with a clean floppy which will not later infect a computer
- The administrator knows what the user is exposed to before the organization gets infected by it
- Virus incident statistics are automatically compiled

I do not consider the failure to restore a BS on a write protected disk as a reason to interrupt the user. This disinfection failure should be forwarded to the virus administrator and the appropriate decisions made by the administrator.

What's left to do? The virus administrator needs to determine how and when the user is notified of the problem which could have disrupted their work. The administrator also needs to determine the fate of the collected sample.

B THE BS/MBR VIRUS-INFECTED SYSTEM

By definition, this should not happen on a FAR-protected system. Suppose, for this example, that someone booted an infected floppy while the regular user was on vacation.

The FAR response is:

1. Identify the ITW virus
2. Save a sample of the MBR or BS
3. Clean the MBR or BS
4. Reboot the clean system
5. Create an infection report
6. Send the report and sample to the virus administrator when possible
7. Integrate the infection report into the virus administrator's summary

Again, the user continues to work without an interruption, the administrator knows what is happening, and statistics are gathered.

C THE FILE VIRUS EXPOSURE

This looks much like the response to the exposure to the infected floppy BS, but additionally must deal with an object that must be executed. There will be cases where the 'clean the virus' requirement may not be able to be fulfilled. If this situation exists, all other FAR requirements should be fulfilled.

- Identify the ITW virus on copy, execution, etc
- Save a sample of the file
- Clean the object if not write-protected
- Perform the originally requested action (copy, execute, etc) if possible

- Create an infection report
- Send the report and sample to the virus administrator when possible
- Integrate the infection report into the virus administrator's summary

Several things happen and don't happen in this instance:

- The user continues working with a clean executable that does not later infect the system
- The administrator knows what the user is exposed to before the organization gets infected by it
- Virus incident statistics are automatically compiled.

I do not consider the failure to restore a write-protected file as a reason to interrupt the user. This disinfection failure should be forwarded to the virus administrator, and the appropriate decisions made by the administrator.

Same cleanup as the BS exposure: The virus administrator needs to determine how and when the user is notified of the problem that could have disrupted their work. The administrator also needs to determine the fate of the collected sample.

D THE FILE VIRUS INFECTION

File infections come in a huge variety of infection types and techniques. In some cases, the virus is non-resident, making it easy to deal with. In others it may be prudent to use the resident virus. Some viruses are so virulent that they must be removed from memory entirely. The exact technique or series of steps to disinfect the system is dictated by the virus type and capability. However, the goal is the same: remove the virus without user intervention, get a sample, and make a report.

In this case, we imagine an executable which is infected without FAR protection. The subsequent starting of this system thus makes the virus resident. Let's say someone booted their own floppy, infected the system, and left. The virus infection will be active when the system is subsequently booted by the regular user.

The general FAR response is:

- Identify the ITW virus infection
- Establish a clean environment or change configuration, if desired or necessary
- Secure a sample of the virus
- Restore the infected objects or delete companions
- Create the infection report
- Restore the system to an operational configuration if necessary
- Start or restart the system
- Send the report and sample to the virus administrator when possible
- Integrate the infection report into the virus administrator's summary.

The order of some items can be changed depending on the implementation philosophy. The point is to get a sample, remove the virus, and notify the appropriate party when possible.

While the user may get a little show as the system is restoring itself, the user goes to work with the minimum interruption once the restoration is completed, the administrator knows what is happening, and statistics are gathered.

E THE MULTIPARTITE VIRUS EXPOSURE AND INFECTION

The FAR response for the multipartite is much like the BS and file exposure and infection responses, with the added requirement of the additional object restoration.

Again, the user continues to work without an interruption, the administrator knows what is happening, and statistics are gathered.

8 WHEN NOT TO FAR

Not doing FAR does not mean not doing *all* of FAR tasks. Whatever steps can be done, should be done. In many cases, this requires notifying the user and possibly creating an interruption. If nothing else, there should be a sample gathered and report created. The report can be sent if or when practical. If there must be a workplace interruption, the user can be directed to call the Help Desk with a message.

There are times when you don't want to FAR. This would be when removing the virus would deny access to data (in whatever form this may take). In this case it may be necessary to back up data while the virus is present and then remove the virus. Years ago, this concept was illustrated by the Volga virus. It's not that it is impossible to FAR with Volga or One_half, there just may not be enough use of the particular FAR technique to warrant the expense of developing it.

Another instance is when your FAR product meets an anti-AV product that is FAR hostile. Theoretically this should not happen, as the FAR response should be to remove the virus without 'setting off the bomb'.

Other non-FAR situations are those where the object is write-protected. This can be a file on the server, a write-protected floppy, or write protection on the workstation. These situations are non-FAR as cleaning the object is concerned, but should still be as FAR handled as possible.

There are other non-FAR situations that can be described. AV developers may decide to make different viruses FAR and non-FAR. *The definition of a non-FAR virus is 'that virus for which no AV developer can provide a software only solution'.* Not all AV developers are created equally. I wouldn't be surprised to see some 'partial-FAR solutions' for those which lack the skills for a complete solution. I'm sure we will see some viruses considered non-FAR because the virus is not sufficiently in the wild to warrant the work necessary for the FAR solution. There are many shades of gray which the tailor may use.

9 FAR BENEFITS

Of course the benefits are obvious. The enterprise has software do all the work, and the security and network administrators get all the credit. The software even justifies itself for you. The benefit for AV developers is new product potential, significantly less user technical support required, and enterprise users have the justification to relicense next year.

10 FAR EVALUATION

I hope that FAR product evaluation does not parallel many of the current testing and evaluations. If the evaluation does not make a good case for the utility of the product, and properly evaluate that product, we should disregard the test. FAR testing should primarily be concerned with the ability of the product to do the job (and the job is viruses). Tests which are not primarily concerned with the ability of the product to do the job for which it was designed should be ignored.

We, the community which needs and will use FAR, should require that all primary testing is geared to the security aspect of the product, dealing with ITW viruses. Other tests for ease of use for administrators should be clearly labeled as secondary tests, not directly measuring the product for its primary job of dealing with ITW viruses. All Objective conclusions pertain to the primary job of the product; Subjective conclusions pertain to the secondary job of the product.

11 CONCLUSION

FAR is what users want.

FAR saves organizational resources (money) responding to virus incidents.

FAR makes almost all viruses in the enterprise environment a non-event.

FAR takes no technology other than what is already in use to implement.

FAR will be a reality when we demand it.

The question is not 'do we need it?'; the question is 'when will it be provided?'. I'll bet the first AV developer which produces a working version will find that it is 'the better mousetrap'. Just open the door.

CREDITS

PC Viruses in the Wild	(Wildlist) is a collation by Joe Wells in co-operation with many AV product developers and AV professionals.

The ideas and opinions expressed are wholly my own and are not necessarily those of my employer or associates. I wish to acknowledge and thank all those people who have contributed to the AV community and its advancements; their efforts and views become a part of all of us and are many times difficult to separate from our opinions.

Special thanks to my friends and associates who have listened, assisted, and criticized me during the writing of this paper. The remaining imperfections here are mine alone.

THE PC BOOT SEQUENCE, ITS RISKS AND OPPORTUNITIES

(THE USE AND IMPORTANCE OF THE BOOT SEQUENCE IN REDUCING BOOT VIRUS EPIDEMICS)

Jonathan D. Lettvin

OverByte Corporation, 194 Waltham Street, Lexington, MA 02173-4914, USA
Tel +1 617 860 9119 · Email jonathan_lettvin.lotus@crd.lotus.com

ABSTRACT

A single bad user habit accounts for the epidemic spread of most boot sector viruses (leaving diskettes in the boot drive). Users need education to 'unlearn' this habit. The best time to educate users is when they boot.

No general anti-virus product provides boot-time educational text, or anti-virus code. Some products provide a virus detecting TSR for DOS. This would usually be ideal because the TSR identifies a boot sector virus before it is activated by finding it on the floppy before the user boots. However, lack of compliance, bad habits, and unavoidable events make booting from floppy a serious continued source of virus spread.

We have discovered that specially prepared diskettes, formatted for general use and for delivery of commercial products, provide a new and important layer of virus epidemic prevention.

WHO AM I?

I am an employee of *Lotus Development Corporation* and, at the same time, I am president of *OverByte Corporation* having a special relationship with *Lotus*. *Lotus* uses our anti-virus products and services. I speak to you today as the President of *OverByte*.

WHO CARES ABOUT EPIDEMIC BOOT SECTOR VIRUSES?

I think everyone in this room has spent time dealing with epidemic boot sector viruses. The 'virus prevalence charts' document that epidemic boot sector viruses dominate the virus industry. If you are an anti-virus developer or reporter, you are confronted daily by epidemic boot sector viruses. The majority of corporate anti-virus dollars are spent on removing epidemic boot sector viruses.

I've spent over six years at *Lotus* studying viruses and the corporate response to the problem. In the last five years, I saw exactly one 'file infector' attack by Natas. All other virus attacks were attacks by the top twenty or so epidemic boot sector viruses. At *OverByte* we know customers need to detect thousands of other viruses just for diligence, but we rarely hear of even those thousands of viruses that are in the wild.

WHO CARES ABOUT THE BOOT SEQUENCE?

I believe the moment of virus attack is the best time for virus defense. At no other time may any program assume total control of a PC without risk. The boot sequence is unique in its opportunities for virus combat. All epidemic boot sector viruses rely on special knowledge of the boot sequence. Boot sector viruses modify BIOS resources before any operating systems are loaded. Once an operating system is loaded, restoring BIOS resources perfectly is impossible.

When a virus is detected, many anti-virus products force a *clean* reboot of the PC as part of virus removal. The methods used by *OverByte* to restore the boot sequence often pre-empts the need for rebooting.

I feel that this is important. I believe that you in the audience will agree with me when you experience the difference DisQuick diskettes make both in initial virus detection time within a group, and the elapsed time between an attack of an epidemic boot sector virus and when you can get back to work.

WHAT IS THE BOOT SEQUENCE?

This talk is focused on what I have identified as the boot sequence. I consider the boot sequence as the transition from Power-Up, through POST (Power On Self Test), MBR, and BS to any OS (Operating System). Virus programmers have exploited this transition more than any other aspect of the PC. I will certainly not be able to cover the subject exhaustively during this talk. Mostly, I will describe a few of the methods already exploited by virus programmers and mention quickly other areas needing protection.

I know of no good book describing the exact requirements of the boot sequence. I constructed the 'GENERIC PC STANDARD', which we will describe shortly, and how it is used by the boot sequence from first-hand experience and hints from the many popular books which cover PC Hardware, BIOS, DOS, and collateral subjects. I have put a short bibliography of our most frequently referenced books at the end of this paper.

WHAT IS THE GENERIC PC STANDARD?

What defines a generic PC is a fairly uniform standard of operation between hardware and firmware. This standard is so well documented that very little stands unrevealed, although almost none of it is official. The standard is so firmly entrenched that a specific PC has little market unless it scores very close to 100% against compatibility tests.

Compatibility tests include recognition of hard-numbered BIOS entry points for vectors, exact contents of certain RAM locations during the POST (Power On Self Test) and boot process, exact contents of certain disk locations, and many, many more. The actual standard even allows for relaxation of enforcement on documented standards. This means that methods considered to be permanent standards often change.

One example of an item incorporated in the GENERIC PC STANDARD is that the absolute real-mode ROM-BIOS location F000:EC59 will probably remain permanently as the entry point for floppy diskette BIOS services. Many useful programs use this hard-coded entry-point.

To clarify a little more, consider what a generic PC is not. It is not the applications we run every day. It is not the device drivers loaded by our preferred operating systems. It is not DOS, OS/2, or Windows. However, almost all of these rely on the GENERIC PC STANDARD, and will fail on PCs which deviate from this norm. It is not even the specific PC on your desk which has a specific configuration and physical options set, although, in all likelihood your PC follows the generic PC standard completely.

WHAT ARE THE RISKS IN DEVIATION FROM THE STANDARD?

Deviation from the standard can cause failure of operating systems, device drivers, and application programs. Developers of add-in cards must usually recognize and follow these standards completely to be successful.

On a system where a standard breaking card is installed, viruses sometimes fail, with dramatic results. Even on a system where the standard has been upgraded, viruses which depended on an earlier standard may cause system failure. For instance, some common viruses do not recognize the 1.44 MB drives which are now the standard boot drive. These viruses make the assumption that the drive is 720K, and use unintended sectors for propagation.

Most viruses are not successful in following the new standard for operating systems which engage 'Protect Mode' on the newer *Intel* chip family members. These operating systems will sometimes warn the user with obscure references like 'Cannot load 32 bit driver', when a virus has chained a vector which needs to be in ROM-BIOS to follow the new standard.

I suspect that part of the reluctance in adopting new operating systems has been due to interference by viruses, making a given PC appear to ignore the new standard, and thus misbehave. This misbehaviour is often seen as a dramatic decrease from the intended performance of the new operating system. This kind of anecdotal report spreads rapidly by word of mouth, making potential new users reluctant to purchase otherwise fine software.

WHY ARE ANY VIRUSES SUCCESSFUL AT ALL?

Not surprisingly, most successful epidemic viruses attack PCs as hardware. Most virus programmers use knowledge of the GENERIC PC STANDARD, as well as knowledge of specific file systems and operating systems, to propagate. Since the average user is interested only in application programs useful to their daily activities, the very slight changes viruses make in resources and operations on which the operating system relies are usually either invisible or merely nettlesome until the warhead triggers.

WHY IS THE FORM VIRUS THE MOST SUCCESSFUL?

Probably the most frequently found virus in the wild is the 'Form' virus. We believe the reason for its success is its very careful attendance to the PC and FAT file system standards. I note the remarkable care taken by its author to allow Form to adapt. Form appears to lack a formal warhead, but it has certain technical flaws which amount to an accidental warhead. I could be convinced that the flaws were intentional. The consistency of code style breaks for this flaw.

WHAT CHARACTERIZES BOOT-TIME RAM AND VECTORS?

The actual condition of the vectors and BIOS data when the boot sector has been read in is required to have a certain character. When this character is not correct, we believe a number of methods can be used to correct it. The only time this character may be corrected safely is during the boot.

For example, a documented BIOS vector will point into one of several known areas. If the vector is in a segment F000 or above, we consider it safe. If the vector is in segment C000 or above, but below D000, we consider it safe. If the vector is below segment A000, we consider it distinctly unsafe. However, the standard leaves segments D000 through EFFF as areas of conjecture. We have methods for dealing with these as well, but the standard is unclear for that range.

WHAT ARE THE STANDARD STEPS IN THE BOOT SEQUENCE?

The PC boot sequence is a long process to explain in words. I will be discussing the four numbered items of the boot sequence in greater detail after the description of the entire boot sequence in digested form. I break the sequence down into four major sections:

NORMAL PREAMBLE TO FLOPPY OR FIXED DISK BOOT

Starting at power-on, the Intel chip loads REAL CS:IP with FFFF:0000.
This location is in ROM-BIOS (Writable FLASH EPROMS in newer PCs).
The instructions at FFFF:0000 start the Power On Self Test (POST).

1. POST installs vectors pointing into ROM or shadow RAM.
   POST performs all its duties and scans for additional ROMS.
   Each ROM may initialize more vectors, returning control to POST.
2. CMOS memory is examined to determine the device to be used for BOOT.POST finishes and checks for presence of a floppy in the BOOT drive.
   If a CMOS confirmed floppy is present, BIOS performs the FLOPPY BOOT.
   If not, BIOS performs the FIXED DISK MBR BOOT.

NORMAL FLOPPY BOOT

Read drive 0, sector 1, head 0, track 0 into location 0000:7C00.
Then continue the process with the GENERIC OPERATING SYSTEM BOOT .

NORMAL FIXED DISK BOOT

Read drive 80, sector 1, head 0, track 0 into location 0000:7C00.
Set CS:IP to 0000:7C00.

3. The code executed is usually a standard MBR.
   The MBR contains two critical data areas, one undocumented.
   DRIVE ID used by newer Microsoft Operating Systems
       Offset 1B8H through 1BEH is undocumented.
   PARTITION TABLE (PT)
       Offset 1BEH through 1FEH is well documented.
   The MBR code starts by copying itself to location 0000:0060H.
   The code sets CS:IP to 0000:0060H transferring control to the copy.
   The MBR code reports any errors or absence of partitions in the PT.
   If there are no errors, the MBR code loads the active partition BS.
   This BS is loaded at 0000:7C00.
   Continue the process with the GENERIC OPERATING SYSTEM BOOT.

GENERIC OPERATING SYSTEM BOOT (USING IBM DOS AS AN EXAMPLE)

Set CS:IP to 0000:7C00.

4. The code executed is usually a DOS BOOTOR SECTOR sector.

   This code may change diskette-tuning parameters in BIOS.

   This code analyses the BPB (BIOS Parameter Block) for disk structure.

   The BPB contains the size of the system areas and the sector count.

   Using the BPB, the directory is found and loaded.

   Compare the first items in the directory to IBMBIO.SYS and IBMDOS.SYS.

   If they exist, set the cylinder/head/sector to logical cluster 2.

   Calculate the cluster count from the size of IBMBIO.SYS.

   Load those sequential clusters starting at cluster 2 into RAM.

   If absent, notify the user of a 'Non-System Disk' and request reboot sector sector sector sector. If present, set CS:IP to the beginning of the loaded IBMBIO.SYS.

For brevity, I have marked only four of the many places of virus risk in the sequence. These four will be discussed next.

WHAT ARE SOME PRIMARY RESOURCES EXPLOITED BY EPIDEMIC BOOT VIRUSES?

1. VECTOR CHAINING

Most boot sector viruses simply chain INT 13H (BIOS disk services). Some boot sector viruses chain a vector like the timer and wait for DOS to be loaded. All current epidemic boot sector viruses occupy RAM and not ROM. This may change, due to certain advances in PC BIOS distribution.

2. CMOS MODIFICATION

Some viruses will modify CMOS RAM during infection, and force the PC to ignore the diskette drives during subsequent boot. This kind of virus will then always boot from fixed disk. Once the virus boots, it may hide its having booted from fixed disk by detecting, loading, and executing the floppy boot.

3. MASTER BOOT SECTOR CHAINING

Many methods are used by viruses to gain control during fixed disk boot. One virus has completely rewritten the code for loading the active partition. Others use methods similar to diskette boot sector chaining. Some viruses change methods on fixed disk, when the same method would suffice.

4. OPERATING SYSTEM BOOT SECTOR CHAINING

We see two common methods for storing boot sector viruses on a floppy. The first method moves the original boot sector to a fixed location. The second method calculates a place to store the original boot sector. When using the fixed location, the virus programmer counts on rare overwrites. When using calculation, the virus programmer counts on rare disk maintenance. Both methods have been successful. An attempt to subsume the entire legitimate boot code will probably fail.

WHAT ARE SOME BOOT-TIME OPPORTUNITIES AGAINST EPIDEMIC BOOT SECTOR VIRUSES?

1. VECTOR CHAINING

Certain portions of the first 1 MB are known to be ROM only. Other portions may be either RAM or ROM, and still others are RAM only. We detect vectors which are potentially pointing into virus-owned memory.

2. CMOS MODIFICATION

Where a specific virus has modified the CMOS memory, the original values can usually be restored.

3. MASTER BOOT SECTOR CHAINING

We cannot claim to protect Master Boot Sector code from rewriting viruses. However, in combination with our diskette methods, we will still detect them. For chaining viruses, our code will usually alert the user to a virus attack. We will install a new and better Master Boot Sector code portion than the original when we remove one of these viruses.

4. OPERATING SYSTEM BOOT SECTOR CHAINING

BIOS has a documented unchangeable diskette vector which can be safely used. Certain precautions must be taken to guarantee that safety. With the virus disabled, the vector chain through the virus may also be used.

WHAT ARE THE 24 CRITERIA BY WHICH WE MEASURE ANTI-VIRUS PRODUCTS? (MORE OR LESS IN THE ORDER OF EXPECTED OCCURRENCE)

We feel that epidemic boot sector viruses should be given special attention, beyond simple removal. Over time, we have developed a set of criteria by which we measure our own and other anti-virus products. We currently have 24 such criteria.

Some criteria measure solutions to technical problems. For example, all viruses can be disabled while they are active in RAM. We expect anti-virus products to disable at least the epidemic viruses.

Other criteria measure solutions to social and operational problems. For instance, we believe the most effective time to educate a virus victim is the exact time at which they put themselves at risk. The action which put them at risk must be defined simply, and the ways of avoiding future risk must be explained, also simply.

Eight items in the following list will be given extra attention during the talk. They are marked by asterisks (*).

* 1 **Detection**

There is a virus here.

* 2 **Identification**

It is this virus, with sufficient certainty.

3 **Acquire pre-removal recovery data**

Recovery is sometimes difficult without the virus present.

* 4 **Disable viruses**

Chain through all virus ISRs, or replace vectors.

5 **Report virus disabling to user and identify the virus**
Let the user know when the virus function has been stopped.

6 **Overwrite the virus in memory**
Leave ISR chains when required to guarantee operations.

7 **Restore/replace boot sector/master boot record as necessary**
Restore when a copy has been saved, replace when overwritten.

* 8 **Offer a new Master Boot Record even when no virus is present**
Only install new MBR to detect future viruses if user accepts.

* 9 **Overwrite virus data on disk with removal distinguishing data**
Users must be able to identify overwritten data easily.

10 **Restore CMOS RAM data where possible**
Some viruses protect themselves by preventing floppy boot.

* 11 **Report damaged files if possible**
Decode the FAT and report files overwritten by boot sector viruses.

12 **Restore program files where possible, back up ambiguous restorations**
Leave the choice to use the infected file up to the user.

13 **Report infected files**
Believe it or not, some products do not report properly.

14 **Report virus removals**
This too is sometimes inadequately reported.

* 15 **Describe how this virus infects PCs, and how to avoid re-infection**
The user must be educated to avoid re-infection.

16 **Provide virus Hot-Line information customizable for MIS departments**
Most companies want an internal specialist to be notified.

17 **Announce product name, version, and copyright**
Everyone does this correctly nowadays

18 **Perform all these functions optimized for speed**
Some anti-virus products are slow and/or cumbersome.

19 **Store encoded session results on original media if write-enabled**
Keep a very condensed record of results: there may be many.

20 **Allow user to write-enable for storing encoded session results**
Give users a chance to recover from a simple mistake.

21 **Offer to copy contents from virus-infected floppy to clean one**
Users need to continue using their current active diskettes.

22 **Suggest write-protection for current diskette (compliment if found)**
Keep encoded session safe for later review.

23 **Provide session results to caller**
For desk-to-desk virus removal, an easy review method is good.

* 24 **Restore RAM (if boot loaded) or suggest reboot (if COMMAND loaded)**
Bring the PC back to the GENERIC STANDARD for the OS.

We have developed these 24 criteria as a guideline. Each point helps us produce high-quality products. We add to these criteria whenever we see an opportunity to improve our virus handling. Our compliance with our own criteria is as close as possible to 100%. The criteria are demanding, and we are occasionally forced to drop one or two points for certain viruses where the criteria cannot be met. For instance, the AntiCMOS virus overwrites the MBR. An original MBR cannot be recovered. We have written a better-than-original replacement MBR which obeys all known constraints on content and function, both documented and undocumented.

General anti-virus products must cover thousands of viruses, regardless of their rarity. Many of these products have truly important and exceptional qualities which address virus problems that we do not. We continue to recommend purchasing at least one top-rated anti-virus product and keeping it up-to-date, to use in addition to our product, for better overall coverage.

However, when we compare our score on these 24 criteria with other anti-virus products, we find that we stand apart. This is understandable, given the narrower technical objective we have set for ourselves. We focus strictly on those viruses which are considered epidemic or everyday nuisances.

Our *Lotus* experience has taught us that criteria other than these 24 are far less valuable on a day-to-day basis. Mostly, what our customers demand is immediate and complete relief from epidemic boot sector virus attacks. We think *OverByte* answers this demand well with *DisQuick/ViRemove* diskettes.

WHAT DOES OVERBYTE PRODUCE AND WHAT ARE ITS FEATURES?

We produce pre-formatted diskettes for general sale, and special formats for third-party software companies. These formats and contents are protected under pending copyrights, trademarks, and patents.

DQ™, *DisQuick*™, *ViToler8*™, *ViRemove*™, *DQExpert*™

DisQuick optimizes diskette I/O increasing Read/Write speeds by 30%+ (on a full diskette, 40 seconds is saved on combined Read/Write)

ViToler8 prevents data from begin stored in areas viruses damage (virus tolerance means that the probability of data damage is reduced)

DisQuick launches compliant applications from special boot sectors (we work with other anti-virus companies to launch their products too)

ViRemove detects, identifies, disables, and removes viruses, (we update this to handle more viruses regularly)

DQExpert debugging allows expert user intervention of viruses (we develop new features to allow easy interception of new viruses)

Booting from a *DQ* diskette causes educational text to be displayed. The display program adapts the text to the immediate user needs. Fast use by corporate MIS is easy. Slow comprehensive reading is also.

Our focus on responding to epidemic boot sector viruses at boot-time makes *OverByte* products quite different from other anti-virus programs. Many of our anti-virus operations may be done safely only at boot-time.

Along with *DisQuick/ViRemove*, we have provided a boot-time debugger. *DQExpert* allows the virus professional to examine the PC in detail. New viruses can be analyzed, disabled, and removed before the OS runs. The features of *DQExpert* have been optimized for virus investigation.

WHAT DO WE RECOMMEND TO PREVENT OR WIPE OUT EPIDEMIC BOOT SECTOR VIRUSES?

Identify behaviours that cause virus spread. Educate users to avoid these behaviours. Use all available tools to facilitate this education and behaviour prevention. Determine what features an anti-virus needs to be useful in your organization. Buy the best general anti-virus product with those features. Install it properly and keep it up to date. In addition, we feel that when the majority of diskettes used in an organization have *DQ* technology, the number of virus attacks will decrease dramatically.

OverByte Corporation welcomes licensing to and industrial partnerships with other anti-virus companies.

REFERENCES

[1] Undocumented PC (Frank Van Gilluwe, Addison Wesley)

[2] Undocumented DOS (Andrew Shulman et al, Addison Wesley)

[3] System BIOS for IBM PC/XT/AT Computers and Compatibles (Addison Wesley)

[4] Pentium Processor User's Manual (Intel)

[5] DOS Technical Reference (IBM)

[6] MS-DOS Encyclopedia (Microsoft, Microsoft Press)

[7] PC Interrupts (Ralf Brown & Jim Kyle, Addison Wesley)

[8] The Programmers PC Sourcebook (Thom Hogan, Microsoft Press)

[9] Zen of Assembly Language/Code Optimization (Michael Abrash, Coriolis)

[10] The Waite Group's MS-DOS Developer's Guide (Howard Sams & Company)

ABOUT LOTUS

We will maintain close ties with *Lotus Development Corporation.* Of course, *Lotus* is to be held harmless regarding *OverByte* issues.

Trademarks: *DQ*™, *DisQuick*™, *ViToler8*™, *ViRemove*™, *DQExpert*™

SECURING DOS

Neville Bulsara

Quantum System Software, 52 Regency Chambers, Near Nandi Theatre, Bandra (W), Bombay 400 050, India

Tel +91 22 643 1233 · Fax +91 22 642 2182 · Email neville.bulsara@f1.n606.z6.fidonet.org

ABSTRACT

Ever since the arrival of viruses on the MS/PC-DOS platform, various methods have been adopted to deal with the problem. These methods (virus specific and non-specific) have met with varying degrees of success (or failure!).

What with the increasing number of viruses and the threat posed by the mutation tools looming on the horizon, the problem of glut threatens to swamp the developers of virus specific solutions.

While the importance of scanners cannot be understated, the need for generic solutions increasing seems to be a foregone conclusion. One of the 'weapons' in the armoury of the developers of generic products is the behaviour blocker. Integrity checkers attempt to plug the holes left open by behaviour blockers.

This paper aims then to highlight the fact that existing behaviour blockers and integrity checkers fail to 'SECURE DOS' effectively. It also provides an insight into why we find ourselves in this sorry state!

This paper highlights the author's argument that an effective way to SECURE DOS (as Microsoft seems not to be concerned with the virus menace), would involve parking oneself between the 'D' and the 'OS' of DOS - *that is, intercepting viruses at a level* below *the generic OS calls for file I/O and at a level above the actual physical disk.*

The paper goes on to explain precisely what happens in the 'innards' of DOS with respect to translating file I/O to actual physical disk I/O.

*The paper puts forward a new generic method which the author feels, in conjunction with existing methods of scanning and checksumming, would offer a fair level of robust security. The method would involve the creation of a '**Safe Zone**' (non-writable) on media which would be host to all executable entities. This zone would effectively be a* 'disk within a disk' *(a mountable volume à la DOS's CFV's), which would be mounted via a device driver.*

An insight into how such a device driver would function so as to enable it to determine without user intervention (since the elimination of the associated nuisance of a behaviour blocker in this case is a primary design goal), whether a write to the Safe Zone (read that as creation/modification of an *executable) is legitimate or not etc., is explained in detail. Various methods are discussed to deal with the presence of companion viruses and EXE file header infectors on the Safe Zone.*

The paper ends by highlighting the possible problems which could crop up with such a system in place and how effective such a solution would be under NetWare and Windows.

PREFACE

First things first - **THIS PAPER IS TOO LATE!!** As a matter of fact, **this subject itself is too late.** Securing DOS - at a time when DOS as we (or rather I) understand it is on it's way out? Sure, one may say that Windows '95 will continue to support DOS applications. Yes, it will, but it's still not DOS (as I see it), and hence I repeat - this paper is too late! However, the idea behind it is not.

As we all know, a large number of methodologies exist for combatting the virus menace - Scanners, Behaviour Blockers, Checksummers etc. All of these have their advantages - and their disadvantages. While it is not my intent to debunk any method which one may employ to safeguard their systems, it is only fair that we highlight the major problems associated with the above.

The greatest problem with scanners is that they are able to detect only known viruses. A heuristic scanner may be able to detect an 'unknown' virus, but then there are ways of bypassing such scanners. As someone put it - *'No virus was ever first detected by a scanner'*. The problem with scanners then is that they are **reactive** - someone has to get infected before the scanner is updated to handle a new virus. Nevertheless, because of their irreplaceable propose of detecting known viruses, a scanner is a compulsory weapon in an anti-virus armoury.

The greatest advantage of a checksummer is that it can *'detect unknown viruses'*. As we all know by now, checksummers detect changes, not viruses! Sure, a change may be due to a virus (and then, it may not!). Hence, checksummers are an important tool in generic virus detection. The problem with them, however, is that they are **even more reactive than scanners**. A file must become infected (with a 'known' or 'unknown' virus) before the checksummer does its work. Another major problem with checksummers is that they are liable to be 'led up the garden path' by stealth viruses - unless they use some very low-level routines to bypass the operating system in order to process files.

The only pro-active weapon in the anti-virus armoury is the behaviour blocker. These look out for *'virus-like activity'* rather than viruses. That is to say that they *aim* to 'stop the virus at the letter v' - when it tries to go memory-resident, infect files etc, etc. The problem - or rather the problems - are as follows:

(a) they leave the decision in the hands of the user

(b) they raise false alarms

(c) **they can be bypassed - VERY easily bypassed.**

What, then, is the way out? Do we live with this state of affairs? Or is it time to look at the problem from a different angle - perhaps even redefine the problem? Perhaps the problem can be additionally tackled at a *different level.* Perhaps it's time to redefine the rules of the game!

The problem with trying to solve the problem (no pun intended!) by redefining the rules in the middle of the game is - well, you'll see that for yourselves!

Having been a 'hard-core' DOS programmer (so they tell me) for the last 11+ years and an anti-virus researcher (now THAT, I am!) for the last seven, if 'Securing DOS' is one of the topics of discussion, here then is a *proposed* method of doing so - for better or for worse.

Around half of this paper is devoted to exploring in depth the way DOS handles files. You will find a regular sprinkling of several Data Structures maintained by DOS. At first glance it might seem that this piece is devoted to a dissection of the innards of the DOS filing system, rather than a paper on 'Securing DOS'. ***However, it is not possible to SECURE DOS without knowing what DOS is.*** And the filing system is what DOS is all about (or at least a majority of it is).

I would like to clarify that, in this paper, I'm not suggesting a rigid solution. As a matter of fact, there are no solutions in here. This paper raises more questions than it answers; questions which I've asked myself over the past few years. Some of them I've managed to answer. The rest are for you to ponder. If these questions lead anyone to devise a solution, similar or drastically different from the one I propose, this paper will have served its purpose.

Iacta alea est: The die is cast.

WHEN ARE VIRUSES A PROBLEM?

I became involved with computer viruses in 1988. Back in those days, I used to teach at a computer training institute. They happened to get hit by the Brain virus. It was a bad situation, 60+ infected systems (they used simple PCs - not even XTs - in their training labs back then). All disks were infected, including the student's floppy disks - it wasn't funny! Anyway, it so happened that I was the only person around who could perhaps do anything about it - I did. The rest as they say, is history.

Since then I've served as a consultant to several large organizations, the Government and some Defence Establishments. Over the years, I've come to a conclusion that *viruses (by themselves) are not a problem.* By themselves they're just like any other software or hardware glitch. You're just unlucky if it happens to hit you.

The problem is **THE PROLIFERATION OF A VIRUS**. I've always encouraged organizations that have had a major outbreak to try and reconstruct the sequence of events which led to that outbreak. In over 98% of the cases, it turned out that the virus came in on a single floppy disk. From this entity, it jumps to usually just one system (so far this is NOT a problem you can't cope with). From this system it jumps to other disks and from these to other systems (it could also go over a network). By the time you detect the intruder, scores of systems are infected. *NOW you have a problem on your hands!*

I've always stated that the **earlier** you detect the intruder, the **faster** you can deal with it, saving oneself much heartache later.

The trick then lies in ***DETECTING THE INTRUDER AT THE EARLIEST AND PREVENTING IT FROM PROLIFERATING!***

MY CONCEPT OF SECURING DOS

My concept of securing DOS is securing the executable entities which can be infected by a virus. For the purpose of this paper, I've restricted myself just to securing executable files. Boot Blocks, by comparison, are far easier to secure.

When I talk of securing executable files (before I proceed further and I actually detail how this can be done), it is important that we have a clear understanding of just how DOS deals with files - especially since what we are going to end up doing is securing files.

HOW DOS KEEPS TRACK OF FILES

DOS keeps track of files on a volume using its **file system.** The DOS file system consists of the File Allocation Tables (FATs) and the Root Directory. The Root Directory can contain both files and subdirectories. Each subdirectory can also contain files or further subdirectories.

For the next couple of lines (unless stated otherwise), I will refer to a file as an 'Entity'.

Every entity has an 'entry' (hereafter referred to as a Directory Entry or **DirEntry**) in either the root or a subdirectory. That DirEntry is the starting point from which DOS manages that entity. From the DirEntry, DOS knows (amongst other things) where the file actually starts on the disk, and its length (the file size).

If the DirEntry provides information as to where the file actually starts, the FAT provides information as to where it resides as a whole on the volume. For a detailed understanding of how the DirEntry and FATs are used to keep track of a file, one can refer to several books on the topic.

Suffice it for now to state that the Directory Entries and the FAT are used by DOS to keep track of the actual layout of files on a given medium.

While it is not important for us to know the FAT structure, it is important to know precisely what the DirEntry looks like, as we will be employing it in our scheme. Each Directory Entry is 32 bytes in length and has the following structure:

Table 1 : DirEntry (Directory Entry) Structure

```
DirEntry          STRUC

DE_PrimaryName    db 8 dup (' ')       ; 8 byte primary name

DE_Extension      db 3 dup (' ')       ; 3 byte extension

DE_FileAttrib     db ?                 ; 1 byte for file attribute

DE_Reserved       db 10 dup (?)        ; 10 bytes reserved field

DE_FileTime       dw ?                 ; 2 bytes (word) file time

DE_FileDate       dw ?                 ; 2 bytes (word) file date

DE_Cluster        dw ?                 ; 2 bytes (word) starting cluster

DE_Size           dd ?                 ; 4 bytes (dword) file size

DirEntry          ENDS
```

As you can see, the DE_Cluster (starting position of the file) and the DE_Size fields in a DirEntry and then using the FAT, DOS is able to get to a required file.

Since we've got around to defining some structures, we might as well define another - especially as we're going to employ it in our proposed scheme. We'll call this structure the **Current Directory Structure (CDS).** A CDS is maintained for each and every drive in the system. Each CDS Entry **(CDSEntry)** contains the current working directory for that drive (which explains why DOS doesn't forget which directory you're in on drive C when you go to drive A or whatever!). Apart from the current working directory, the CDSEntry contains a pointer to the device driver to be called for performing actual disk I/O on that media; bit attributes which specify the availability of the disk, whether it is JOINed, SUBSTituted, or a network drive, etc.

The CDS is built at boot time by DOS as it processes CONFIG.SYS. If the said file has LASTDRIVE=F, DOS builds CDSEntries for drives A through F. If there were no drives beyond C, the bit attributes for CDSEntries for drives D thru F are masked to indicate that those drives are invalid or not available.

The structure of each entry in the CDS is as follows:

Table 2 : Structure of Entry in CDS (one for each drive till LASTDRIVE)

Structure valid for DOS 4+

```
CDSEntry            STRUC

CDS_CurrentPath     db 67 dup (0)    ; 67 bytes to hold current path for the drive in the
                                     ; form: C:\DOS\VERSION5. The path is null
                                     ; terminated

CDS_DrvAttrib       dw ?             ; 2 bytes that hold flags to indicate whether drive
                                     ; is physical, network JOINed, SUBSTituted etc.

CDS_DPB_Ptr         dd ?             ; far pointer to the Drive Parameter block for this
                                     ; drive

                          For local drives

CDS_StartCluster    dw ?             ; start cluster of current directory

CDS_Unknown1        dd ?             ; unknown

                          For network drives

CDS_Redirector      dd ?             ; far pointer to Redirector

CDS_UserData        dw ?             ; user data from 21h/5f03h

                          For all drives

CDS_SkipCount       dw ?             ; holds count of bytes to skip over when
                                     ; displaying the current directory. Normally 2 so
                                     ; that the drive and the colon are masked.
                                     SUBST and JOIN change this so that only
                                     ; appropriate parts are visible

CDS_Unknown2        db ?             ; unknown

CDS_IFSPtr          dd ?             ; far pointer to IFS driver

CDS_Unknown3        dw ?             ; unknown word

CDSEntry            ENDS
```

Phew! That much information, just so that DOS can remember what the current directory for a drive is? Well, it has some other purposes too, as we shall see below.

INSIDE THE MSDOS FILE SERVICES BY ...

'Inside the IBM PC', by Peter Norton, was the first book which delved into the innards of the PC. A great many programmers got their first taste of low-level activity by reading that book. Before that, it was just a case of:

'That box there does it (though it ain't black!). We don't know how it does it, but it does. So there!'

That's what we're all prone to do. Take things for granted. The classical **'black box' approach.** Supply the (correct) inputs and get the desired output. The black box approach guarantees correct results (we hope!). There's nothing wrong with this approach, but it's taboo to a hacker. A hacker needs to know what makes the box tick. Till he figures that out, well - hopefully you know what I'm talking about!

Let's consider the typical black box approach when it comes to doing things (opening/reading/writing/ closing) with a file. If we use the DOS API, all we need to do is (a) Open the file, (b) Read from it, (c) Write to it and (d) Close it. Pure and simple. You call DOS with the correct inputs, DOS does its work. **You're OK, DOS is OK** (with due apologies to the author of I'm OK, you're OK).

But what happens inside DOS when let's say you open C:\COMMAND.COM, read from it, write to it and then close it? Here's what happens in reality:

- DOS indexes into the current PSP to locate a free entry in the **JFT** (Job File Table). If no free entries exist, DOS returns, indicating that there are too many open files.
- Having found a free entry in the JFT, DOS remembers the position of this entry. This index into the JFT eventually will become the **'handle' or the JFN** (Job File Number) which will be returned to your application, assuming that the Open is successful.
- DOS goes through its **SFTs** (System File Tables - more on this later) looking for the first free SFT entry (SFT_Entry). If no SFT entries can be found, DOS returns, indicating an error.
- DOS parses the filename to determine the drive on which the file is supposed to exist (in our example it would be C).
- DOS goes through the **CDS** (remember the CDS?) to determine whether the said drive exists and is valid. If not, DOS returns with an error.
- DOS examines the CDSEntry for that drive to determine whether the drive is ***networked.*** If it is not, DOS uses CDS_DPB_Ptr to derive the address of the Drive Parameter Block (DPB) for that drive. The DPB is used by DOS in order to locate the root directory and the address of the device driver to be called, in order to actually do the low level disk i/o.
- DOS calls the device driver for that drive to read the root directory. It processes what is read to look for COMMAND.COM. If not found, DOS returns with an error code. Remember at this point that what is read is a series of entries of type DirEntry.
- Having found the required file, DOS updates the SFT_RefCount in SFT_Entry. ***The index of this SFT_Entry within the SFT itself is referred to as SFN (System File Number).*** DOS then updates SFT_Entry with information such as the filename, starting cluster, initial open mode, address of the device driver etc.
- The SFN for this file is copied into the index into the JFT (JFN) which DOS had determined at step 2.
- The JFN is returned to the application as the file handle.

Let us assume that the handle returned on the Open request was 'X'. Whenever you wish to refer to this file subsequently, you just call DOS with one of the inputs as X. This is what happens when you try to read from COMMAND.COM:

1. DOS uses X as an index into the JFT to determine the SFN. Having found the SFN, DOS indexes into the SFT in order to locate SFT_Entry for that file.
2. DOS calls the device driver, whose address is stored in the SFT_Entry, to read the block of data.

3. DOS returns any necessary information to the application.

When you try to write to the file, the following happens:

1. Identical to step 1 above
2. DOS examines the initial open mode field stored in that SFT_Entry to determine whether you can write to the file (to prevent writing to files opened in read mode). If not, DOS returns with an error.
3. DOS calls the device driver to write data.
4. DOS returns any necessary information.

When you close the file:

1. You got it! It's identical to step 1 above.
2. DOS calls the device driver to flush any data it might have had in its internal buffers. This includes the updated DirEntry with the new Date/Time stamps, etc.
3. DOS decrements the SFT_RefCount field to indicate that SFT_Entry is now free for use.

All that just to open, read from, write to and close a file? Yep! No one said it was going to be easy!

Since we've been discussing SFTs, we might as well shed some more light on the topic. *An SFT is nothing but a table of SFT_Entries. Each file which is open at any given time under DOS has an entry in the SFT. If Dir_Entry is the structure by which DOS manages a Directory Entry on disk, SFT_Entry is the structure which enables DOS to manage all file-related activities for a given file.* Below is the layout of a System File Table entry:

Table 3 : Structure of System File Table Entry (SFT_Entry)

```
Structure valid for DOS 4+

SFT_Entry   STRUC

SFT_RefCount          dw ?          ; count of number of file handles referring to this file

SFT_InitialOpenMode   dw ?          ; initial mode (read,write etc) in which the file was
                                    ; opened.

SFT_FileAttrib        db ?          ; the attribute of the file on disk field is filled up from
                                    ; DE_FileAttrib; during the open.

SFT_DeviceInfo        dw ?          ; this word holds various bits indicating whether the
                                    ; file is remote, the drive number on which the file
                                    ; resides, etc.

SFT_Ptr1              dd ?          ; far pointer to device driver header or DPB or REDIR
                                    ; data

SFT_Cluster           dw ?          ; starting cluster for file - field filled up from
                                    ; DE_Cluster on Open

SFT_FileTime          dw ?          ; file time (taken from DE_FileTime).This field is
                                    ; updated whenever the file is written to

SFT_FileDate          dw ?          ; file date (taken from DE_FileDate). This field is
                                    ; updated whenever the file is written to
```

```
SFT_Size            dd ?              ; file size - taken from DE_Size on Open. Updated if
                                      ; neccessary when file is written to.

SFT_FilePtr         dd ?              ; file pointer - indicates where the next read or write
                                      ; will occur (relative to start of file)

                               For local files

SFT_RelCluster      dw ?              ; relative cluster within file of last cluster accessed

SFT_Sector          dd ?              ; number of sector containing DirEntry

SFT_SectorIndex     db ?              ; number of DirEntry within sector

                               For remote files

SFT_REDIRIFS_Ptr    dd ?              ; far pointer to REDIRIFS record

SFT_Unknown         db 3 dup (0)      ; unknown

                               For all files

SFT_FCBName         db 11 dup (' ')   ; filename in FCB format

SFT_Share1          db 6 dup (?)      ; information for use by SHARE

SFT_PSP             dw ?              ; PSP address of owner of the file

SFT_Share2          dw ?              ; information for use by SHARE

SFT_AbsCluster      dw ?              ; absolute address of last cluster accessed

SFT_IFSPtr          dd ?              ; far pointer to IFS driver for file

SFT_Entry           ENDS
```

But ***WHAT'S ALL THIS GOT TO DO WITH VIRUSES?***

Well, so far we've made a candle. Now it's time to light it!

WHY DO BEHAVIOUR BLOCKERS FAIL?

As I mentioned before, a behaviour blocker is the only proactive tool which can detect (hope to?) an unknown virus before it manages to infect an entity. Forgetting for the time being the associated nuisance value which comes along with one, you would think that these busters can keep out all viruses - past, present and future. Think again!

You can liken a behaviour blocker to a sentry commissioned to guard the entrances of a room. A behaviour blocker posts sentinels at every entrance (read that as **KNOWN ENTRANCES**) to the room in order to screen visitors for suspicious objects. So you post one at those two doors and one each at the five windows. Then pray like hell!

The problem is ***you didn't build that room.*** Neither did the security agency (read that as anti-virus vendor) which supplied the guards. The room (as a matter of fact the whole building) was built by that company up in Redmond. And for all you know, that room of yours could have a **trapdoor** that neither you nor the Vendor know anything about. Did I say ONE? Well, think perhaps in the hundreds! Anyway, that's the loophole an intruder could exploit to get into your system - and they do.

Let us consider a few methods used by some of the more 'novel' viruses to bypass behaviour blockers:

- Opening a file in Read mode (this allows the behaviour blocker to allow the request). Upon return from DOS, use the handle to index into the JFT; from there get the corresponding SFT_Entry; twiddle with the SFT_InitialOpenMode field to indicate that file opened in Read+Write mode.
- Use the starting cluster field in the DirEntry; process the FAT; infect the file using Int 25h/26h, thereby bypassing any Int 21h handlers set up by a behaviour blocker.
- Use the start cluster field to infect the header of EXE files. Again using Int 25h/26h bypass Int 21h.
- Using Int 13h to monitor reads to disks and 'infecting' any sector which starts with the 'MZ' indicator.
- Twiddling with the starting cluster field in the SFT_Entry in order to 'open a file, but write to another'.
- Using various tunneling techniques to derive the address of the original interrupt handlers and subsequently making far calls to them.
- Using 'reserved' interrupts/functions in order to open, read from,write to files (e.g. using the undocumented DOS Indirect Server Call).
- Infecting the file system itself - DIR-II, for example.
- Modifying the contents of DOS's buffers (infecting them), then setting their bits to indicate that the buffer is dirty, causing DOS itself to flush their contents to disk.

Get the idea? Too many **unguarded** DOS, WINDOWS, and safety NetWares (sorry that should read doors, windows and nets), about which one knows little or nothing.

THE PREREQUISITES FOR WINNING A BATTLE

Sun Tzu, the author of 'The Art of War' wrote as follows:

'Do not understand yourself? You will lose 100 percent of the time. Understand yourself? You will lose 50 percent of the time. Understand yourself and your opponent? You will win 100% of the time.'

Well we so far have understood ourselves (the DOS filing system). We understand - or at least try to - the methods cooked up by the opponents. Unfortunately, the game being as it is, it is impossible to win all the time. But at least we can try.

By now I have hopefully lit a candle or two. It is now time to feed the flame.

SECURING AN OS

As I view it, the way to secure any OS from viruses is really very simple. If a single word could describe it, the word that fits the bill would be **SEGREGATION**.

The current problem (to the best of my limited knowledge) lies in the fact that all ***Operating Systems store both data and program files on the same medium***. That is to say, both these types of files lie on the same volumes, and the OS uses the same filing system and the same file system (File system = FATs, Directories etc. Filing system = routines to manage the file system) to keep track of them. It is hence obvious that *if the OS fails to distinguish between code and data, viruses will exploit this loophole and continue to proliferate.*

The key then lies in SEGREGATING data files and program files; storing them on 'different volumes', and using different file systems and filing systems to manage them. If this segregation can be achieved, then program files can reside on a so called 'Safe Zone' - to coin a new term. This **Safe Zone** (hereafter referred to as SZ) would be a read-only volume managed exclusively by the OS. Since the SZ would be read-only, a new virus introduced into the system would be unable to spread (though it would attempt to). However, the OS could keep track of these write requests, and maintain a log. Frequent examination of this log would display strange behaviour and would indicate the fact that a possible virus is attempting to spread - albeit unsuccessfully. Presto - you have just secured your OS!

Seems simple enough - but there is a lot more to it, as we shall shortly see.

SECURING DOS USING THE SEGREGATION PRINCIPLE

Since DOS (as it stands today) has no concept of SEGREGATION, it is obvious that such a concept would need to be externally induced. I have been thinking about this concept for the last two plus years. I have conducted various experiments which have shown that it is TECHNICALLY feasible to employ such a scheme.

The seed for this idea was sown shortly after I disassembled the infamous DIR-II virus to see what made it tick. After studying the virus, I reached a conclusion that the only way to secure DOS pro-actively was by parking yourself between the D and the OS of DOS. That is to say, at a level between the OS and the actual physical media. The proposed method does precisely that.

It is said that examples bring the obscure to life. Keeping this in mind, we'll proceed with a real example of how such a system is implemented.

Let us consider a system having a single partition (drive C). As things stand, the following directories and files exist on that volume:

C:\		(root directory)
	IO.SYS	(executable entity)
	MSDOS.SYS	(executable entity)
	COMMAND.COM	(executable entity)
	CONFIG.SYS	(executable entity)
	AUTOEXEC.BAT	(executable entity)
C:\DOS		(sub directory)
	HIMEM.SYS	(executable entity)
	EMM386.EXE	(executable entity)
C:\NC		(sub directory)
	NC.EXE	(executable entity)
	NCMAIN.EXE	(executable entity)
	NC.INI	(non-executable entity)
	NC.MNU	(non-executable entity)

Our objective at this stage is the creation of a Safe Zone (SZ) which will be host to executable entities. All non-executables will reside on a normal volume called the Data Zone (DZ).

STAGE 1 - PREPARATION OF THE SZ

A program (let's call it SZPREP) is run. **SZPREP** sweeps through your hard disk determining the directory tree and collecting the names of all executable entities which can be moved to the SZ. It determines that C:\COMMAND.COM, C:\AUTOEXEC.BAT, C:\NC\NC.EXE and C:\NC\NCMAIN.EXE can be moved to the SZ. (IO.SYS, MSDOS.SYS, CONFIG.SYS, HIMEM and EMM386 cannot be moved, as they must load prior to any other drivers).

Having determined the names (and the total size) of the files which can be moved to the SZ, SZPREP creates *a hidden read-only file in the root directory of drive C.* Let us call this file SAFEZONE.NB (how imaginative!). A more generic term for this file would be a ***Safe File Volume (SFV)*** - there, I've just coined another new term!

We need to clarify at this point itself what the structure of the SFV will be like. Since it will essentially play host to files, it must have its own file system and data space. The exact structure of the file system is a matter of design. For simplicity, it can be almost identical to the DOS file system - i.e. it can have a boot sector, FAT, directory and data area. I would suggest that, instead of having the concept of sub directories, a flat directory structure, using the 10 unused bytes in DirEntry (DE_Reserved) to store the checksum of the path where this file resides (refer to Table 1) be employed.

SZPREP then moves (copy to destination, delete from source) all the above-mentioned files which are candidates for being moved into the SFV. SZPREP then plonks a file (let's call it **SZMOUNT.SYS**) into the root directory of drive C, and modifies CONFIG.SYS. Your SZ is now ready, and drive C looks like:

C:\		(Root Directory)
	IO.SYS	(executable entity - can't be moved)
	MSDOS.SYS	(executable entity - can't be moved)
	CONFIG.SYS	(executable entity - can't be moved)
	SAFEZONE.NB	**(SFV)**
	SZMOUNT.SYS	**(executable entity to manage the SFV)**
C:\DOS		(Sub directory)
	HIMEM.SYS	(executable entity - can't be moved)
	EMM386.EXE	(executable entity - can't be moved)
C:\NC		(Sub Directory)
	NC.INI	(non-executable entity)
	NC.MNU	(non-executable entity)

As we can observe, all but the barest minimum of infectable objects have 'disappeared' from the volume. In a real-world situation, hundreds of objects would have been moved into the SZ, leaving just six possible candidates for infection.

STAGE 2 - MOUNTING THE SZ

The SZ is mounted at the time when the system is booted off the hard disk. DOS processes CONFIG.SYS, and loads the SZMOUNT.SYS driver. SZMOUNT detects the presence of the SFV, reads it (in order to determine its size, volume characteristics) and then (as one would guess) implements it as a drive - à la STACKER, DBLSPACE, DRVSPACE - right?

WRONG!

In the case of the above mentioned 'drive doublers', they would either treat the SFV (in their case it is a CFV - Compressed File Volume) as C: and the original drive as D: - or vice-versa. Our software can't afford to do that, as most programs expect their non-executable components (configuration files, INI files etc) to reside in the same drive and directory as they do. As for example, NC.EXE would expect NC.INI and NC.MNU to be present where it loaded. Hence, NC would not work (as would not a large number of other programs) if we implemented our SZ as C and the original volume as drive D or vice-versa.

What we need to do is make it seem as though they exist on the same drive, which in reality they do not. This calls for a bit of magic!

Keep in mind too that we need to monitor all activities (such as DIR, Open, Read, Write, Close, Delete, MD, RD, CD, Get Disk free space etc) - that are in any way connected with disk I/O. We need to pass these requests seamlessly either to the original volume or to the SZ. And most important, both volumes must look like the same original volume. And in case we've forgotten, we've got to maintain the integrity of files on the SZ.

In a nutshell, here's what SZMOUNT does when it loads up:

- copy the CDS Entry (refer to Table 2) for drive C: (The DZ) into one of its internal buffers
- take over the **DOS interrupt 21h** (practically all the file/directory handling calls require an Int 21h)
- take over the **DOS Multiplex interrupt (Int 2Fh)**
- determine the address of the DOS Swappable Data Area (SDA - more on this later).

That is all that SZMOUNT needs to do when it loads up. The rest, which is the integration of the SZ with the DZ, is sheer magic!!!

STAGE 3 - SEAMLESS INTEGRATION WITH THE DZ

Stage 3 comes into play the moment Stage 2 is over. It works as follows:

On the occurrence of any Int 21h, control comes to the Int 21h handler inside SZMount. Again, as it is far simpler to explain using examples, I shall supply a few.

Let us suppose that SYSINIT (part of the DOS startup-code) finished processing CONFIG.SYS. It then needed to allocate some memory. SYSINIT issues an Int 21h, with the AH register = 48h. This is trapped by SZMOUNT's Int 21h handler. SZMOUNT goes through a lookup table, and determines that ***this function does not deserve its attention, and so LOWERS a flag, and passes control to DOS to do the necessary.***

On the other hand, let us suppose that the request was to **open** a file (let us say C:\NC\NC.MNU). SZMOUNT determines that an open file request is ***one of the many requests which is deserving of its attention. So, it RAISES a flag and passes control to DOS.***

If we recollect our earlier discussion 'Inside the MS-DOS File System', DOS essentially checks the CDS, determines that C: is a valid drive, calls the device driver associated with the drive, fills up a SFTEntry etc,etc, and then returns with a handle (JFN) specifying that the open was successful.

At this stage, control comes back to SZMOUNT before it returns to the application (as DOS was called by SZMOUNT which was called by the Application). At this point, SZMOUNT ***checks whether its flag is in a raised state (it is).*** Now, SZMOUNT ***checks whether DOS has returned an error (it hasn't).*** So, SZMOUNT LOWERS its flag and returns to the application. This is how SZMOUNT manages file opens on the DZ (actually, DOS does). As a matter of fact, this is how SZMOUNT manages any file/directory related activities on the DZ.

What then, if an application wanted to ***open C:\NC\NC.EXE. Well, the flag is RAISED, control passes down to DOS but DOS can't find an NC.EXE, and so returns to SZMOUNT with an error. SZMOUNT sees that its flag is raised (a raised flag indicates POSSIBLE activity may be required on the SZ) AND DOS has returned an error. This tells SZMOUNT that it should attempt to repeat that activity on the SZ***. And at this point the Int 21h handler of SZMOUNT begins its real juggling act.

First, the Int 21h handler ***twiddles with the CDS entry for drive C, indicating that the drive is a NETWORK drive.***

Second, the handler refers to the DOS SDA to find out the contents of the various registers when DOS's Int 21h handler got control of the request (the contents of registers, along with other information is stored within the SDA). The registers are reloaded with the stored contents.

Lastly, SZMOUNT lowers its flag and REISSUES the call to DOS!

This time across, DOS examines the CDS for drive C, and discovers that it's a NETWORK (remote) drive. A NETWORK drive, as far as DOS knows, may not have a DOS compatible FAT, and can be implemented by any vendor he/she deems fit.

In order to integrate alien not-FAT file systems with DOS, DOS provides hooks by which it calls programs to implement remote drives. These hooks constitute the DOS ***Network Redirector Interface***, and a program using these hooks is called the Network Redirector. The Network Redirector interface is called via Int 2Fh (that is why SZMOUNT captured that interrupt, too) with the AH register loaded with the value 11h, and the AL register loaded with the sub-function (Open, Read, Write, Close etc.)

To cut a long story short, DOS, finding that the drive is a network drive, issues an Int 2Fh with AH=11h and AL=16h (open existing file). DOS also passes a pointer to an uninitialized SFT along with other relevant information in the SDA.

On receiving control via Int 2Fh, SZMOUNT determines the function, derives a pointer to the filename (C:\NC\NC.EXE), reads the file system from the SZ into memory (at the disk level, not making any DOS calls!), and ending the file, fills information into the supplied SFT. At this point, SZMOUNT twiddles with the SFT entry for that file to indicate that this file is a remote file. SZMOUNT then twiddles with the CDS for drive C, to indicate it is a physical drive, and returns. Application wanted the file opened; SZMOUNT (and not DOS) has opened it for you.

Similarly, if you issued a 'DIR C:\NC' command, first the files on the DZ followed by the files on the SZ would be displayed - despite the fact that they lie on seperate volumes! Well, actually, a DIR (that translates into a series of find first/find next calls to the OS) requires a bit of special handling (as do the remove directory and the rename functions), but it is possible to do it.

Reads (and writes) do not raise the flag in the Int 21h handler of SZMount. The handler does not need to meddle with the CDS to indicate that the drive is networked. A read request on a file will result in DOS issuing an Int 2Fh, AH=11h, AL=8 if the SFT for that file indicates that the file is remote. Hence, if a file on the SZ needs to be read from (or written to), the Redirector gets control and does its job (actually reading but never writing).

Similarly, various functions to take care of setting attributes, deleting files, managing directories, etc, can be seamlessly integrated as ***DOS will always call the Redirector when it needs to access 'networked drives'.***

WHAT OF A VIRUS TRYING TO INFECT FILES ON THE SZ?

- If a virus tried using one of the standard (or even undocumented) DOS calls to write to a file, an Int 2Fh would nevertheless be invoked
- If a virus opened a file in Read mode and then changed the SFT entry to Read Write, this would not matter, because writing to the SZ is prohibited, as implemented by SZMOUNT
- The file system as used by SZMOUNT is alien to the standard DOS file system. Hence, no virus would be able to infect files by manipulating the file system directly
- For viruses which infect the header of EXE files, it is very simple to stop them in their tracks by encrypting data before it is written to the SZ and decrypting it (in memory) in case DOS needs to read it
- Viruses can tunnel their way to 'Kingdom come' and derive the address of DOS for all we care, as we are operating at a level below DOS
- File system infectors would fail on an alien file system (they wouldn't even see it!)
- Any virus trying to infect would end up ultimately calling the Redirector. The Redirector could log these attempts and, in the event of a threshold being crossed, could sound an alarm indicating the presence of a virus.

As we can see, by SEGREGATING our code and data, we achieve the purpose of SECURING DOS. Unfortunately we achieve it too well...

PROBLEMS WITH THE APPROACH

This proposed solution (if it can be called one) raises quite a few questions (as I'd promised!) which remain unanswered. But then at times, I seek not to know the answers, but to understand the questions.

- **What of our six files on the DZ: how do we secure them?**

  I DON'T KNOW!

- **What of self-modifying files?**

  You know, it is becoming increasingly difficult to convince me that we ought to tolerate companies which in this day and age write programs that modify themselves.

  But then, I guess that it does not speak very highly of the anti-virus community when the publisher of a certain product still calls its configuration file - what was it? I forget. Something like.. *someCFG.BIN.*

  Well, the only thing to do with such files is to keep them on the DZ. And pray that people learn!

- **What happens when you update software?**

  I knew you were gonna ask that one. Quite frankly, this is a sticky situation. My proposal is, treat the update as you would treat a create request. Let the executable get created on the DZ. At the earliest, move it to the SZ (but ask for user permission before doing so - especially when you're overwriting existing files on the SZ).

  Files created on the DZ are vulnerable until moved to the SZ. If it is any consolation, only these files can get infected, they cannot spread the infection to other files (THAT IS NO CONSOLATION!).

- **What happens with files on CFVs created via STACKER etc?**

  The concept can be extended on such volumes too - though you'd have to call upon the device driver that manages that drive to perform disk I/O. That is a negative point; it makes that volume vulnerable!

- **What of disk utilities?**

  Disk utilities (CHKDSK/NDD/NU etc) should work just fine. This is because, at any given instance, an application sees only the host drive (the DZ). None of the executable entities would be visible at a sector level.

- **What of development environments?**

  Programs under development should be kept in the DZ until you're done with them.

- **What of executables with non-executable extensions?**

  Bad habit - and bad manners! It is time to change.

- **What of DOS files residing on a NetWare server?**

  I see no reason why the Segregation principle cannot be employed at the server level.

- **What of Windows?**

  Yikes! You got me there! I'm no cat at Windows programming. So I can't answer this one. But some educated guesses...

  Windows 3.11 - with 32 bit file access - no way is that going to work! But perhaps a VxD would. Someone needs to explore this further Windows '95 - hey we're talking of Securing DOS here! Remember I said that it may run DOS apps, but it still ain't DOS. It ain't - the file system is different, which is what we're discussing here.

- **What of targeted attacks against this method?**

  The encryption/decryption of the SZ data should be variable. So should the name of the file which actually holds the SFV, and the name of the driver that implements it.

  The driver should be almost 64K (on disk). This prevents it from being infected by a device driver infector. To safeguard it against a possible future device driver infector, it should avoid having repeated sequences of bytes (no large uninitalized buffers).

  Perhaps the driver itself should be polymorphic - with apologies to Alan, Vesselin & Fridrik (in no particular order) - I hope they don't strangle me for suggesting this. But this reduces the possibility of a targeted attack.

- **Is the method safe?**

  I've been an Assembler programmer for as long as I can remember. And despite the fact that my programs may have the declaration:

  'assume cs:code, ds:code ..'

  what precedes this is a reminder to myself which reads...

  ';; —— assume NOTHING ——' (Neville Bulsara's first law of programming).

  To be quite honest, the method described makes a lot of assumptions. It assumes that the documentation of the Network Redirector Interface is accurate (it is reputed to be undocumented even within *Microsoft*!). As for example, most of the information on the Network Redirector is to be found in the book, 'Undocumented DOS'. One would tend to take this information at face value. Beware - mistakes do occur. Case in point - Schulman states that Novell uses the Redirector in versions 4 and above of NetWare. Actually, it started with Version 3.12! We also assume that Microsoft will not change the data structures overnight. All these assumptions make me a wee bit nervous...

But then as I said before, if someone carries the ball from here, and takes the segregation principle to its logical conclusion, the paper will have solved its purpose.

CONCLUSION

In theory, the Segregation principle may seem the best way of securing executable files. The scheme is best used if implemented within the OS itself. However, as mentioned, at all times must executables and non-executables appear to lie on the same volume. The actual juggling, must be done at a level lower than the OS's file I/O API.

The method can (can it?) be implemented within any OS. My only regret is that I thought of this method of securing DOS when DOS finally seems to be on its way out!

ACKNOWLEDGEMENTS

I would like to thank the following people for providing inputs which directly or indirectly contributed to this piece:

Dr. Alan Solomon - for discussions on viruses which extended till the wee hours of the morning.

Vesselin Bontchev - whose suggestion that I submit a paper made this possible.

Chetan Varde and Jhankar Shah from back home - for thinking along lines as crazy as I do.

FURTHER READING

Advanced MSDOS - by Ray Duncan
This book gives an accurate description of the MS-DOS file system.

Undocumented DOS - by Andrew Schulman
For descriptions of the Network Redirector and various structures such as the CDS, SDA, SFTs, etc.

MODERN METHODS OF DETECTING AND ERADICATING KNOWN AND UNKNOWN VIRUSES

Dr. Dmitry Mostovoy

DialogueScience, Inc. Computing Center of the Russian Academy of Sciences, 40 Vavilova Street,
Moscow 117967, Russia.
Tel +7 095 137 0150 · Fax +7 095 938 2970 · E-mail dmost@dials.msk.su

ABSTRACT

Viruses are growing in number from day to day, so it is obvious that soon anti-virus programs like NAV or MSAV will not be quite efficacious. Therefore, we started designing a program that would annihilate not individual infectors, but viruses in general, regardless of whether a virus is known or not, or whether it is old or new.

The first outcome of our efforts in this direction, ADinf (Advanced Diskinfoscope), is a forecasting center which alerts the user in advance with great reliability about the intrusion of viruses, even HITHERTO unknown infectors. As distinct from all other data integrity checkers, ADinf inspects a disk by scanning the sectors one by one via direct addressing of BIOS without the assistance of the operating system and takes under check all vital parts of hard disk. To evade such detection tactics is almost impossible.

ADinf alerts the user in time about virus intrusion and restores infected boot sectors. How to restore the infected files automatically? Our next step was to produce a curing companion to ADinf. The new tool, ADinf Cure Module, deploys a novel strategy. Paradoxically, ninety seven percent of the viruses in our collection fall under a few standard groups by the types of infection methods. New viruses are as a rule designed on one of these common infection principles and, therefore, ADinf Cure Module will be about 97% efficient in its performance also in the future.

ADinf and ADinf Cure Module are parts of DialogueScience anti-virus kit - the most popular anti-virus in Russia.

INTEGRITY CHECKING

The basic classes of anti-virus programs are well known. They are scanners/removers, monitors, and vaccines. I would like to discuss the development of programs to which, in my opinion, anti-virus designers undeservedly pay little attention. This class of anti-virus programs is known as 'integrity checkers', although the name does not fully characterize the programs' policy which we describe below. This is the only class of purely software anti-virus protection, which permits the detection of known and unknown viruses with reliability approaching 100% and eradication up to 97% of file infectors, including hitherto unknown viruses.

The operation of integrity checkers is based on a simple fact: even though it is impossible to know all information about potentially infinite number of viruses, it is quite possible to store a finite volume of information about each logical drive in the disk and to detect virus infection from the changes taken place in files and system areas of the disk. As already mentioned, the name 'integrity checker' does not fully reflect the essence of these programs. Infection techniques are not restricted to a simple modification of the program code. Other paths for infection either already exist or are also possible. For example, companion viruses [1]. A disk can be corrupted by restructuring the directory tree, say, by renaming the directories and creating new directories, and by other such manipulations. Consequently, to provide reliable protection, integrity checkers must take care of far more parameters that the mere changes in the size and CRC of files as is done by most programs of this class. Thus, master boot record (MBR) and boot sectors of logical drives; a list of bad clusters; directory tree structure; free memory size; CRC of Int 13h handler in BIOS; and even the Hard Disk Parameter Tables must be under the control of integrity checkers. Changes in the size and CRC of files, creation of new files and directories and removal of old files and directories are obviously objects for strict control. A designer of an integrity checker must be one step ahead of virus designers and block every possible loophole for parasite intrusion.

Despite the large amount of controlled information, an integrity checker must nonetheless be user-friendly, simple in usage, and quick in checking disks. It must at the same time be user-customizable as regards the levels of messages displayed on the changes occurred in the disk and be capable of conducting a preliminary analysis of the changes, particularly the suspicious modifications such as:

- changes in size and CRC of files without any change in datestamp
- illegal values of hours, minutes or seconds in the datestamp of infected files (for example, 62 seconds)
- year greater than the current year (certain viruses mark infected files by increasing the year of creation by 100 years, which cannot be detected visually because 'dir' command only displays the last two figures of the year
- any changes in files specified in the 'stable' list
- change in master boot record or boot sector
- appearance of new bad clusters on the disk and others.

Let us now discuss the main problems faced by a designer of 'integrity checkers'. First, there is the dodging ability of viruses based on stealth-mechanisms. Integrity checkers that rely on operating system tools in their scanning mission are absolutely helpless against this class of viruses. They have stimulated the development of an integrity checker that checks disks by reading the sectors via direct addressing through BIOS. Stealth viruses cannot hide the changes in an infected file size; on the contrary, under such a scanning technique the stealth-mechanism betrays the presence of known and hitherto unknown stealth viruses through the discrepancy between the information given out by DOS and the information obtained by reading via BIOS. Such algorithms have been created and successfully detect the appearance of stealth-viruses.

Scanning a disk by reading the sectors by direct addressing of BIOS has one more important merit which is often overlooked. If a computer is infected by a so-called 'fast infector' [1], (i.e., a virus that infects files not only when they are started, but also when opened), such an integrity checker will not spread the infection to all files in the disk, because it does not at all address the operating system for reading a disk via sectors and uses an independent file opening system, preventing the virus from getting any control.

Finally, an integrity checker utilizing direct reading of sectors is twice as fast at checking a disk than any other program that relies on the operating system tools, because a disk scan algorithm can be created that reads each sector only once and optimizes the head movements.

Disk handling via BIOS has its own hurdles. The foremost problem is the compatibility with the large number of diverse hardware and software, including disk compactors (Stacker, DoubleSpace), specialized drivers for accessing large disks (Disk Manager), SCSI disk drivers etc. Furthermore, there are many MS-DOS compatible operating systems that have imperceptible but quite important features in partitioning logical drives. Integrity checkers must pay due attention to these fine factors.

VIRUS REMOVAL TECHNIQUES

Modern integrity checkers are useful not only in detecting infection, but are also capable of removing viruses immediately with the help of the information they retrieve from an uninfected machine at the time of installation. An integrity checker can kill known viruses as well as the viruses which were unknown at the time of creation of the integrity checker.

How this is done? Most obvious are the methods for removing viruses from the master boot record and boot sectors. An integrity checker stores images of uninfected boot sectors in its tables and in case of damage can instantly restore them. The only restriction is that the restoration must also be effected via direct addressing of BIOS and after restoration, the system must be rebooted immediately in order to prevent the active virus from reinjecting infection while accessing the disk via INT 13h.

Removal of file viruses is based on a surprising fact, namely, despite the vast number of diverse viruses, there are only a few techniques by which a virus is injected into a file. Here we only briefly outline the file restoration strategy. Figure 1 shows a schematic diagram of a usual EXE file.

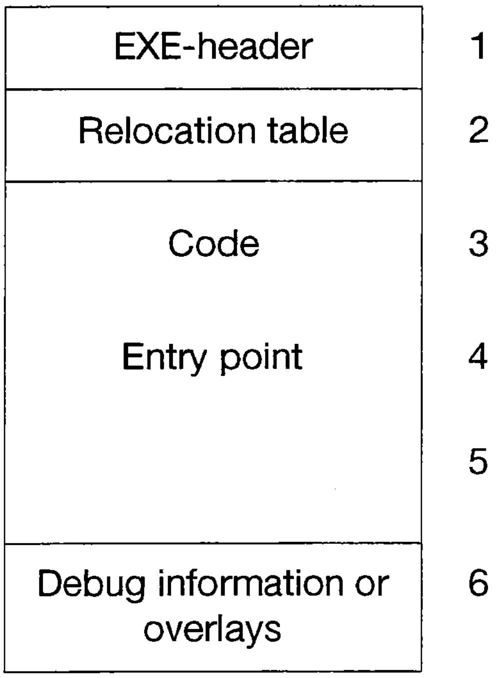


Fig. 1

For each file, the integrity checker keeps a header (area 1), relocation table (area 2) and the code at the entry point (area 4). Strings (area 3 and area 5) are vital because they are the keys to identifying the mutual locations of various areas in an infected file when a virus writes its tail, not at the file end, but at the file beginning or in the file body (after the relocation table or at the entry point). In an infected file, after determining the area that coincides with the imaged areas in the table, the displacement of a block (for example, the block for area 3 begins at the end of area 2 and ends at the beginning of the area 4) can be identified by string 3 position and thus moved back to its original location.

Image of area 6 takes about 3-4 Kb and is essential in recovering a file corrupted by viruses which damage the debug information and overlays in the course of defective infection.

Thus, a file is recovered by reinstating its original status, overwriting the image of its structure stored in integrity checker tables on an infected file. Consequently, a knowledge as to which virus infected the file is not mandatory.

Tables containing information necessary for recovering files take about 200-450 Kb for one logical drive. The table size can be cut down to 90 Kb, if a user decides not to save the relocation information and this has no perceptible influence on the quality of recovery in most cases.

CONCLUSION

Integrity checkers undoubtedly do not provide a panacea against computer viruses. Unfortunately, there is no such panacea, nor can there be one. But they are quite reliable protection utilities which must be used jointly with other classes of anti-virus tools. The highlights of integrity checkers described above are all implemented in ADinf program, the most popular itegrity checker in Russia. It is also known in Germany where it is distributed on CD-ROM as a component of the DialogueScience Anti-Virus Kit. It checks a disk by reading its sectors one by one directly addressing BIOS, easily traps active stealth viruses by comparing the information obtained through BIOS and DOS. It instantly restores up to 97% of files corrupted by known and unknown viruses.

REFERENCES

[1] Vesselin Bontchev, Possible Virus Attacks Against Integrity Programs And How To Prevent Them, *Proc. 2nd Int. Virus Bulletin Conf.*, September 1992, pp. 131-141.

[2] Mostovoy D. Yu., A Method of Detecting and Eradicating Known and Unknown Viruses, *IFIP Transactions*, A-43, Security & Control of Information Technology in Society, February, 1994, pp. 109-111.

EVALUATING DISTRIBUTED VIRUS PROTECTION PRODUCTS

Scott Gordon

McAfee Associates, 2710 Walsh Avenue, Suite 200, Santa Clara, CA 95051, USA

Tel 001 408 988 3832 · Fax 001 408 970 9727 · Email 77321.2764@compuserve.com

Preserving the integrity, confidentiality and accessibility of information resources continues to present new challenges to the data security officer, IS department, network administrator and end user. In the midst of computer 'upsizing, downsizing and rightsizing', the risk of corporate information destruction, modification and disclosure is increasing. The classic centralized model of passwords and physical guards are difficult to apply within a vastly distributed environment of growing, moving and changing workstation and server technologies. The computer industry is well aware that one of the most serious and dangerous security threats to the burgeoning microcomputer and network environment is the virus. Computer viruses are hazardous entities which require continuous industry vigilance in terms of detection, containment and resolution. Although, with all the publicity generated by computer viruses, one might assume that the marketplace is well equipped and can dismiss the thoughts of epidemic contamination. This is not the case.

A 1994 *National Computer Security Association* (NCSA) virus impact survey showed little statistical change since its original Dataquest research of 1992. Among U.S. corporate respondents, over 50% still do not adequately employ anti-virus solutions. Only 38% of corporate users consistently apply workstation anti-virus products. This contributes to the fact that more than 40% of all networks have viruses! A majority of infections are introduced innocently by employees. Clean up can be very costly, and reinfection occurs within an average period of 30 days. In 1994, viruses cost American business approximately $2.7 billion. Researchers are discovering new viruses at a rate of more than two dozen viruses/strains a month; thus, the threat to data security will be worse before it gets better. As present, there is substantial growth of polymorphic virus incidents. Would-be-writer can obtain, and learn to create, generic viruses from 'how-to' books, virus kits, the internet, bulletin boards and even CD-ROM. Since the threat of accidental and intentional virus damage to the corporate environment, including off-site date, is strong, there are several competing issues which must be addressed. This paper will explore the criteria on which customers may base their choice of anti-virus protection.

THE VIRUS WORLD

A computer virus is a program which replicates itself, attaches itself to other programs, and performs unsolicited, it not malicious, actions. Computer viruses are predominantly written for IBM-compatible and Macintosh operating systems. The two fundamental virus categories are 'boot' and 'file' viruses such as 'Form' and 'Yankee Doodle'. Boot viruses, the most reported type, are programs which become active on system start-up. They dwell within the boot sector of a system's infected floppy or hard disk. Most commonly, the boot virus spreads as it becomes memory resident, replicates and attaches onto other available logical disks. File viruses are programs which become active only when executed - these include .EXE, .COM, .DLL and other executable files. The file virus spreads on execution as it typically becomes memory resident, replicates and attaches to other executable programs. Multi-partites are a type of virus which have both file and boot virus characteristics. A virus may monitor for a trigger event, a computer condition which causes a payload to be delivered. A payload may vary from an 'amusing' message, disruption of computer processes or data destruction, to the most lethal type, inconspicuous activity and minute data damage spread cross long periods of time.

Worldwide incident reports indicate that the majority of viruses are introduced innocently to the corporate environment from unsuspecting employees bringing viruses from their home computer or elsewhere outside the office, or through electronic distribution such as bulletin board systems. Virus authors themselves range greatly in terms of age, background and intent. Also, their ability to design and test their programs gives evidence to the diversity of virus performance and propagation. Many viruses require ideal conditions for proper execution, and indeed, many fail to operate, or never execute at all. Certain viruses are more prevalent in one part of the world than in others. Virus researchers judge the most common prevailing viruses as those which consistently and successfully execute and replicate under universal conditional and/or are substantially observed in the computer community. The two hundred most common viruses (which range between one half and three years) account for the majority of infections. Protecting your system from these common viruses may offer sufficient protection, because the likelihood of infection by another virus is quite slim. Still, while there is no standard naming convention, industry researchers have defined more than three thousand 'documented' viruses.

VIRUS COUNTERACTION OPTIONS

Many virus symptoms are not easily distinguishable until overt damage has been done. Often, the only early indication may be that a computer is running slowly or that a program doesn't execute correctly. Currently, there are four techniques to detect the computer virus: integrity checking, memory detection, interrupt monitoring and signature scanning. The integrity checking method determines if a program's file size has increased due to virus attached-code. The memory detection method recognizes the location and code of a given documented virus while in memory. The interrupt monitoring method observes all program system calls (i.e. DOS and Macintosh) in the attempt to stop a sequence of calls which may indicate virus activity. The signature scanning method relies on identifying a unique set of hexadecimal code, the virus signature, which a virus leaves within an infected file. Al anti-virus products use variations of some or all of these virus counteraction techniques.

Two dominant virus classes, stealth and polymorphic, offer additional demands on the anti-virus community. A stealth virus, both passive and active, cloaks its actions, thus making it difficult to detect. A passive stealth virus might maintain a program's preinfected file size in order to hide the infected program's actual increased file size. Therefore, it would evade most integrity checkers. An active stealth virus might target and eliminate the detection functionality of a commercial anti-virus product. The most difficult type of virus to detect is a polymorphic virus. It contains a 'mutation engine' which produces a virus that randomly changes its signature on certain replications. To detect a polymorphic virus, a signature scanning engine must apply a set of scanning rules. If a product utilizes the signature scanning technique (and all leading vendors do), it must actively update its virus signature file (an encrypted file which contains known virus signatures used by the scanner) as well as maintain the scanning engine itself (whose rules must be refined). Depending on the implementation, all of the above-mentioned counteraction methods can be employed on both the workstation or server, and either in real-time or not.

Virus counteraction technology is not without deficiencies, and effectiveness does vary by their application. Integrity checking involves applying a calculation algorithm to a file in order to produce a checksum value. The end user registers a file, and on later execution, the anti-virus system automatically tries to determine if the file has been altered. The down-side to this technology includes: registering infected files, missing stealth viruses, flagging self-modifying executables, and managing the checksum database. Memory detection technology may impose resource requirements and obstruct PC operations. interrupt monitoring, which finds the virus after execution, often flags valid system calls, and has had limited success against all virus types. Signature scanning is only as good as how recent the signature file is updated; albeit, most signature files contain all common virus signatures. Sometimes the signature scanner may identify a signature within a valid file (false positive detection). Depending on how detailed the rule set of a heuristics-based scanning engine, polymorphic iteration might also avoid detection. The combined use of all these technologies suggests a more thorough method of virus obstruction. As the number of viruses continues to increase, virus counteraction must also mature.

EVALUATION CRITERIA FOR THE DISTRIBUTED ENVIRONMENT

To choose from among the leading anti-virus vendor, the best anti-virus data protection solution for a specific environment, a set of pertinent evaluation criteria must first be identified. Many assumptions, which at first or even second glance seem crucial factors, prove, with further exploration, to be misleading. We will first discuss these most obvious criteria and then suggest an optimal list of determining factors.

Is the product which claims to catch the most number of viruses the best? Not necessarily. Since the number of viruses increases each month, each vendor's documented virus detection rate will change monthly. In addition, this detection rate relies on the availability of the product's updates; one month is the de facto standard for signature files. A typical experiment sends a library of viruses through each product's countermeasures and either counts the number of detections or declares a detection percentage. Obtaining appropriate viruses for a library can be complicated, since vendors cannot ethically aid in the distribution of viruses. The results of this test will vary significantly due to: the depth of the virus library (how many are needed?) and how many of what type of virus (i.e. polymorphic), the purity of the library (only viruses?), the naming conventions utilized (uniform?), the most current update (when was the test?), and whether the tester maximizes or normalizes each product's counter-measures. As previously discussed, the two hundred most common viruses account for the vast majority of infections. Therefore, on further investigation, it is apparent that it may be of little consequence that one product detects only a few more viruses than another.

Is the product which claims to perform the fastest and with the least resource utilization is the best? Not necessarily. Evaluators must average multiple test runs for both workstations and servers in order to produce a standard set of values. Being that servers, by nature, perform multi-tasking, one should not assume a product's server memory requirements or server thread usage to affectperformance. Typical experiments will send a library of files and viruses through a given server. The experiment is repeated for each product with countermeasures and features activated. Comparisons can then measure the time it took to scan a number of files and/or detect a number a viruses. The results of this test will vary significantly based on: the size and types of files (how many are necessary?), the testing platform (server and workstation configurations), the most current update (when was the test?) and whether the test maximizes or normalizes each product's countermeasures and/or features. The next question is what detection rate was found at which performance level. Acknowledging that most tests are implemented under controlled conditions, how can one have more confidence in a product which performed merely a few 'seconds' faster, or with slightly less resource utilization?

Is the product which claims to offer the most features the best? Not necessarily. Evaluators must strive to explore each product's feature claims in terms of implementation. All features are not created equal, nor is their implementation suited for all needs. Typical experiments will first establish a standard feature set, such as Novell and NCSA certification. Each product's feature set will be tabulated and some unique features will be discovered. Comparisons can then be made to those products which offer the most features. The results of this test will vary significantly due to the perceived need of the feature (is it just a 'nice-to-have' or is it needed?), the depth of analyses (did empirical results or assumptions qualify each feature?), the most current update (when was the test?), whether the tester maximizes or normalizes each feature set, and how these features are to be implemented (will it be beneficial?). From this discussion it is obvious that one cannot choose a product merely because it offers a greater number of features.

Certainly, the above limitations notwithstanding, an evaluator must test a product's detection rate, performance and features. In addition, products designed for today's vastly distributed environments must safeguard network servers, connected workstations, remote users and stored data. Although the network server itself cannot support virus replication, it can substantially contribute to the spread of viruses to other servers or workstations. While running under a network operating system such as Novell or Windows NT, virus system calls are unsuccessful. Moreover, the server is a vital organ of the network whose availability and manageability remains crucial to efficient operations. Workstations remain the breeding ground and entry point for the vast majority of viruses. If a workstation becomes inoperable, or data is diminished, the user is unproductive. The remote user, distant or isolated from the server, is equally at risk as the connected user. In fact, some argue that the remote user is at greater risk, due to the usual difficulty in serviceability, critical need for data availability, and value of summarized or customized corporate information.

Most users do not associate viruses with data storage management. Within a distributed environment, it is not unusual to witness the daily backup of up to a terabyte of network data. A high performance, uninterrupted virus-free backup is critical to the integrity of the network. Therefore, to omit any of these areas from protection is to assume some level of vulnerability. *Overall, the question is which product offers the most useful features while saving the most time and effort.*

The author believes that one must consider how a product is best suited to managing the virus threat within the evaluator's distributed environment in terms of detection, performance, administration, notification and reliability.

- Detection - Does the product detect all 'common' viruses; consistently detect predominant viruses; and employ a variety of configurable counteraction methodologies?
- Performance - Will the product's impact on the environment, both server and workstation, be minimal in the light of user detection requirements?
- Administration - Can the product be centrally managed and maintained by the administrator while remaining transparent to, and convenient to operate for, the end-user?
- Notification - Does the product offer a variety of ways to enable quick response to a virus incident?
- Reliability - Is the product's design stable; can the vendor support the product; is the vendor pursuing ongoing product development; and does the vendor have the facilities and relationships to meet your current and future needs?

CONCLUSION

Virus controls must be practical and plausible within the context of data preservation, confidentiality and accessibility requirements. With the enterprise not wholly protected, the possibility of a virus spreading quickly and wreaking significant damage increases. It is important to have a firm understanding of the threat of virus exposure, current detection technologies, and present and future environmental needs and constraints. Furthermore, user-customizable testing methodology is a necessity. Ultimately, to select the best enterprise-wide anti-virus solution, the evaluator must fully understand the detection, performance, administration, notification and reliability criteria which best suits the environment.

DYNAMIC DETECTION AND CLASSIFICATION OF COMPUTER VIRUSES USING GENERAL BEHAVIOUR PATTERNS

Morton Swimmer

Virus Test Center, University of Hamburg, Odenwaldstr. 9, 20255 Hamburg, Germany
Tel +49 404 910041 · Fax +49 405 471 5226 · Email swimmer@acm.org

Baudouin Le Charlier and Abdelaziz Mounji

F.U.N.D.P., Institut d'Informatique, University of Namur, Belgium
Email ble@info.fundp.ac.be / amo@info.fundp.ac.be

ABSTRACT

The number of files which need processing by virus labs is growing exponentially. Even though only a small proportion of these files will contain a new virus, each file requires examination. The normal method for dealing with files is still brute force manual analysis. A virus expert runs several tests on a given file and delivers a verdict on whether it is virulent or not. If it is a new virus, it will be necessary to detect it. Some tools have been developed to speed up this process, ranging from programs which identify previously-classified files to programs that generate detection data. Some anti-virus products have built-in mechanisms based on heuristics, which enable them to detect unknown viruses. Unfortunately all these tools have limitations.

In this paper, we will demonstrate how an emulator is used to monitor the system activity of a virtual PC, and how the expert system ASAX is used to analyse the stream of data whicg the emulator produces. We use general rules to detect real viruses generically and reliably, and specific rules to extract details of their behaviour. The resulting system is called VIDES: it is a prototype for an automatic analysis system for computer viruses and possibly a prototype anti-virus product for the emerging 32 bit PC operating systems.

1 INTRODUCTION

Virus researchers must cope with many thousands of suspected files each month, but the problem is not so much the number of new viruses (which number perhaps a few hundred and grows at a nearly exponential rate) as the number of files the researcher receives and must analyse - the glut. Out of perhaps one hundred files, only one may actually contain a new virus. Unfortunately, there are no short cuts. Every file has to be processed.

The standard method of sorting out such files is still brute force manual analysis, requiring specialists. Some tools have been developed to help cope with the problem, ranging from programs which identify and remove previously-classified files and viruses to utilities which extract strings from infected files that aid in identifying the viruses. However, none of the solutions are satisfactory. Clearly, more advanced tools are needed.

In this paper, the concept of dynamic analysis as applied to viruses is discussed. This is based on an idea called VIDES (*Virus Intrusion Detection Expert System*), coined at the Virus Test Center [BFHS91]. The system will comprise of a PC emulation and an IDES-like expert system. It should be capable of detecting viral behaviour using a set of *a priori* rules, as shown in the preliminary work done with Dr. Fischer-Hübner. Furthermore, advanced rules will help in classifying the detected virus.

The present version of VIDES is only of interest to virus researchers; it is not designed to be a practical system for the end-user - its demands on processing power and hardware platform are too high. However, it can be used to identify unknown viruses rapidly and provide detection and classification information to the researcher. It also serves as a prototype for the future application of intrusion detection technology in detecting malicious software under future operating systems, such as OS/2, MS-Windows NT and 95, Linux, Solaris, etc.

The rest of the paper is organized as follows: Section 2 presents the current state of the art in anti-virus technology; Section 3 describes a generic virus detection rule; Section 4 discusses the architecture of the PC auditing system; Section 5 shows how the expert system ASAX is used to analyse the activity data collected by the PC emulator; and finally, Section 6 contains some concluding remarks.

2 CURRENT STATE OF THE ART

For the purpose of discussion it will be necessary to define the term computer virus.

2.1 TERMS

There is still no universally-agreed definition for a computer virus. What is missing is a description which is still general enough to account for all possible implementations of computer viruses. An attempt was made in [Swi95], which is the result of many years of experience with viruses in the Virus Test Center. The following definition for a computer virus is the result of discussion in comp.virus (Virus-L) derived from [Seb]:

Def 1 *A* Computer Virus *is a routine or a program that can 'infect' other programs by modifying them or their environment such that a call to an infected program implies a call to a possibly evolved, functionally similar, copy of the virus.*

A more formal, but less useful, definition of a computer virus can be found in [Coh85]. Using the formal definition, it was possible to prove the virus property undecidable.

We talk of the infected file as the *host program*. System viruses infect system programs, such as the boot or Master Boot Sector, whereas file viruses infect executable files such as EXE or COM files. For an in-depth discussion of the properties of viruses, please refer to literature such as: [Hru92], [SK94], [Coh94] or [Fer92].

Today, anti-virus technology can be divided into two approaches: the *virus specific* and the *generic* approach. In principle, the former requires knowledge of the viruses before they can be detected. Due to advances in technology, this prerequisite is no longer entirely valid in many of the modern anti-virus products. This type of technology is known to us as a *scanner*. The latter attempts to detect a virus by observing attributes characteristic of all viruses. For instance, integrity checkers detect viruses by checking for modifications in executable files; a characteristic of many (although not all) viruses.

2.2 VIRUS SPECIFIC DETECTION

Virus specific detection is by far the most popular type of virus protection used on PCs. Information from the virus analysis is used in the so-called scanner to detect it. Usually, a scanner uses a database of virus identification information which enable it to detect all viruses previously analysed.

The term *scanner* has become increasingly incorrect terminology. The term comes from *lexical scanner*, i.e. a pattern matching tool. Traditionally scanners have been just that. The information extracted from viruses were strings which were representative of that particular virus. This means that the string has to:

- differ significantly from all other viruses, and
- differ significantly from strings found in *bona fide* anti-virus programs.

Finding such strings was the entire art of anti-virus program writing until polymorphic viruses appeared on the scene.

Encrypted viruses were the first minor challenge to string searching methods. The body of the virus was encrypted in the host file, and could not be sought, due to its variable nature. However, the body was prepended by a decryptor-loader which must be in plain text (unencrypted code); otherwise it would not be executable. This decryptor can still be detected using strings, even if it becomes difficult to differentiate between viruses.

Polymorphic viruses are the obvious next step in avoiding detection. Here, the decryptor is implemented in a variable manner, so that pattern matching becomes impossible or very difficult. Early polymorphic viruses were identified using a set of patterns (strings with variable elements). Moreover, simple virus detection techniques are made unreliable by the appearance of the so-called *Mutation Engines* such as MtE and TPE (Trident Polymorphic Engine). These are object library modules generating variable implementations of the virus decryptor. They can easily be linked with viruses to produce highly polymorphic infectors. Scanning techniques are further complicated by the fact that the resulting viruses do not have any scan strings in common even if their structure remains constant. When polymorphic technology improved, statistical analysis of the opcodes was used.

Recently, the best of the scanners have shifted course from merely detecting viruses to attempting to identify the virus. This is often done with added strings, perhaps position dependent, or checksums, over the invariant part of the virus. To support this, many anti-virus products have implemented machine-code emulators so that the virus' own decryptor can be used to decrypt the virus. Using these enhancements, the positive identification of even polymorphic viruses poses no problem.

The next shift many scanners are presently experiencing is away from known virus only detection to detection of unknown viruses. The method of choice is *heuristics*. Heuristics are built into an anti-virus product in an attempt to deduce whether a file is infected or not. This is most often done by looking for a pattern of certain code fragments that occur most often in viruses and hopefully not in *bona fide* programs.

Heuristics analysis suffers from a moderate to high false-positive rate. Of course, a manufacturer of a heuristic scanner will improve the heuristics both to avoid false positives and still find all new viruses, but both cannot be achieved completely. Usually, a heuristic scanner will contain a 'traditional' pattern-matching component, so that viruses can be identified by name.

2.3 GENERIC VIRUS DETECTION

Computer viruses must replicate to be viruses. This means that a virus must be observable by its mechanism of replication.

Unfortunately, it is not as easy to observe the replication as it may seem. DOS, in it various flavours, provides no process isolation, or even protection of the operating system from programs. This means that any monitoring program can be circumvented by a virus which has been programmed to do so. There used to be many anti-virus programs which would try to monitor system activity for viruses, but were not proof against all viruses. This problem led to the demise of many such programs. Later in the paper, we shall discuss how we avoided the problem when implementing VIDES.

A more common approach is to detect symptoms of the infection such as file modifications. This type of program is usually called an *integrity checker* or *checksummer.*

When programs are installed on the PC, checksums are calculated over the entire file, or over portions of the file. These checksums are then used to verify that the programs have not been modified. The shortcoming of this method is that the integrity checker can detect a modification in the file, but cannot determine whether the modification is due to a virus or not. A legitimate modification to, for instance, the data area of a program will cause the same alarm as a virus infection.

Another problem is virus technology aimed specifically against anti-virus products. Advances in stealth and tunnelling technology have made updates necessary. There have also been direct attacks against particular integrity checkers, rendering them useless. Again, the lack of support from the operating system makes the prevention of such attacks very difficult. As a consequence, the acceptance of such products is low.

The non-specific nature of the detection has little appeal for many of the users. Even generic repair facilities in the anti-virus products do not help, despite these methods effectively rendering identification unnecessary. The problem is partly understandable. The user is concerned with his data. Merely disinfecting the programs is not enough if data has been manipulated. Only if the virus has been identified and analyzed can the user determine if his data was threatened.

Generic virus detection technology should not be dismissed. It is just as valid as virus-specific technology. The problems so far have stemmed from the permissiveness of the underlying operating system, DOS, and from the limits in the programs. Both problems can be addressed.

3 DYNAMIC DETECTION RULES

Before we can attempt to detect a virus using ASAX, we need to model the virus attack strategy. This is then translated into RUSSEL, the rule-based language which ASAX uses to identify the virus attack.

3.1 REPRESENTING INFECTION PATTERNS USING STATE TRANSITION DIAGRAMS

State transition diagrams are eminently suitable for representing virus infection scenarios. In this model of representation, we distinguish two basic components: a node in a state transition diagram represents some aspects of the computing system state. Arcs represents actions performed by a program in execution. Given a (current) state s_i, the action a takes the system from the state s_i to the state s_f as shown in Figure 1. The infection process played by a virus can be viewed as a sequence of actions which drives the system from an initial *clean* state to a final *infectious* state, where some files are infected. In order to get a complete description of the actual scenario, a state is adorned by a set of *assertions,* characterizing the objects as affected by actions.

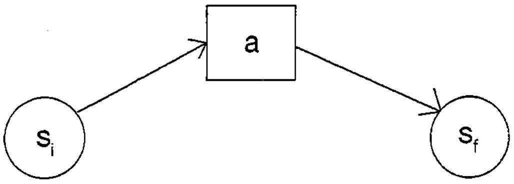


Figure 1: State transition diagram

In practice, we only represent those actions relevant to the infection scenario. As a result, many possible actions may occur between adjacent states, but are not recorded because they do not entail a modification in the current state. In terms of auditing, irrelevant audit records may be present in the sequence of audit records representing the infection signature.

For the sake of simplicity, discussion of the generic detection rules are based on the state transition diagrams described above.

3.2 BUILDING THE RULES

VIDES uses three types of detection rules: *generic detection rules, virus specific rules, other rules.* As its name implies, generic rules are used to detect all viruses which use a known attack pattern. For this, models of virus behaviour are needed for the target system (in our case MS-DOS). Virus-specific rules use information from a previous analysis to detect that specific virus, or direct variants. These rules are similar to virus-specific detection programs, except for the fact that they analyze the dynamic behaviour of the virus instead of its code. Finally, there are the 'other rules' for gleaning other information from the virus which can be used in its classification.

We will not go into the virus-specific rules or the 'other' rules, concentrating instead on the generic rules.

In developing a generic rule for detecting viruses, we need to have a model for the virus attack. No one model will do, because MS-DOS viruses can use choose from many effective strategies. This is compounded by the diversity of executable file types for MS-DOS. Fortunately for us, the majority of viruses have chosen one particular strategy, and infect only two types of executable files. This means that we can detect most viruses with very few rules. On the other hand, a virus which uses an unknown attack strategy will not be detected. For this reason, the prototype analysis system contains an auxiliary static analysis component to detect such problems.

In the following, we will develop a generic rule which detects file infectors that modify the file directly to gain control over that file. We will concentrate on COM file infectors. EXE file infectors are detected in an analogous way.

We must make two assumptions about the behaviour of DOS viruses to help us build the rule.

Assumption 1: *A file-infecting virus modifies the host file in such a way that it gains control over the host file when the host file is run.*

This is a specific version of the virus definition (Def 1). However, it doesn't specify when the virus gains control over the host file.

Assumption 2: *The virus in an infected file receives control over the file before the original host program.*

That is, when the infected file is run, the virus is run before the host program.

Discussion: If the virus never gains control over the host file, it would not fulfil the definition of a virus. This observation leads to Assumption 1. However, there is no reason (in the definition) why the virus must gain control before the host does.

We make an additional assumption that the virus *does* gain control before the host program does. The reason we do this is to avoid very blatant false positives. However, it should be noted that Assumption 2 does not result from the virus definition, and will cause some viruses to be missed. For these cases, other rules are used.

3.3 FINDING COM FILE INFECTIONS

With respect to assumptions 1 and 2, we are looking for two possible infection strategies:

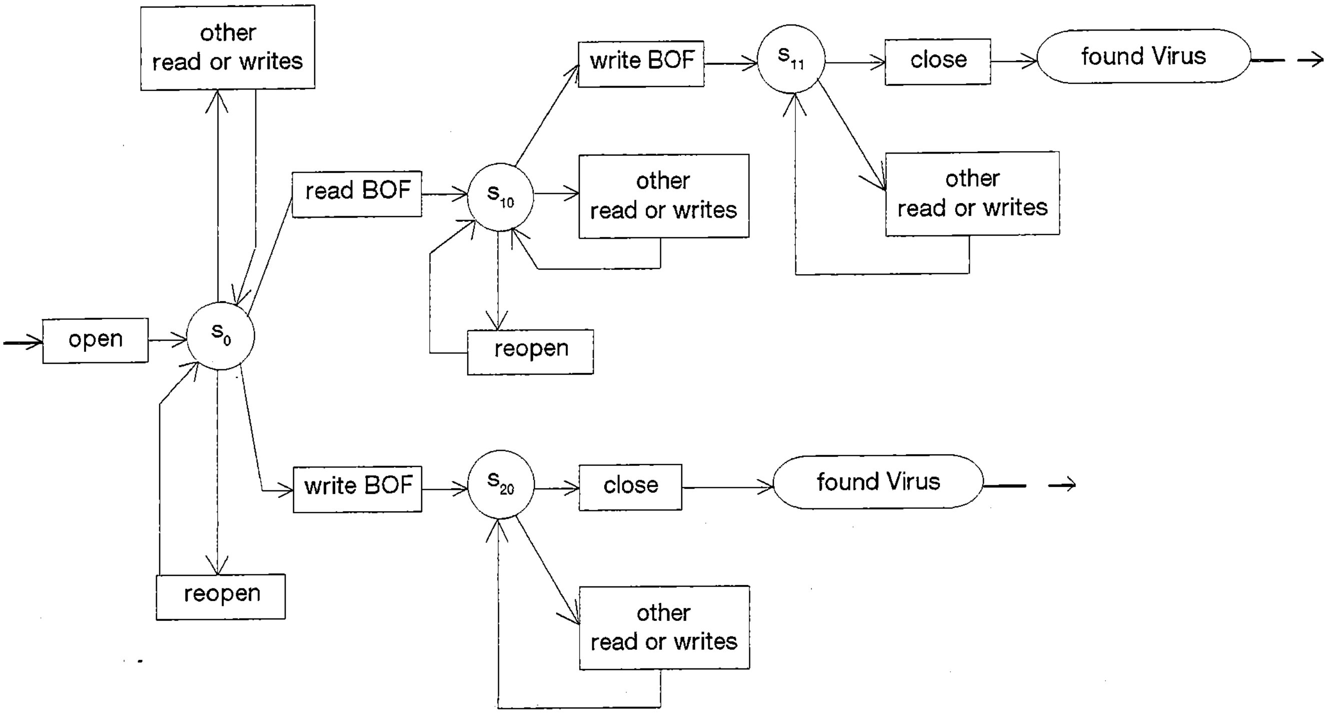


Figure 2: Generic rule for identifying COM file infectors

1 The virus is *overwriting*. Therefore, we are looking for a write to the beginning of the file (BOF), without a previous read to the same location. Other reads and writes are permitted.

2 The virus is *non-overwriting*. We expect to see a read to BOF, then a write to BOF. Before, in between, and after these two events, other reads and writes are permitted.

The assumption in both cases is that the write to BOF causes the virus to gain control on execution.

In the case of a non-overwriting virus, we assume that the virus first reads the original code at BOF and then replaces it with its own code, usually a *jump* to the virus body. In most cases, the number of bytes read will be the same as the number of bytes written, but we cannot assume this. In the case of an overwriting virus, the code is not read (and saved somewhere), but overwritten.

Other reads and writes are not actually relevant to the detection of the virus. They can be logged and used in generating virus- specific rules.

The rule is initiated by the opening of a file (in this case a COM file). The rule is terminated by a close of the file, where this does not have to be done by the virus itself. In between these two events, we expect the actual infection to occur. We look for the *read BOF* followed by the *write BOF* or the *write BOF* without the read. Other administrative operations, like tracking the file position, are also done by the rule. This is shown in the state transition diagram of Figure 2.

Some viruses cause problems for the rule by closing the file after a first set of operations. This is handled by a *reopen* mechanism which waits for a possible open event on the same file from the virus. In order that this rule does not stay active indefinitely and clog up the rule memory, there are a number of terminating events. In fig. 2, *reopen* is abstracted as a transition element, whereas its implementation is as a separate rule.

MS-DOS provides two methods of accessing files. The most common method uses file handles. Access using *file control blocks* (FCB) was provided for compatibility to CP/M, and is rarely used, even by viruses. However, because it is used, we need a separate rule to handle this method. The basic rule stays the same, but internal handling of the data is different.

We could avoid this problem by abstracting the audit data to give us a generic view of the system events. This way, we could reduce the number of audit records to only relevant higher-level records by using a filter. After that, processing becomes simpler as the problems of reopens and handle/FCB use disappear. This method also allows us to apply the rules on non-MS-DOS systems which provide similar file handling.

As a matter of fact, ASAX itself is the logical choice to act as the filter. The first ASAX system reads the raw audit trail, converts it into generic data, and pipes its output as a NADF file for further processing (see Section 5). Using ASAX as a filter allows us to reduce the complexity of maintaining such a system while not sacrificing any power.

4 PC AUDITING

The prerequisite for using an Intrusion Detection (ID) system like ASAX is an audit system which securely collects system activity data. In addition, integrity of the ID system itself must not be compromised: this means that the audit data retrieval, analysis and archiving must be secured against corruption by viruses. Moreover, the ID system must not be prevented from reporting (raising alarms, updating virus information databases) the results of such analysis. DOS neither provides such a service, nor makes the implementation of such a service easy. Its total lack of security mechanisms means that the collection of data can be subverted. Even if the collection can be secured, the data is open to manipulation if stored on the same machine.

For the prototype of VIDES, we were not bound to a real world implementation, so we explored various alternative possibilities. The experience gained by the use of such a system will not benefit DOS users, but should be applicable to users of various emerging 32-bit operating systems which offer DOS support.

We have made several attempts to build a satisfactory audit system: these are described hereafter.

4.1 DOS INTERRUPTS

All DOS services are provided to application programs via interrupts, which can be described as indexed inter-segment calls. Primarily, interrupt 0x21 is used. The requested service is entered into the AH register and its parameters are entered into the other registers. When the service is finished, it returns control to the calling program and provides its results in registers or in buffers.

The very first implementation of an auditing system was a filter which was placed before DOS Services and registered all calls to DOS functions. This was done very early on, together with Dr. Fischer-Hübner, to prove the feasibility of the VIDES concept. It also demonstrated the limits which DOS imposes on the implementation of such an auditing system: it did not run reliably, and could be subverted by tunnelling viruses.

This implementation was soon scrapped, but it did prove that the premise was correct: viruses could be found using ID technology. This was perhaps the first such a trial that had been done [BFHS91].

4.2 VIRTUAL 8086 MACHINE

The Intel iAPX 386 introduced the so-called virtual 8086 machine mode. A protected mode operating system can create many virtual 8086 machines in which tasks can run completely isolated from each other and from the operating system. Each task 'sees' only its own environment. Operating systems such OS/2 use these constructs to provide a full DOS environment for DOS programs. All calls to the machine (via the BIOS interface or direct port access) and DOS are redirected to the host operating system (OS/2 in this case) for processing.

This mechanism can also be used to monitor the activity in DOS session. Because all interrupts are being redirected to the native operating system, the native operating system can record the activity securely and unobtrusively.

Care has to be taken in the implementation of the virtual 8086 machine. The DOS windows in OS/2 have been shown in tests at the VTC to be too permissive. In the course of a comprehensive test including the entire collection of file viruses, many of the viruses running under a DOS window managed to harm vital parts of the system. One problem was that OS/2 files could be manipulated directly from within the DOS session. However, this did not explain the corruption of the running operating system.

Even though using a virtual 8086 machine was the original method of choice, such experiments showed that the complexity of building a safe implementation would be difficult. A more secure method was sought for the prototype.

4.3 HARDWARE SUPPORT

Hardware debugging systems, such as the *Periscope IV*, may be used to monitor system events closely in real time. This is achieved by a card fitted between the CPU and the motherboard and which can set break points on various types of events on the PC's bus. The card is connected to a receiving card in a second PC which is used to control the debugging session.

Monitoring system behaviour on a DOS machine can be accomplished by capturing the Interrupt 0x21 directly, or by setting a break point in the resident DOS kernel. Special memory areas can be monitored by setting a break condition on access to those areas.

The monitoring is completely unobtrusive, i.e. the program will not notice a difference between running with or without the debugger. When an event is triggered, the PC is stopped while the controlling PC is processing the data. If the controlling PC is fast enough, the time delay should be nearly negligible.

A hardware solution using the *Periscope IV* is complicated by the problem of automating the processes necessary to test large numbers of viruses on different operating systems. When such a solution is implemented, it will offer the possibility of testing viruses on other PC operating systems which require full iAPX 386 compatibility.

4.4 8086 EMULATION

The solution which was finally chosen was the software emulation of the 8086 processor. An emulation is a program which accepts the entire instruction set of a processor as input, and interprets the binary code as the original processor would. All other elements of the machine must be implemented or emulated, e.g. the various ports. To simplify and quicken the emulation, the BIOS Code (Basic Input Output System - the interface between the operating system and the hardware) can be replaced with special emulation hooks, so that the complicated machine access can be skipped as long as all access to those services are routed via the BIOS. In the case of a graphics adapter, the entire hardware must be emulated, whereas disk access can be handled with hooks in the BIOS.

Using emulation gives us all the advantages of the hardware solution plus the possibility of handling everything in pseudo real-time with respect to the program running in the emulation. Because even the time-giving functions of the emulation are being steered by the emulation, when interruptable to process an event, the time in the emulation can also be stopped.

The emulation is 'safe' as the running virus has no access to the host machine at all. This is because the target machine's memory is being controlled entirely by the emulation, and file accesses are directed to a virtual disk, stored as a disk image file.

The major problem with using an emulation is its lack of speed. Even on fast platforms, the running speed is only marginally faster than an original *PC/XT*.

4.5 ACTIVITY DATA FORMAT

Audit records representing the program behaviour in general, and virus activity in particular, have a pattern which is borrowed from the Dorothy Denning's model of Intrusion Detection [Den87] (*<Subject, Action, Object, Exception-Condition, Resource-Usage, Time-Stamp>*). However, due to the way processes are handled in DOS, this pattern is slightly modified to collect useful available attributes. For instance, the code segment of a process is chosen instead of the common process identifier in most existing multi-user operating systems.

The audit record attributes of records as collected by the PC emulator have the following meaning: *code segment* is the address in memory of the executable image of the program; *function number* is the number of the DOS function requested by the program; *arg (...)* is a list of register/memory values used in the call to a DOS function; *ret (...)* is a list of register/memory values as returned by the function call; *RecType* is the type of the record; *StartTime* and *EndTime* are the time stamp of action start and end respectively. The final format for an MS-DOS audit record is as follows: *<code segment, RecType, StartTime, EndTime, function number, arg (...), ret (...)>*. An example of an audit trail is given in fig. 3.

```
                                        :
<CS=3911 Type=0 Fn=30 arg() ret( AX=5)>
<CS=3911 Type=0 Fn=29 arg() ret( BX=128 ES=3911)>
<CS=3911 Type=0 Fn=64 arg( AL=61 CL=3 str1=*.COM) ret( AL=0 CF=0)>
<CS=3911 Type=0 Fn=51 arg( AL=0 str1=COMMAND.COM) ret( AL=0 CX=32 CF=0)>
<CS=3911 Type=0 Fn=51 arg( AL=1 str1=COMMAND.COM) ret( AL=0 CX=32 CF=0)>
<CS=3911 Type=0 Fn=45 arg( AL=2 CL=32 str1=COMMAND.COM) ret( AL=0 AX=5 CF=0)>
<CS=3911 Type=0 Fn=73 arg( BX=5) ret( CX=10241 DX=6206 CF=0)>
<CS=3911 Type=0 Fn=27 arg() ret( CX=5121 DX=8032)>
<CS=3911 Type=0 Fn=47 arg( BX=5 CX=3 DX=828 DS=3911) ret( AX=3 CF=0)>
<CS=3911 Type=0 Fn=50 arg( AL=2 BX=5 CX=0 DX=0) ret( AL=0 AX=50031 DX= CF=0)>
<CS=3911 Type=0 Fn=48 arg( BX=5 CX=648 DX=313 DS=3911) ret( AX=648 CF=0)>
<CS=3911 Type=0 Fn=50 arg( AL=0 BX=5 CX=0 DX=0) ret( AL=0 AX=0 DX=0 CF=0)>
<CS=3911 Type=0 Fn=48 arg( BX=5 CX=3 DX=831 DS=3911) ret( AX=3 CF=0)>
<CS=3911 Type=0 Fn=74 arg( BX=5 CX=10271 DX=6206) ret( CF=0)>
<CS=3911 Type=0 Fn=46 arg( BX=5) ret( CF=0)>
<CS=3911 Type=0 Fn=51 arg( AL=1 str1=COMMAND.COM) ret( AL=0 CX=32 CF=0)>
                                        :
```

Figure 3: Excerpt from an audit trail for the Vienna virus

4.6 ACTIVITY DATA COLLECTION

The audit system was integrated into an existing PC emulation by placing hooks into the module for processing all opcodes corresponding with the events (see fig. 4). These are primarily calls to the DOS functions. This was implemented in such a way, that stealth and tunnelling viruses could not circumvent the mechanism. A separate module receives notification of the event and pushes all parameters on to a stack. When the DOS call returns, the parameters are popped from the stack and sent to the audit trail with the return values.

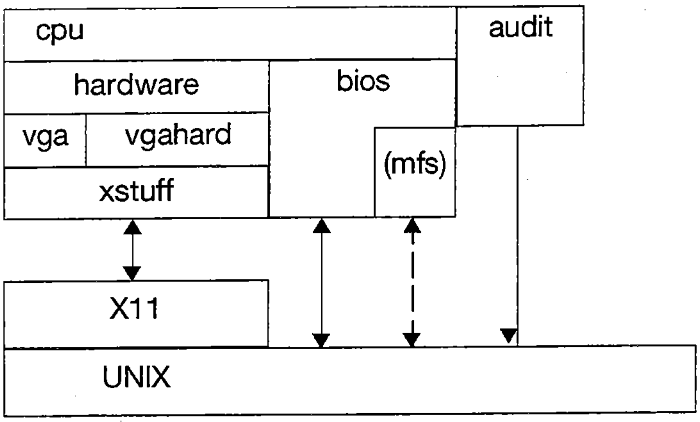


Figure 4: Modules in Pandora

Internally, the audit trail complies to a canonical format, which is also ASAX's native format. This is very generic, and allows most types of records to be implemented.

An example of an audit trail is printed in Figure 3. This is a human readable representation of the binary NADF file.·The example is from an audit trail of the Vienna virus. The text representation does not comply exactly with the binary version. Some of the less important fields are missing so that the audit record becomes clearer and shorter.

In the next section, we show how the activity data produced by the emulator is analysed using ASAX.

4.7 USING RUSSEL TO DETECT INFECTION SCENARIOS

In this section, we show how the RUSSEL language can be used effectively to detect an infection scenario. We first model the infection as a state transition diagram, then briefly show how this diagram can be translated into RUSSEL rules.

Each state in the diagram is represented by a rule describing not only the current state, but also the sequence of previous states leading to it. The actual parameters of the current rule encode all the relevant information collected in previously-visited states. A transition in the diagram is represented by the rule-triggering mechanism of the RUSSEL language as described in section 5. The actual parameters of the current rule are computed from the data items conveyed by the current audit record and from the parameters of the current rule. Once triggered, the new rule represents the new current state in the transition diagram.

In particular, the very first active rule at the beginning of the detection process has no actual parameters, since no information is contained in the initial state (one can argue that the initial state contains this assertion: system is clean. That is then represented by an empty list of parameters). As an example, the states s_0, s_1 and s_2 of fig. 5. are represented by the rules Open, readBOF(...), and 1seekEOF(...) respectively. Figure 4.7 depicts this set of rules in the RUSSEL language. In this figure, RUSSEL keywords are noted in bold-face characters, words in italic style identify fields in the current audit record, and actual parameters are noted in roman-style words.

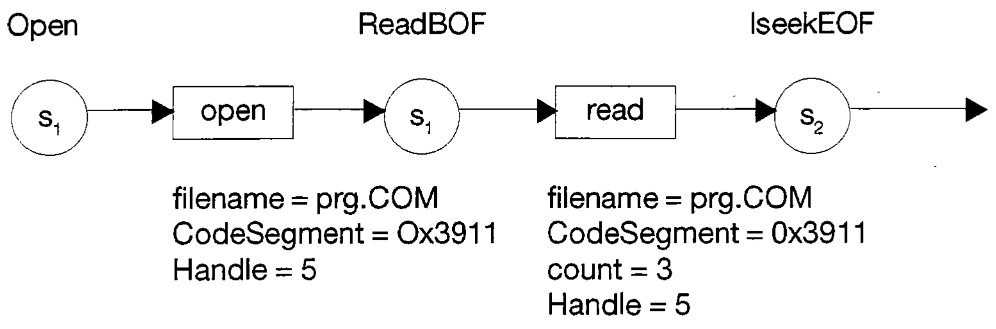


Figure 5: Example of a state transition diagram

Finally, the transition leading to the final state does not trigger further rules, but instead initiates a procedure which raises an alarm message describing the infection and using the data items accumulated along the path.

5 ASAX FEATURES

This section outlines the main features of the ASAX (*Advanced Security audit trail Analysis on uniX*) tool. It is used in VIDES as an expert system for intelligent analysis of virus activity data collected by the PC emulator. For a more detailed description of ASAX's architecture, the reader is referred to [HLCMM92a]. A comprehensive description of ASAX and its implementation is presented in [HLCMM92b, HLCM92].

```
rule Open;
if
                strToInt (Function) = 45 /*open file */
        and match('.COM$',arg_str1 = 1
        ->
                    trigger off for_next
                        readBOF(ret_AX, CS, arg_str1)
        true
        ->
                trigger off for_next
                        Open
fi;

rule readBOF(handle, codeSeg, filename:string);
if
                strToInt(Function) = 47 /* read file */
        and CS = codeSeg
        and ret_BX = handle
        ->
                    trigger off for_next
                        lseekEOF(handle, codeSeg, arg_CX, filename)
                                                    :
fi;

rule lseekEOF(handle, codeSeg, count, filename:string);
if
                                                    :
```

Figure 6: The FileOpen rule

ASAX has proved very powerful for efficient intrusion detection on UNIX platforms. It uses *a priori* rules for detecting malicious behaviour. Two versions of the ASAX system are currently available. The single audit trail analysis version is applicable only to a single audit trail. The other version allows a distributed analysis of multiple audit trails produced at various machines on a network. In the latter version, ASAX filters audit data at each monitored node and analyses the filtered data gathered at a central host (see [MLCHZ95]). In the following, we describe briefly the main features of ASAX.

5.1 UNIVERSALITY

ASAX is theoretically able to analyse arbitrary sequential files. No semantic restrictions are imposed on the file being analysed. For instance, analysed files could be trace data, generated by a process controller, or audit data, collected in a multi-user environment. In the context of this paper, the sequential file is the activity data record produced by the PC emulator. The universality is attained by translating native files to a generic format which is the only one supported by the evaluator. The format is simple and flexible enough to allow straightforward conversion of most file formats. This generic format is referred to as the *Normalized Audit Data Format* (NADF).

An NADF file is a sequential file of records in NADF format. An NADF record consists of the following:

- a four-byte integer representing the length (in bytes) of the whole NADF record (including the length field);
- a certain number of contiguous audit data fields. Each audit data field contains the three following contiguous items:

identifier:	an unsigned short (16-byte) integer which is the identifier of the audit data. This item must be aligned on a 2-bytes boundaries;
length:	an unsigned short integer which is the length of the audit data value;
value:	the actual audit data value.

In addition, audit data identities appearing in an NADF record must be sorted in a strict ascending order. This is important for ASAX to preprocess audit records efficiently before analysis. A user guide for constructing NADF files is presented is [Mou95].

5.2 POWER: THE RUSSEL LANGUAGE

RUSSEL (RUle-baSed Sequence Evaluation Language) is a novel language, specifically tailored to the problem of searching arbitrary patterns of records in sequential files. The built-in mechanism of rule triggering allows a single pass analysis of the sequential file *from left to right.*

The language provides common control structures such as conditional, repetitive, and compound actions. Primitive actions include assignment, external routine call and rule triggering. A RUSSEL program simply consists of a set of rule declarations which are made of a rule name, a list of formal parameters and local variables, and an action part. RUSSEL also supports modules sharing global variables and exported rule declarations. The operational semantics of RUSSEL can be briefly described as follows:

- Records are analysed sequentially. The analysis of the current record consists in executing all active rules. The execution of an active rule may trigger off new rules, raise alarms, write report messages or alter global variables, etc.
- Rule triggering is a special mechanism by which a rule is made active, either for the current or for the next record. In general, a rule is active for the current record because a prefix of a particular sequence of audit records has been detected (the rest of this sequence may still be found in the rest of the file). Actual parameters in the set of active rules represent knowledge about the already-found subsequence and is useful for selecting further records in the sequence.

- When all the rules active for the current record have been executed, the next record is read and the rules triggered for it in the previous step are executed in turn.
- To initialize the process, a set of so-called *init* rules are made active for the first record.

User-defined and built-in C-routines may be called from a rule body. A simple and clearly-specified interface with the C language allows to extend the RUSSEL language with any desirable feature. This includes simulation of complex data structures, sending an alarm message to the security officer, locking an account in the case of an outright security violation, etc.

5.3 IMPLEMENTATION

Efficiency is a critical requirement for the analysis of large sequential files, especially when on-line monitoring is involved. The very principle of the rule-based language RUSSEL allows each record to be processed only once, whatever complex is the analysis. RUSSEL is efficient, thanks to its operational semantics, which exhibit a bottom-up approach in constructing the searched record patterns. Furthermore, optimization issues are carefully addressed in the implementation of the language: for instance, the internal code generated by the compiler ensures a fast evaluation of Boolean expressions and the current record is pre-processed before evaluation by all the current rules, in order to provide a direct access to its fields.

All reports and conference papers related to the RUSSEL language, as well as the whole ASAX package, are available from the anonymous ftp site ftp.info.fundp.ac.be/pub/projects/asax.

6 CONCLUSION

As with all virus detection systems, it is not possible to state that all future viruses will be detected by the system. However, whereas scanner technology requires previous knowledge of the actual virus, VIDES requires a knowledge of the infection strategy. The number of new viruses averages a few hundred every month, however, the number of new infection strategies which are significantly different from the point of view of the detection rules average less than one a year.

This is demonstrated by the generic detection rule which was developed using some of the first viruses. The rule, when used on a collection of all known viruses, scored 95% of all viruses which ran in the emulation. This indicates that significant departures from the mainstream infection strategy are rare.

The advanced rules generate enough information to give a rough idea as to what type of virus is being analysed. With further development of the audit system, we hope to get more details of the virus. In particular, indications to the virus' damage routine would be important.

VIDES could conceivably be used outside the virus lab to detect viruses in a real environment. One possibility is to use it as a type of firewall for programs entering a protected network. Another possibility is the detection of viruses in the DOS sessions of emerging 32- bit desktop operating systems.

For such a system to be accepted, it must not cause false positives. A concept for this is currently under development. Such a system must also be unnoticeable unless a virus is found. As a virtual 8086 machine will be the basis for this, the only extra overhead will come from the audit system and from ASAX. The audit system can be tuned to provide only the nesessary data, which eliminates some overhead. ASAX has proven itself very fast: only the rules must be tuned for speed.

The prototype VIDES system shows that an automatic analysis system for computer viruses is possible and useful. At the same time, it is a prototype for the use of intrusion detection technology for desktop systems.

REFERENCES

[BFHS91] Klaus Brunnstein, Simone Fischer-Hübner, Morton Swimmer. 'Concepts of an expert system for computer virus detection', *Proceedings of the 7th International IFIP TC-11 Conference on Information Security,* SEC '91, May 1991.

[Coh85] Frederick Cohen. Computer Viruses. PhD thesis, University of Southern California, December 1985.

[Coh94] Dr. Frederick B. Cohen. 'A Short Course on Computer Viruses', John Wiley & Sons, Inc., 1994. ISBN 0-471-00769-2.

[Den87] Dorothy E. Denning. 'An intrusion detection model', *IEEE Transactions on Software Engineering,* 13-2:222, Feb 1987.

[Fer92] David Ferbrache. 'A Pathology of Computer Viruses', Springer Verlag, 1992. ISBN 3-540-19610-2.

[HLCM92] N. Habra, B. Le Charlier, and A. Mounji. 'ASAX: Implementation design of the NADF evaluator', Technical report, Institut d'Informatique, University of Namur, Namur, Belgium, March 1992.

[HLCMM92a] N. Habra, B. Le Charlier, A. Mounji, and I. Mathieu. 'ASAX: Software Architecture and Rule-based language for Universal audit trail analysis', *In Proceedings of the Third European Symposium on Research in Security (ESORICS'92),* Lecture Notes in Computer Science, Toulouse, November 1992. Springer Verlag.

[HLCMM92b] N Habra, B. Le Charlier, A. Mounji, and I Mathieu, 'Preliminary report on ASAX'. Technical report, Institut d'Informatique, University of Namur, Namur, Belgium, January 1992.

[Hru92] Jan Hruska. 'Computer Viruses and Anti-Virus Warfare', Ellis Horwood Ltd, 2nd edition, 1992. ISBN 0-13-036377-4.

[MLCHZ95] A. Mounji, B. Le Charlier, N. Habra, and D. Zampunieris. 'Distributed Audit Trail Analysis', *In Proceedings of the Internet Society Symposium on Network and Distributed System Security (ISOC'95),* San Diego, California, Feb 1995. IEEE.

[Mou95] A. Mounji. 'User Guide for Implementing NADF Adaptors', Technical report, Institut d'Informatique, University of Namur, Namur, Belgium, Jan 1995.

[Seb] Brian Seborg. Upcoming comp.virus FAQ.

[SK94] Alan Solomon and Tim Kay. 'Dr Solomon's PC Anti-Virus Book', Newtech, 1994. ISBN 0-7506-16148.

[Swi95] Morton Swimmer. 'Fortschrittliche Virus-Analyse - Die Benutzung von statischer und dynamischer Programm-Analyse zur Bestimmung von Virus-Charakteristika', Diplomarbeit, University of Hamburg, Germany, Fachbereich Informatik, Arbeitsbereich AGN, 1995.

FLASH BIOS - A NEW SECURITY LOOPHOLE

Jakub Kaminski

Cybec Pty Ltd, PO Box 205, Hampton, VIC 3188, Australia

Tel +61 3 521 0655 · Fax +61 3 521 0727 · Email jakub@tmxmelb.mhs.oz.au

INTRODUCTION

When a PC is first started, control is transferred to the program called the BIOS (Basic Input Output System). Until recently, this has usually been stored in EPROM (Electrically Programmable Read Only Memory) and as this can be altered only by exposure to ultraviolet light after being removed from a computer, it has been effectively tamper proof. However the openness of the IBM compatible systems and fast increase in the number of different peripherals designed to extend the functionality of such computers force the system producers to upgrade the BIOS code. The number of constant changes pointed towards another less expensive and easier to maintain solution.

In many modern PCs, the BIOS is stored in the Flash EEPROM (Electrically Erasable Programmable Read Only Memory). Some applications even include the procedures necessary for updating the BIOS version. As a result BIOS vendors can supply the updates in the form of data files sent on a diskette or, even more simply, accessible by modem and stored on BBS or in the producer's FTP site. Simplicity of updating ROM code has many attractions to the manufacturer, but unfortunately also opens a massive security loophole; if the BIOS contents can be reprogrammed (and this can be done by a malicious user as well as by a virus or a trojan horse) we could face the situation when 'the clean system boot' is no longer achievable.

1 WHAT IS FLASH MEMORY

Flash memory is a non-volatile, high-density, electrically erasable and programmable memory (bulk-erasable, byte-programmable), characterised by low power consumption and extreme ruggedness.

Flash memory provides the same non-volatility and alterability as traditional EPROM (erasable under ultraviolet light and electrically programmable read-only memory) used until recently in almost all personal computers to store the BIOS code, but, additionally, offers the easy and inexpensive way of updating the software version. The contents of Flash memory can be changed using the ISW (in system writing) method, eliminating the need to remove the chip from the board in order to reprogram it. It also means that memory can be fixed to the motherboard eliminating unreliable sockets.

2 FOUNDATION SHAKE

We have got used to taking the presence of the BIOS for granted. It was always there, somewhere inside our PCs, imprinted forever in the read-only memory, hidden between dozens of other chips on the motherboard. Like a reliable part of the brain controlling the fundamental activities of the organism ensuring cooperation between all peripherals, the BIOS is an essential part of a computer. When the BIOS code crashes the whole PC may stay unbootable and useless.

We have got used to the fact that malicious programs (viruses and trojans) can damage our files, overwrite whole disks, even corrupt the CMOS data. We keep backups of our work, we make rescue disks containing copies of the boot sectors and we create copies of the CMOS configuration. It is not only average users, but also people involved in anti-virus research and those concerned with security issues, who do not keep copies of their BIOS code (read from their PCs).

Until now, there was no need for it. And for the vast majority of users there won't be a need in the very near future. At least not until most of the PCs on the market are equipped with Flash BIOS, when trojans and viruses targeting such systems may become a real threat.

3 TRYING SOMETHING NEW

In 1993, in the third edition of 'Upgrading and Repairing PCs', Scott Mueller wrote: 'Without the lock, any program that knows the right instructions can rewrite the ROM in your system - not a comforting thought! Without the write protection, it is conceivable that virus programs could even be written that copy themselves directly into the ROM BIOS code in your system!'

The idea that BIOS code could, theoretically, be altered without even opening the computer case has already attracted the virus writers' attention. The first such virus, completely näive and funny in its ignorance, without any chance of working, was a boot sector virus; its source code was revealed in October 1994.

Coincidentally, a few weeks later, I had a chance to investigate a PC that was showing all the signs of an unknown virus infection. When booted on 13th of November, the PC played the Happy Birthday tune and the system hung. After eliminating the possibility of an infection with a file and later with a boot sector virus I started looking through the BIOS code. I've found the malicious procedures and fortunately discovered that the trojan does nothing but plays the tune and locks up the computer. As it turned out later, the 'Happy Birthday' trojan was written 'in house', and the big party of trojanised motherboards was shipped from one of the South-Asian countries all over the world. This was not a case of malicious software targeting specific BIOS types but it encouraged me to investigate further into the BIOS security problem and the implementation of Flash EEPROM chips in the modern PCs.

4 THE REAL THING

There are many different Flash components, including: Flashfile Memory, Bulk-Erase Memory, Boot Block Memory, Flash Drives, Flash Cards. The manufacturers' Flash Memory catalogues contain hundreds of pages. Currently it seems like the most commonly used as BIOS storage elements are the Boot Block Memory chips with 128 KB capacity in typical desktop implementations (eg PVI-486SP3, GA-586AP). One of the motherboard distributors in Australia, ASUSTek, supplies Flash Memory Writer utilities supporting a few typical flash block components, including: Intel 28F001B, SST 27EE010, Winbond W29EE010.

The Boot Block Memory is a component in which the memory array is divided into blocks that can be erased separately and an additional hardware protection feature is added to protect data in the one selected area (the boot block).

The example of internal memory map of one of the 128 KB Boot Block device is pictured in fig.1.

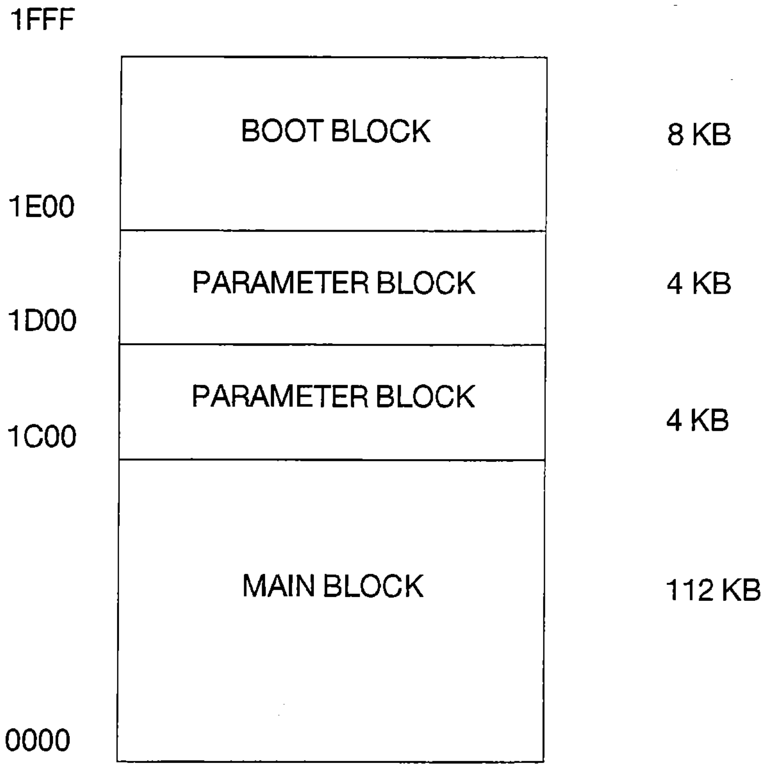


Figure 1. Memory Map of Intel 28F001BX

The four blocks allow logical segmentation of the entire code. The parameter blocks provide convenient storage for software and hardware configuration data backing up or even reducing system SRAM and battery configurations. The boot block can store the recovery procedures in case of system failure while updating the code in the main or parameter blocks.

In order to access and control the Flash memory component in-built in the system structure, it's necessary to provide the proper interface including the required signals and signal levels. The basic requirements of the Microprocessor System - Flash Memory interface is presented in fig.2

5 THE WORKING MODES

The typical Block Flash Memory chip has a few different modes which define specific possible operations. The most important, from the security point of view, are Read and Write Modes. The modes are selected and changed by external memory-control pins like: RP, CE, OE, WE, Address and Data lines. The level of Vpp does not determine the type of current working mode.

5.1 READ MODE

After the initial power up, the Flash memory operates as a standard EPROM. From the users point of view, the process of reading the contents of Flash memory looks exactly the same as reading any of the memory mapped inside the system memory resources (EPROM, Static or Dynamic RAM).

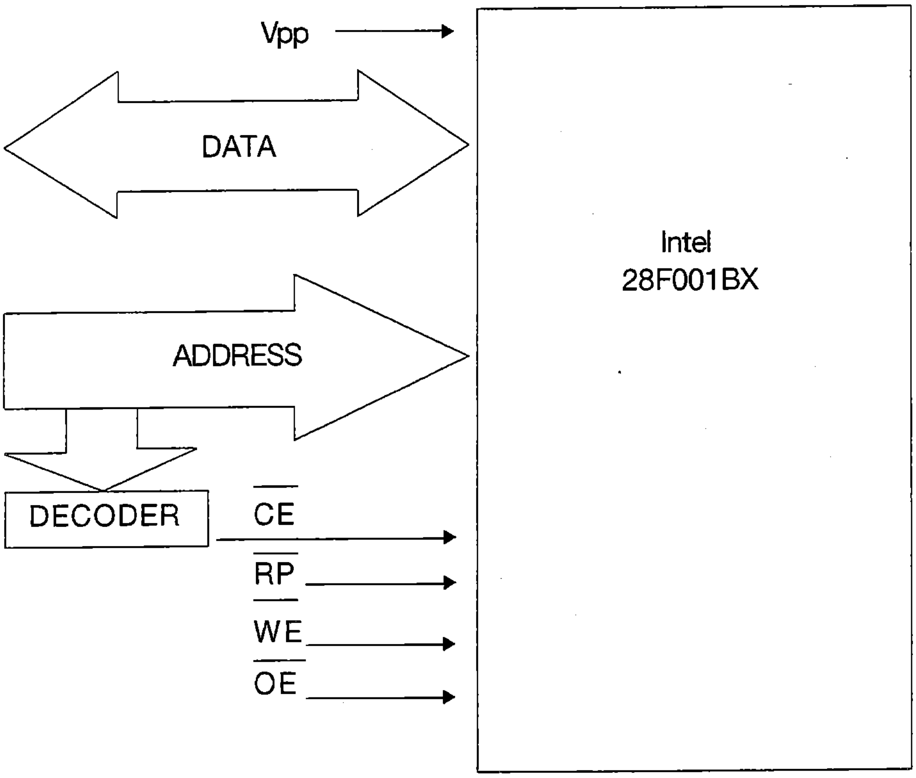


Figure 2. System - Flash Memory Interface

5.2 WRITE MODE

Write Mode together with Read Mode enables access to the Command Register and invokes a special set of operations like reading the Intelligent Identifier or reading and resetting the Status Register. Together with the high (active) level of Vpp, Write Mode enables erase and program operations. The Command Register does not occupy any specific memory location, and it's function is fulfilled by the latch storing commands, addresses and data needed to perform the specific operation.

6 ERASE/PROGRAM PROTECTION

The Flash BIOS memory chip installed in the specific PC is usually quite well protected against accidental or unwanted erasing or reprogramming. The general protection scheme is presented in fig.3. The elements that can be considered as parts of the erase/program protection are: programming voltage (Vpp), hardware environment of the specific implementation, segmentation and additional boot block protection, specific component requirements.

6.1 RAISING PROGRAMMING VOLTAGE (VPP)

Turning the Vpp on/off is sometimes referred to as 'the absolute data protection'. This term perfectly reflects the fact that without providing the active level of Vpp alterations to the Flash memory data is impossible. It also means that whoever controls the Vpp, controls, in fact, any changes to memory contents.

Some of the motherboard producers prefer to implement a hardware switch to enable the erasing and programming of the Flash ROM memory or to protect the chip against unwanted writing. The jumper placed on the motherboard enables/disables the active state of Vpp. In the latest solutions (eg PVI-486SP3), a set of two jumpers not only allows you to turn Vpp on, but also selects its level depending on the type of the Flash

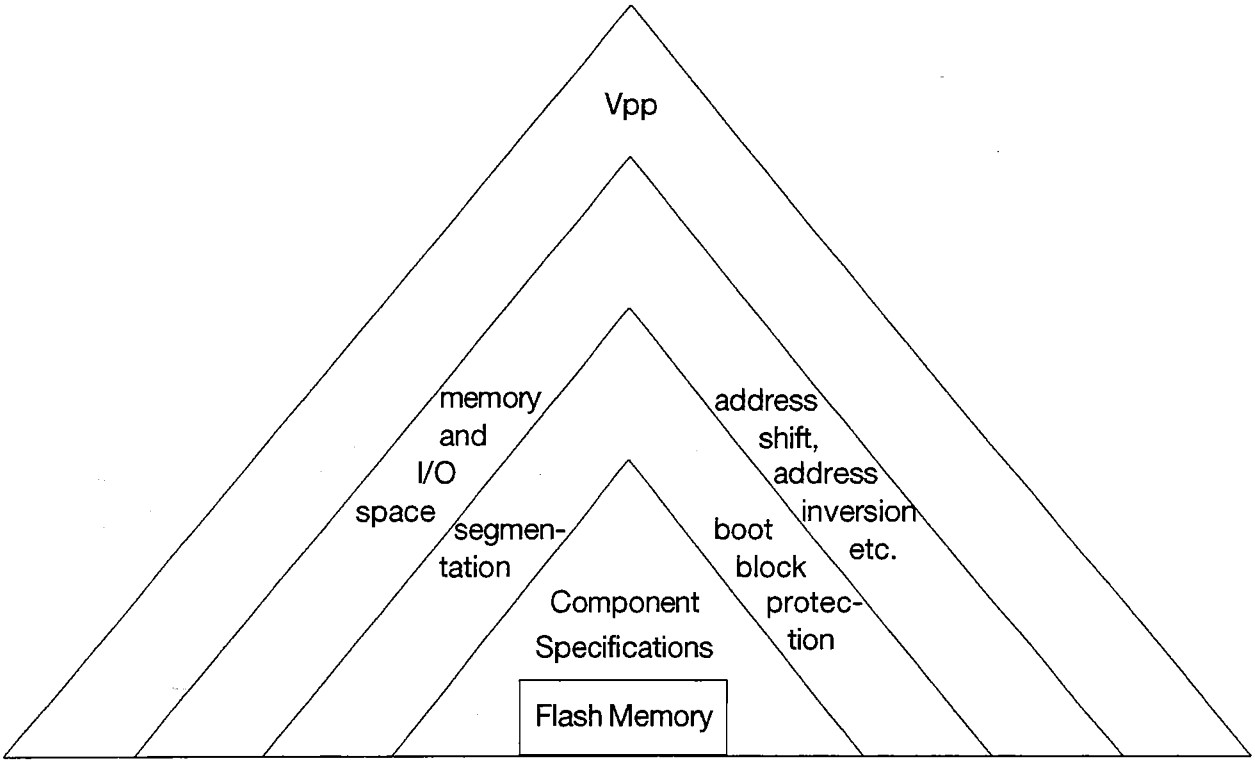


Figure 3. Erase/Program Protection Scheme

memory chip (12V or 5V voltage can be chosen). The producers of complete computers (especially laptops and notebooks) prefer to rely on software methods for controlling and switching the memory modes. This is mainly because of space hungry hardware designs, which don't make the task of taking machines apart (in order to change a switch) fast and easy.

Another, quite drastic solution is to leave Vpp high all the time and leave the memory in the programmable state (erase and program enable). Such a design is not likely to be found in the modern PCs, partly because of the additional power required and partly because of security considerations (a constant program enable state of the memory containing the most important system resources makes a whole system extremely vulnerable to an accidental destruction by overwriting valuable BIOS code).

From the point of view of the user updating the Flash BIOS, the activation of the Vpp can be achieved in three different ways:

- by using the program provided by the producer of the motherboard enabling and disabling Vpp when necessary, depending on the general function selected by the user (eg. Save BIOS, Program BIOS)
- by using one of the BIOS interrupts supporting the Flash EPROM interface (see Appendix)
- by directly accessing the specific I/O devices and executing the sequence of necessary commands, with the required timing.

6.2 HARDWARE ENVIRONMENT

The endless number of hardware and software designs implemented in the personal computers available on the market shifts the level of compatibility to the higher level. Those who write DOS-platform based software know that even using the DOS calls does nor guarantee code portability these days. Relying on BIOS services or direct I/O access is possible only after correctly identifying dozens of different system features like version number, structure, components type, memory mapping or available I/O ports. In most of the systems, direct access to the flash memory (essential for BIOS updating) is not easy to achieve without complete documentation. After shadowing, address shifting or address inverting, the flash BIOS does not appear to the user in the last 64 or 128 KB of the first megabyte of memory. Additionally, the access to the flash chip may be impossible without unlocking the Flash Memory Configuration Space (usually a sequence of read/write operations from/to specific memory and I/O addresses). The I/O port numbers and the data values necessary to gain access to the Command Register differ from one motherboard to another.

6.3 BOOT BLOCK PROTECTION

The segmentation of the memory enables logical segmentation of the code and speeds the erasing and reprogramming of selected parts of the BIOS. The Boot Block containing start up and recovery procedure is additionally hardware protected against undesirable alterations. The erase and programming of the Boot Block is possible only when additional high level voltage is connected to either RP or OE pins.

6.4 COMPONENT SPECIFICATIONS

Different types of flash devices take different additional measures to prevent accidental memory alterations. The main feature is the special format of commands passed to the Command Register for execution. In the most common solutions, the commands like Erase or Program are two-step instructions. In some cases, producers went even further; Sector Erase memory from AMD: AM29F010 requires six-step Erase and four-step Program commands.

Because of the variety of designs, Flash components implement a function that returns the Intelligent Identifier characterising each particular chip. The Identifier consists usually of two bytes: the manufacturer code and the device code (eg. Intel's 28F001BX-T returns 89H,94H; 28F001BX-B returns 89H,95H; AMD's AM29F010 returns 01H,20H).

7 UPDATING VS DAMAGING

Updating the contents of a Flash BIOS memory using the ISW method requires stepping through a certain procedure. The particular implementations will strongly differ from one another, depending on flash device type, Flash ROM interface and the hardware environment. The general algorithm of updating the Flash BIOS is presented in fig.4.

Using the flash BIOS updating software supplied by the producer of the particular motherboard, a user does not need to worry about identifying the hardware environment and the way of controlling Vpp or accessing the Command Register. However someone who wanted to destroy or infect/trojanise the contents of the Flash BIOS memory would have to properly identify the type of motherboard, type and version of BIOS, type of a Flash chip, the chipset configuration, etc. The possible algorithm of the infection/trojanising and corruption of Flash BIOS are presented in figures 5 and 6.

8 CONCLUSION

The new way of updating the BIOS code without removing the ROM chip from the motherboard and reprogramming the system using only software tools has enormous advantages for both a computer vendor and a computer user. The new procedure cuts the costs of upgrading the system and shortens the waiting time for incorporating any BIOS fixes and new features. Flash devices are already implemented in many different technical solutions not limited to the PC market (laser printers, mobile phones and military equipment).

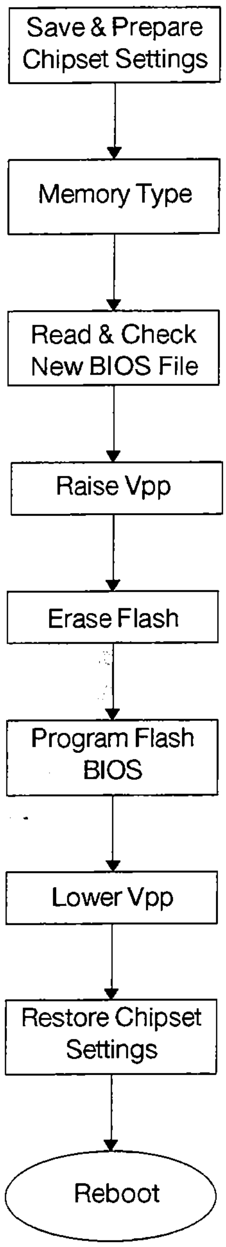


Figure 4.

Flash BIOS Update

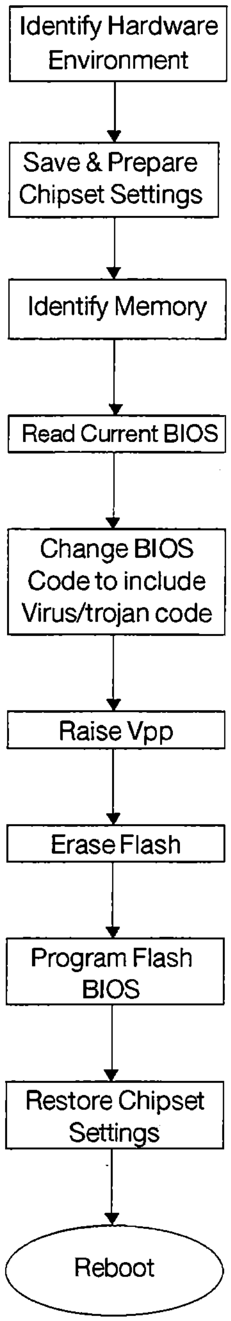

Identify Hardware Environment
Save & Prepare Chipset Settings
Identify Memory
Read Current BIOS
Change BIOS Code to include Virus/trojan code
Raise Vpp
Erase Flash
Program Flash BIOS
Restore Chipset Settings
Reboot

Figure 5.

Flash BIOS Infection

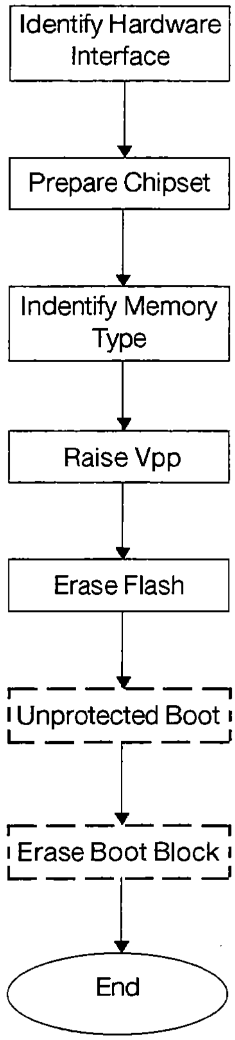


Figure 6.

Flash BIOS Destruction

Since mobile computing is constantly growing, Flash components are finding a large market who are hungry for low-power, high density and extremely rugged elements. Flash Memory Cards and Flash Disks using new faster technology will successfully compete against less reliable mechanical solutions (hard drives).

On the other side, virus writers and generally speaking malicious software authors will try to abuse the fact that destruction of the BIOS code leads to complete system crash. We will see the efforts of reprogramming/damaging the contents of the Flash BIOS memory but very few of them will succeed. The problems discussed in this paper show us quite clearly that the skills and knowledge needed for successful reprogramming the Flash BIOS are far beyond the reach of the average virus writer.

APPENDIX. AMIBIOS

American Megatrends corporation produces one of the most popular and well documented AMIBIOS. The 08/08/93 and later versions provide additional functions to support the American Megatrends Flash Utility (AMIFlash). INT 16H, function E0H provides 14 subfunctions that facilitate the use of the AMIFlash EPROM programming utility, so that it can be used successfully with all types of Flash ROM hardware.

These subfunctions are:

subfunction 00H Get Version Number of BIOS/Flash Memory Interface
This call returns the version number in BCD format stored in the BX register.
subfunction 01H Save and Restore Status Requirement
This call returns in BX number of bytes needed to save the chipset environment.
subfunction 02H Save Chipset Status and Prepare Chipset
This procedure saves the chipset features, disables Shadow RAM, cachememory, power management functions and other chipset features.
subfunction 03H Restore Chipset Status
This procedure restores saved by function 02H chipset features.
subfunction 04H Turn Programming Voltage Off
This function lowers Vpp to its normal level and disables Erase/Program mode.
subfunction 05H Turn Programming Voltage On
This function raises Vpp to the high level and enables Flash memory alterations.
subfunction 06H Write Protect
This function is usually redundant to function 04H
subfunction 07H Write Enable
This function is usually redundant to function 05H
subfunction 08H Flash Memory Select
This function is called only if Flash and standard EPROM are located on the same motherboard.
subfunction 09H Flash Memory Deselect
This function is complementary to function 08H.
subfunction 0AH Verify Allocated Memory
This procedure checks if memory used by AMIFlash is accessible (after disabling the shadowing).
subfunction 0BH Save Internal Cache Status
subfunction 0CH Restore Internal Cache Status
subfunction 0FH CPU Reset
This subfunction doesn't return to the calling program and reboots the system after successful updating of the Flash BIOS.

REFERENCES

[1] 'Flash Memory', vols I,II, Intel Corporation, 1994

[2] 'Flash Memory Products', Data Book/Handbook, Advanced Micro Devices, Inc. 1994

[3] 'Programmer's Guide to the AMIBIOS', American Megatrends, Inc., 1993

[4] 'Intal's SL Architecture', Desmond Yuen, McGraw Hill, 1993

[5] 'GA-586AP', User's Manual, 1995

[6] 'NM29N16 CMOS NAND FLASH E2PROM', Data Sheet, National Semiconductors Corporation, 1994

[7] 'Flash File Systems', Drew Gislason, *Dr.Dobb's Journal*, May 1993

[8] 'The Microsoft Flash File System', Peter Torelli, *Dr.Dobb's Journal*, February 1995

AUTOMATIC VIRUS ANALYSER SYSTEM

Ferenc Leitold

Hunix Ltd, Budafoki ut 57/a, Budapest 1111, Hungary
Tel +36 1 209 2711 · Fax +36 1 166 9206

ABSTRACT

This paper highlights the fact that Bulletin Board Systems can also be used as powerful tools against computer viruses. At the last Virus Bulletin Conference, Jeremy Gumbley presented a paper about Virus eXchange BBSs [1]. This paper intends to show the other side, i.e. a possible way of using BBSs against viruses. The Automatic Virus Analyser System (AVAS) can be used to establish general features of a virus and also to generate the searching and killing algorithms of a virus, with certain restrictions.

1 INTRODUCTION

The large-scale growth of computer viruses sets increasing tasks for anti-virus specialists. On discovering a new and unknown virus, specialists should examine it, add it into their virus database, and develop a remedy. In most cases, these activities represent phases which can be performed nearly automatically. In the automation of these activities, the first step was to develop a prcess called VIRSKILL, which describes the algorithms of seeking and killing viruses [2]. On the one hand, as shown in Fig. 1, VIRSKILL significantly accelerates the development of the remedy for a new virus after its first appearance. On the other hand, VIRSKILL is a powerful tool for sharing searching and killing algorithms.

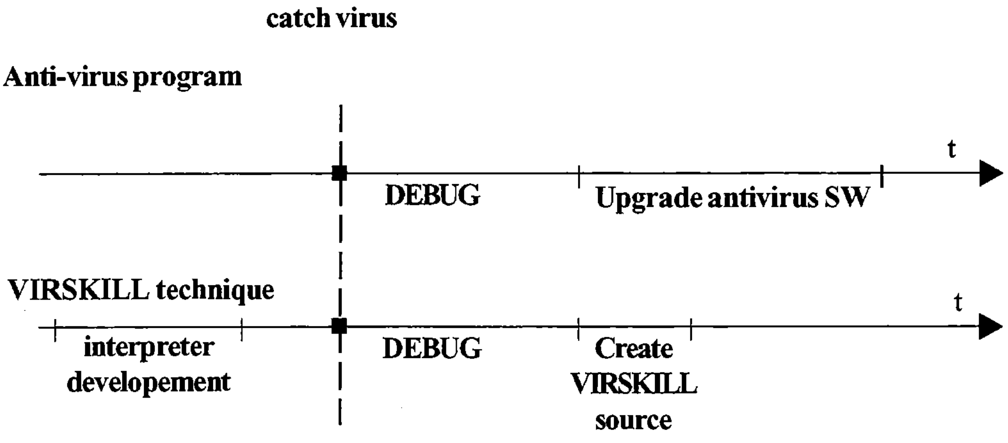


Fig. 1: Upgrading periods I.

However, in addition to the development of searching and killing algorithms, anti-virus specialists should also perform an examination of the virus and insert the data into the database. The Automatic Virus Analyser System (AVAS) is intended to provide assistance in performing these tasks.

2 THE PURPOSE OF THE AVAS

During the examination of a virus, the AVAS discovers its basic features as follows:

- what the infected code areas are
- whether or not the virus is polymorphic or not
- which anti-virus program is able to identify the virus; by which name, and whether it can be completely 'killed'
- whether the virus is already stored in the database

It frequently occurs that certain features (e.g. spreading) are different when examined under different versions of operating systems. Thus, it is desirable to perform the examination under several operating systems, which further increases the large number of repeated actions.

Following the establishment of the basic features, the AVAS is capable of formulating the searching and killing algorithm for the virus in the language VIRSKILL. The algorithm thus made is verified, using a large number of virus infections. If the virus has not yet been inserted into the virus database, the AVAS then inserts it, together with the results obtained.

Considering that it is impossible to develop a perfect anti-virus system, even AVAS has its limits of operation as follows:

- it is only capable of examining viruses which also infect executable files under DOS
- the searching and killing algorithm developed can only be used in the case of file infections
- the system is unable to yield results with every kind of virus. It may be that the virus reproduces itself only under certain circumstances. The virus may also be polymorphic. In fact, in the case of polymorphic viruses, the development of the searching and killing algorithm is more difficult, as the traditional sequence-searching algorithm cannot be used in most cases.

3 STRUCTURE OF THE AVAS

The main tasks of the AVAS are performed by two PCs; the MASTER and the SLAVE:

- The MASTER controls the operating cycles of the SLAVE, as well as examining, evaluating and collecting the results obtained by the SLAVE.
- The SLAVE computer serves the MASTER while changing the different operating systems, generating a virus-free environment, and reproducing viruses.

Of course, it is also possible that a third PC is charged with the task of serving the user(s), e.g. through BBS, Internet, or some other medium.

3.1 HARDWARE AND SOFTWARE ENVIRONMENT

As mentioned, the main tasks of the AVAS are performed by two PCs (Fig. 2). On the one hand, the communication between the two PCs is performed through an Ethernet cable, using the Novell NetWare Lite software. On the other hand, the MASTER can issue either of the two commands listed below to the SLAVE:

- ***Reset:*** restarts the SLAVE. The same function is obtained by depressing the <Reset> key on the SLAVE.
- ***ChangeFloppyDrives***: inverts the sequence of drives A and B in the SLAVE.

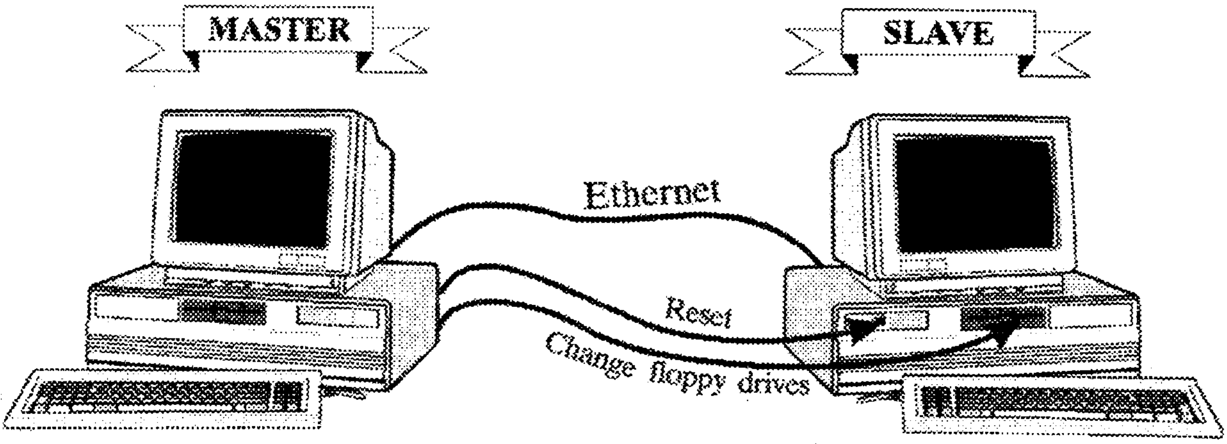


Fig.2: The structure of AVAS

These two commands enable the MASTER to create a clear and virus-free environment for the SLAVE. In fact, during operation of the AVAS, one of the floppy drives on the SLAVE holds a write-protected boot disk, while the other drive is empty. Thus, the MASTER is capable of changing the boot drive of the SLAVE, on the one hand, and perform the booting on the other hand. This link between the MASTER and the SLAVE is connected to a parallel port on the MASTER. Within the SLAVE, a dedicated electronic unit controls the reset terminal of the mainboard as well as 'changing over', certain wires of the flat cable connecting the floppy drives and the controller.

3.2 THE STATES OF THE SLAVE

During the operation of the AVAS, the active drives are changing in the SLAVE. This is done by changing the Master Boot Record (MBR) of the SLAVE and changing the CMOS.

The SLAVE includes an IDE disk area of 130-MByte capacity, divided into 22 logic drives (Fig. 3). The change among the various operating systems is performed on the first partition (VIRTEST) located at the beginning of the disk. This partition serves for reproducing the viruses. It has a 30-MByte capacity, which is limited by the fact that it has to function as a boot partition, e.g. even under the operating system MS-DOS 3.30. The next partition, which has a 20-MByte capacity (COMMON), stores the programs independent of the operating systems which are necessary for the operation of the SLAVE. The various operating systems are stored on the next 20 partitions of 4 MBytes each.

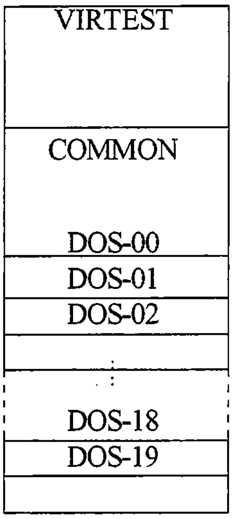
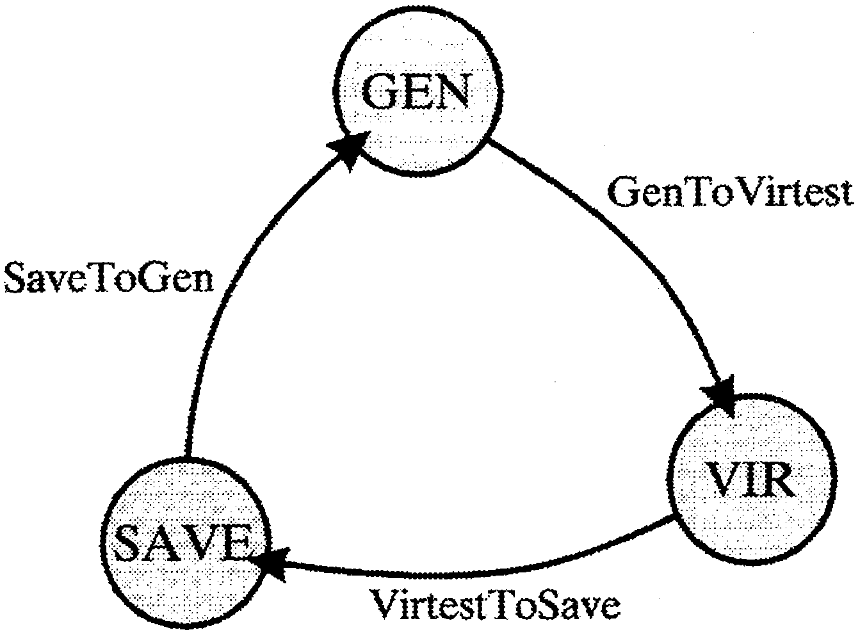


Fig. 3: The structure of the disk in SLAVE

During the operation of the SLAVE, three principal states are possible as follows (Fig. 4):

- virus reproduction state (**VIR**)
- data saving state (**SAVE**)
- regeñeration state (**GEN**).

Fig. 4: States of the SLAVE



The SLAVE regards the partitions mentioned above and the floppy drives as different logic drives in its various states. The disk parameters of CMOS also change in various states. This is represented in Fig. 5.

States/Valid drives			Disk parameters				
					Start	End	Cyl
VIR	SAVE	GEN	Name	size	Cyl	Cyl	Num
C:	C:	E:	VIRTEST	30 MByte	0	147	148
-	D:	D:	COMMON	20 MByte	148	240	93
-	-		DOS-00	4 MByte	241	260	20
-	-	C:	DOS-01	4 MByte	261	280	20
-	-		DOS-02	4 MByte	281	300	20
-	-		DOS-03	4 MByte	301	320	20
-	-		DOS-04	4 MByte	321	340	20
-	-		DOS-05	4MByte	341	360	20
-	-		DOS-06	4MByte	361	380	20
-	-		DOS-07	4MByte	381	400	20
-	-		DOS-08	4MByte	401	420	20
-	-		DOS-09	4MByte	421	440	20
-	-		DOS-10	4 MByte	441	460	20
-	-		DOS-11	4 MByte	461	480	20
-	-		DOS-12	4 MByte	481	500	20
-	-		DOS-13	4 MByte	501	520	20
-	-		DOS-14	4 MByte	521	540	20
-	-		DOS-15	4 MByte	541	560	20
-	-		DOS-16	4 MByte	561	580	20
-	-		DOS-17	4 MByte	581	600	20
-	-		DOS-18	4 MByte	601	620	20
-	-		DOS-19	4 MByte	621	640	20

Fig. 5: Active drives and partitions in difference states

In the **virus reproduction state (VIR)**, according to the CMOS disk parameters, there is only one disk of 149 cylinders in the computer. In this state, the BIOS is unable to handle the remaining cylinders. In logic respect, only the partition VIRTEST is active (as drive C); drive A does not include a floppy diskette, while a write-protected diskette is inserted into drive B. Thus, the reproduction of the virus does not endanger other disk areas.

The **data saving state (SAVE)** differs from the virus reproduction state in that the partition COMMON also appears as drive D. In addition, drive A is that drive which contains the floppy diskette. Thus, in this state, the virus-free environment can be ensured by booting from the virus-free diskette in drive A.

In the **regeneration state (GEN)**, drive B contains the floppy diskette. Booting will be performed from the partition of the operating system to be generated. This is drive C (DOS-xx). As in the data-saving state, the partition COMMON appears as drive D, while the partition VIRTEST is shown as drive E.

4 FUNCTIONS OF THE SLAVE AND THE MASTER

During the operation of the AVAS, the SLAVE and the MASTER function by turn: as long as one of them performs some function, the other waits for the termination of that activity. The only exception is the virus reproduction state, in which the SLAVE performs virus reproduction while the MASTER monitors it continuously. One group of activities belonging to the SLAVE performs the change-over among the states of SLAVE as follows:

- **GenToVirtest:** After regenerating the partition VIRTEST, the regeneration state is changed automatically to the virus reproduction state. On terminating regeneration, both the CMOS and the MBR will be updated and, by restarting, the SLAVE enters into the virus reproduction state.
- **VirtestToSave:** The SLAVE can be entered from the virus reproduction state to the data saving state with the aid of the MASTER. The MASTER changes over drives A and B of the SLAVE and restarts the SLAVE. Thus, the SLAVE will be booted from the floppy diskette in drive A. During the booting procedure, it detects that no drive D (partition COMMON) is included. Therefore, after updating the MBR and the CMOS, it restarts the computer. At the time of the second booting from the floppy diskette, drive D already exists; thus, the SLAVE will be set in data- saving state.
- **SaveToGen:** On termination of the data-saving state, the MASTER places a file describing the next MBR and CMOS contents through NetWare Lite onto the partition COMMON. These items of information will be written by the SLAVE into the MBR and the CMOS, respectively, and the computer restarts itself.

The other group of activities belonging to the SLAVE includes those performed by the SLAVE in the specific states:

- **MakePart:** In the regeneration state, the SLAVE creates the full content of the partition VIRTEST; i.e. the files of the actual operating system, the files to be infected, and the file(s) already infected. Furthermore, the files ensuring operation of the virus regeneration state, the files of NetWare Lite and the batch file performing the running will also be created. Of course, the SLAVE has to connect to the MASTER during regeneration, in order to copy the files to be examined (reproduced).
- **MakeInfections:** In the virus reproduction state, the first step is for the SLAVE to make connection with the MASTER through NetWare Lite. This is necessary to enable the MASTER to monitor the infection process. Then, the SLAVE starts the files already infected and, subsequently, those not infected. This activity will be repeated until the MASTER restarts the SLAVE.
- **PassToMaster:** In the data saving state, the SLAVE makes connection with the MASTER and waits until the MASTER performs saving and evaluation. Then, the SLAVE enters into the regeneration state.

As a result of the function PassToMaster, control passes to the MASTER. The MASTER performs two main functions: it evaluates the files infected, then creates the files necessary for the generation of the subsequent virus-free environment and passes control to the SLAVE. The evaluation of files presents the following fundamental functions:

- **CheckIntegrity:** By examining the integrity of the files, this function establishes which files were changed, and identifies the difference between the original and the infected files.
- **CheckPolymorph:** Examining the infection of the files, this establishes whether or not the virus is polymorphic. This will be obtained by finding bytes which are the same in the infected files. Using this byte mask, the polymorphic character can be recognized.
- **CheckDatabase:** This establishes whether the virus is in the database but can be perfectly performed only in the case of non-polymorphic viruses.

- **CheckAntiVir:** By running the anti-virus products stored in the anti-virus database, the 'hit probability' of recognizing the virus is established.
- **MakeStrain:** If the virus is not polymorphic, a searchstring will be selected from among the identical bytes.
- **RunEmulator:** If the virus is polymorphic, this function creates the searching algorithm for the virus. In certain simple cases, this procedure is also suitable for creating the killing algorithm of the virus. Using this function, a processor emulator tests the instructions and activities carried out in each infected file.
- **TestAlgorithm:** This function tests the searching and (if any) killing algorithms created by MakeStrain or RunEmulator.

Further fundamental tasks of the MASTER consist of performing the functions listed below. These are necessary for the next generation:

- **PassToSlave:** The MASTER places a file describing the next MBR and CMOS contents onto the partition COMMON. The SLAVE monitors the creation of this file and, after closing it, the SLAVE updates the MBR and the CMOS and enters into the regeneration state.

As a result of the PassToSlave function, 'control' passes to the SLAVE. Then the MASTER performs the activity Look&Reset:

- **Look&Reset:** The MASTER monitors continuously the start of the virus reproduction procedure. During virus reproduction, it examines the changes in the monitored files through NetWare Lite. If each file is infected, or the time-out expires, the MASTER restarts the SLAVE by changing the sequence of floppy drives, and waits for the PassToMaster function to be initiated.

5 WORKING PERIODS

In respect of the examination of viruses, the AVAS includes two databases which are subject to continuous changes:

- the virus database (including the virus and its features)
- the anti-virus programs database

Both of these databases may change continuously, as new viruses, or a new version of an anti-virus program appear. Of course, installing a new anti-virus program, means that the complete database has to be examined: the examination of the viruses included in the database has to be repeated (!). On appearance of a new virus, examination is significantly faster; in fact, only one virus must be examined. This examination procedure is represented in Fig. 6.

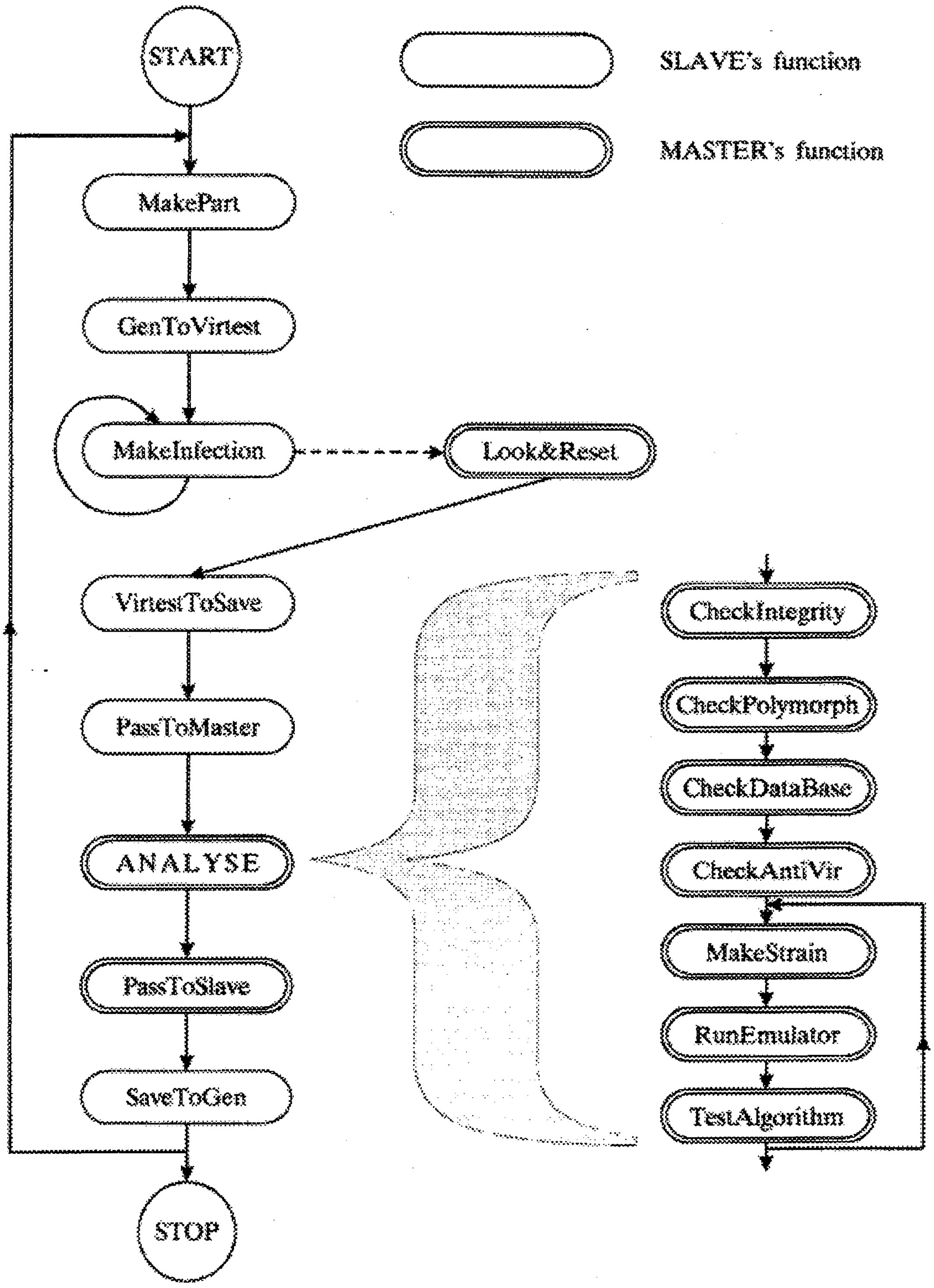


Fig.6: Working Cycles

6 SUMMARY

This paper has highlighted how the Automatic Virus Analyser System could be used to establish general features of a virus, and how it can generate the searching and killing algorithms of a virus, (with certain restrictions). Of course, this system cannot be perfect, because no anti-virus system is perfect. However, it should be possible to use this system widely. Using AVAS, the upgrading period of anti-virus software is much easier (Fig.7).

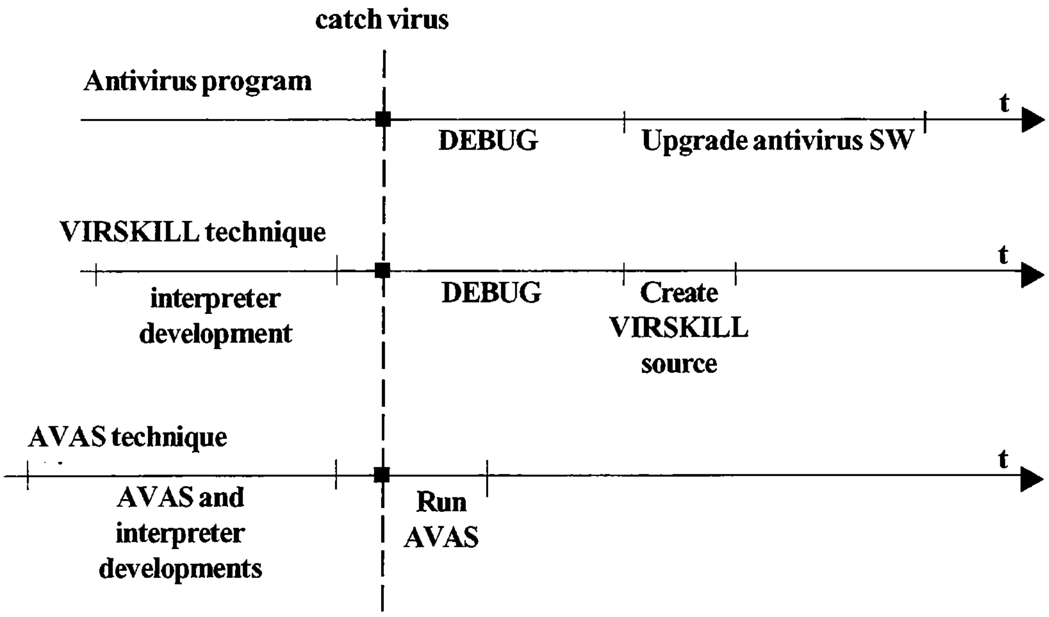


Fig. 7: Upgrading periods II

REFERENCES

[1] Jeremy Gumbley: VX bulletin boards, *Proceedings of 4th International Virus Bulletin Conference*, Jersey, UK, 1994.

[2] Leitold, F.: Csótai,J.: Virus Searching and Killing Language, *Proceedings of 2nd International Virus Bulletin Conference*, Edinburgh, 1992.

THE PROBLEMS IN CREATING GOAT FILES

Igor G. Muttik

S & S International PLC, Alton House, Gatehouse Way, Aylesbury, Bucks HP19 3XU, UK
Tel +44 1296 318700 · Fax +44 1296 318777 · E-mail MIG@sands.co.uk

ABSTRACT

Having more than 6000 viruses for IBM PCs, the maintenance and updating of a virus library of samples is a difficult task. Parasitic file infectors are the majority of this great quantity and testing their properties and creation of samples takes great effort. To help solve this problem, the author has developed a special tool for anti-virus researchers, which allows the creation of bait files (also called sacrificial goats). Theoretical points of bait creation (infectable objects, unusual infection conditions, environmental requirements) are discussed and a detailed description of the GOAT package is given.

This paper is an attempt to summarize the problems which appear during weeding suspicious files and replicating viruses. A safe testing environment based on hardware hard disk drive (HDD) protection is described. The paper also describes DOS peculiarities, which appear when working with long directories.

The possible appearance of viruses targeted against anti-virus research environments are discussed.

1 VIRUS SAMPLES

1.1 WHAT IS 'A VIRUS SAMPLE'?

A file-infector virus usually attaches itself to an executable file using an appending or prepending technique. Such viruses are called parasitic infectors. Among anti-virus researchers these viruses are usually transferred in the 'sample form' - the virus is attached to a do-nothing file of some fixed size (usually divisible with 10**N or 16**N) and simple contents (do-nothing or printing a short message on the screen). The result of infection of such a goat file is called "a virus sample". We have:

Virus sample = Virus(Goat file)

or, simply:

Virus sample = Goat file + Virus

Can we 'standardize' the virus sample? Generally speaking, unfortunately not. All polymorphic viruses have zillions of instances and it is impossible to select some 'standard' image of such a virus. Oligomorphic and encrypted viruses are difficult to 'standardize' too. Even for non-encrypted viruses the problem is not simple - they usually have some variables, stored inside their body (especially resident viruses) and, though, their image is variable.

1.2 TYPES OF GOAT OBJECTS

We have many infectable objects in the DOS environment. This includes:

1) Files:
- EXE/COM/OV? executable files (usually started by the user)
- SYS drivers (called by DOS kernel at startup)
- BAT files (run on user request or from AUTOEXEC.BAT)
- OBJ/LIB/source files (compiled into executables on the user request)
- DLL/CPL/etc. (NE, LE, etc. - Windows, OS/2 executables)
- DOC/WK?/etc. (including macro and OLE files)

2) Pointers:
- MBR partition table (à la *Starship*)
- DBR pointers (IO.SYS/MSDOS.SYS; IBMBIO.COM/IBMDOS.COM)
- directory entries (ex., *DIR-II* family)
- FAT pointers (ex., *Necropolis*)

3) Startup code:
- Flash ROM (called by microprocessor after RESET)
- MBR code (called by ROM BIOS after POST)
- DBR code on HDD (called by MBR code)
- DBR code on floppy (called by BIOS)
- DOS kernel code (called by DBR code)

Each mentioned object can be infected and, therefore, requires preparation of a 'goat object'. Fortunately, most types of unusual infection techniques are very rare or even not yet found. And creation of bait objects for bizarre viruses is a rare task - the great majority of known viruses are simple parasitic file infectors. Furthermore, the creation of a goat BAT file (or source file) is rather easy - one can use a text editor to make a bait for the virus. To create a goat floppy diskette we can use standard FORMAT utility.

Anti-virus researchers are mostly disappointed with a problem of 'virus glut' [Skulason]. 'Virus glut' means the rapid increase of the number of known viruses, the great majority of which are file viruses. So, in most cases, the attention of anti-virus researchers is focused on parasitic file infectors. We'll discuss only this type of virus in the rest of the paper.

1.3 CREATION OF GOAT FILES

To try to replicate a virus, one has to have a set of goat files. Most anti-virus researchers have their own pre-created sets of files, produced using an ASM source or directly from the DEBUG utility. This approach has a drawback - if new goat file is required it has to be created manually. And if we need a lot of files (eg., for testing polymorphic virus detection rate) the process must be repeated many times.

Obviously, a specific automated tool has many more options and capabilities. It can create sets of files on one invocation.

It is convenient to use a set of goat files with linearly increasing length (say, 1000, 2000, ...20000). If the virus leaves alone short victims after infection, and file growth can be calculated subtracting the size of the infected file from the original size, this will be easily noticeable.

2 INFECTION OF A GOAT FILE

2.1 'WEEDING PROBLEM'

From the point of view of an anti-virus researcher all incoming suspicious samples should be classified in one of the following groups (for definition - see VIRUS-L FAQ [FAQ]):

- innocent file (includes garbage and damaged programs)
- virus (includes germs, droppers, viruses of the 1st generation)
- trojan
- intended
- joke

One mentioned classification problem is called 'weeding'. There are automated and manual methods, used to weed a set of files. The following automated tools are used:

- scanners, detecting viruses by name
- heuristic scanner
- TRASHCAN/DUSTBIN, detecting non-viruses, jokes, garbage and intendeds

Manual 'weeding' methods are used after automatic ones:

- visual analysis (eg., presence of 'MZ', 'PK' identifiers)
- tracing in DEBUG (includes partial on-the-fly disassembling)
- full disassembling

We should take into account that the infected sample may be compressed with one of the EXE-packers (PKLITE, LZEXE, DIET, EXEPACK, COMPACK, PGMPACK, KVETCH, SHRINK, TINYPROG, WWPACK, AXE, IMPLODE, AVPACK, etc.). In such a case UNP and UUP programs should be used to remove the compression code before manual analysis.

Visual checks of incoming suspicious files are usually made using DEBUG or HIEW (Hackers View) - a a wonderful viewer of executable files. The latter combines features of simple ASCII/HEX viewer with a built-in disassembler/assembler (both 16 and 32-bit modes) and a binary file editor. Although I can hardly recommend this utility for all anti-virus researchers.

2.2 SAFETY PROBLEM

Every anti-virus researcher faces a problem when he needs to start the infected (or just suspicious) program or trojan horse. The usual solution is to use a special goat PC (usually an old PC/XT/AT). But the malware can easily destroy data on the hard disk of this PC. It can even cause malfunction of the hardware (eg., low-level format IDE disk, if any) and it will take significant time and effort to restore your testing environment. The hardware protection of the hard disk of your PC can only be a 100%-reliable solution. To make hardware protection you will need some switch, which selects an operation mode - 'normal'/'protected'.

2.2.1 Hardware protection using 'Turbo' switch

'Turbo' switch is rarely used in computer operation for the following reasons: first, any user will usually select the highest possible speed to minimize the response time of the software. Second, most available BIOSes support toggling of turbo mode using the keyboard eg., AMI BIOS uses [Alt]-[Ctrl]-[+] to set higher speed and [Alt]-[Ctrl]-[-] to set lower speed). Therefore, you can easily replace your connection of

'Turbo' switch to the motherboard with a simple jumper. Now your 'Turbo' switch connector is free for use as a hard-disk protection switch. Typically the connector of 'Turbo' switch has three contacts (and a switch shorts two left contacts or two right ones). The use of this switch to turn the disk protection on/off looks like an elegant solution.

Now find the jumper on your hard disk controller, which enables its operation (examine the controller manual if needed). Most MFM, IDE and SCSI controllers have such a jumper. Remove this 'HDD-enable' jumper and substitute it with the connector of 'Turbo' switch (the connector should replace the jumper on the controller and short the contacts instead of the jumper).

Now, after the above modification, you can easily turn off the HDD by simply pressing the 'Turbo' switch pressing it again to return it to operation.

The LED indicator (or simple LED) of your PC (which usually shows the current frequency of processor operation) is wired to the turbo switch and reflects its state. You can easily configure the LED indicator to reflect the current mode of operation (say, 'On'/'FF').

2.2.2 Software shell for hardware protection

To work without HDD you will need some media instead of it. An ideal solution is to use a ramdrive. You have to add the following statement to your CONFIG.SYS:- DEVICE=RAMDRIVE.SYS nnnn (where nnnn stands for the size of ramdrive in kilobytes, you may also need /e switch to use extended memory). Ramdrive size <2MB is usually not sufficient, so it's better to select 2-4MB.

First, copy all software, needed for virus testing (plus suspicious files) to your virtual disk. When your hard disk is switched off, all programs will be inaccessible, so make a good selection (in my case it took around 1MB or more). Now you are ready to disable hard disk. But DOS still thinks that HDD is present. Its internal buffers and cache utilities (if any) still remember the current contents of some portions of your hard disk in the computer memory. The most obvious solution is the elimination of all 'notes' about hard disk presence. To simulate the absence of a hard disk on the PC, I wrote a special program, which clears INT_41h and INT_46h (pointers to the HDD disk tables) and sets the number of available hard disks (BIOS variable at [0:475h]) to zero. To reroute any access from hard disk (eg., drives C:, D:, E:) to the virtual disk, I use DOS' SUBST utility, which replaces drives C:, D: and E: with the virtual disk drive letter (F: in my case). SUBST also clears HDD cache contents. Finally, the DOS environment variables (eg., COMSPEC and PATH) should be rewritten to point on the ramdrive objects.

2.3 'REPLICATION PROBLEM'

The problem of infecting a goat file, with a sample of a possible virus is called 'replicating'. Very often one researcher will ask the others, 'I have a sample that I think is a virus, but cannot replicate it. Have you tried? If anybody has succeeded in doing this, send me a sample, please...' And this is repeated very frequently. We see that the 'replication problem' is one of the most common problems. The question is to find correct computer environment and meet all virus infection conditions. Obviously, both problems can be solved with the help of full disassembly of the viral code, but that is not a very practical approach, because it takes much time. Usually, suspicious files are simply tested in the so-called 'goat computer'. Only in the case of problems (files not replicating, but looking suspicious) are they disassembled and analyzed in deep. We have already seen one approach to the replication problem - to ask for help from other researchers. There are also other options:

- try a lot of different goats
- try a lot of different environments
- manual analysis (tracing, debugging, disassembling) to find out all infection conditions (i.e., requirements for the goat and environment).

2.4 INFECTION CONDITIONS

To replicate a virus we have to feed it a goat file, which meets internal virus infection conditions. This must be done in the environment which is appropriate for the current virus. Fortunately, to make viruses more infective, they are usually made to operate in a wide range of environments. On the other hand, sometimes, numerous limitations are implemented to simplify the viral code (eg., *Ping-Pong, Vindicator, Yale* and *Exeheader.Mz1* viruses work only on 88/86 processors; *3APA3A* and *MIREA.4156* viruses require a 16 bit FAT hard disk; the *AT144* virus requires a 286 processor or higher; The *Green_Caterpillar* virus needs a CMOS clock; the *Lovechild* virus requires MS-DOS 3.2; the *Nightfall* virus does not replicate without an XMS driver [Brown]; the *EMMA* virus requires the presence of EMS [Kaspersky]; etc.). In the case of specific requirements, only a random environment selection or manual analysis of the virus internals may help to find the correct environment.

Parasitic file infectors can theoretically infect all of the following types of files:

- COM
- EXE

  MZ/ZM (DOS executables)

  NE (Windows, OS/2 16-bit)

  LE, W3 (Windows VxD, Win386)

  LX (OS/2)

  PE (Windows, NT 32-bit)

  MP, P2, P3 (Pharlap DOS extenders)
- SYS/COM (normal DOS drivers)
- SYS/EXE (understood only by DOS 5.0, 6.0)

There are following infection conditions (except file type)

- file size
- filename
- attributes
- file timestamp (date/time of creation/modification)
- file contents

The most common infection condition is file type (COM/EXE) and the second is the size of the victim. Very short files are usually avoided, because their growth is too noticeable. Infection of do-nothing goat files (like primitive INT_20, 2-byte files)is also avoided.

Most file infectors are targeted against simple DOS executables - COM files and EXE files (with MZ or ZM markers). Some file infectors are capable of infecting DOS drivers of SYS type (eg., *SVC.4644, SVC.4661, SVC.4677, Alpha.4000, Astra, Astra_II, Cysta* or *Comsysexe, Terminator.3275, CCBB, Talon* or *Daemaen, Ontario, VLAD.Hemlock, Face.2521,* etc.). All other formats of executables need reclamation of the virgin lands from virus writers. For example, there are only a few known Windows viruses up to date (all infecting only executables in NE-EXE format).

Speaking of the contents of goat files, we should mention that viruses, which check the internals of the victim file, are rather rare. I do not mean a selfcheck to avoid multiple infections of the same file. I mean checking of virus-free areas (same as inspection of the uninfected file). Nevertheless, such viruses exist.

The *Lucretia* virus looks for an 0xE8 byte (Intel x86 CALL instruction) in the file and replaces the offset of the call to point on the viral body. The *Warlock* virus avoids all files having the 0Eh byte at the start of program code (includes all LZEXE-packed programs). *Raptor* virus does not infect EXE files with SS in the header equal to 07BC, 141D, ...2894 (13 entries). The behaviour of *Internal.*1381 virus depends on the contents of the EXE header too. Moreover, there are *Zerohunter* viruses, which look for a series of zeroes (412 bytes for *Zerohunter.412* and 415 for *Zerohunter.415*) in the file and infect the victim overwriting this block of zeroes, if found. *Zerohunter* viruses are typical representatives of the class of 'cavity viruses' (like *Helicopter.777, Grog.Hop, Gorlovka.1022/1024, Russian_Anarchy.2048, Locust.2486, Tony.338,* etc.).

There are also viruses of the exeheader type - *Dragon, Hobbit, SkidRow, Mike, VVM, Bob, XAM, Mz1, Pure,* etc. They infect only EXE files, having a long block of zeroes (around 200-300 bytes) in the EXE header (it is 512 bytes by default). They can be regarded as a subclass of cavity viruses.

Many viruses do not infect some programs. They usually avoid command processor COMMAND.COM and certain anti-virus or widely used programs (archivers, command-line shells, etc.). The following reason come to my mind: infection of COMMAND.COM is very noticeable and causes many incompatibilities, so virus writers simply filter-off COMMAND.COM to avoid compatibility problems. This approach has a drawback (from the virus writer's point of view), as the infection of COMMAND.COM with a resident virus guarantees that the computer will come up with a virus installed in memory, because COMMAND.COM is always automatically invoked during the boot process. Viruses try to avoid anti-virus programs - they normally check their own integrity and a virus will be detected immediately. The list of viruses that avoid infection of certain programs is given in Table 1.

A more difficult case occurs if the virus infects only on certain days of week, or during the first 20 minutes of an hour (like *Vienna.644.a* does). For example, the *Kylie* virus affects the victim if the current year is not 1990. The *Fumble* virus infects only on even dates. The virus called *Invisible* avoids certain COM files by doing a checksum on the name of the victim. Viruses of the *Phoenix* family (also called *Live_after_Death*) avoid some file sizes and about 1/8 of files are left uninfected. The *Russian Mirror (Beeper)* virus infects only every third executed file. Some of these viruses are called 'sparse' infectors. Random environment/goat selection may not help in this case and viruses have to be traced and/or disassembled.

Many viruses require a JMP instruction in the beginning of the victim file (eg., the first versions of *Yankee_Doodle, Russian_Tiny.143, Rust.1710, Screen.1014, Leapfrog.516,* etc.)

All mentioned exclusions and conditions must be taken into account when trying to create goat files suitable for the infection and if the virus does not replicate.

2.5 INFECTION MARKERS AS AN OBSTACLE FOR INFECTION

Almost all viruses try to 'mark' their victims to avoid multiple infections of the same file, because the growth of files beyond some reasonable limit cannot go unnoticed (because of waste of disk space and delays for the reinfections) and may even cause infected an file to hang (eg., COM file >64k). Viruses use different 'infection markers':

- detection of self-presence (check own code; full or partial)
- sequence of bytes (text or binary designator; usually at specific position)
- timestamp (62 seconds, >2000 year, etc.)
- file size (ex., *Uruguay-#3, #4*)
- attribute (some viruses mark their victims as ReadOnly).

Some viruses use perfectly legal markers - for example, seconds value (say, all infected files have 33s) or file length (say, all infected files' lengths are divisible by 23). If, occasionally, our goat file carries a 'marker' of the virus, it will not be infected. Some unusual infection markers are listed in Table 2. Fortunately, most viruses use specific markers. In fact, viruses have to behave in such a way to be infective. Therefore, it is usually easy to make an infectable goat file if the first attempt of replication failed because of a coincidence with a legal virus marker.

2.6 CHECKING OF GOAT FILES AFTER ATTACK

After trying to infect a goat we have to detect possible changes. If we see file growth (in a directory listing) - the reason is obvious: longer files are virus children. One additional test is recommended - to check whether virus child is itself replicating. In some cases (because of the errors in the virus) it is not and, therefore, must be classified as intended, not a virus. Visual checks after the attack are made just like before the attack - see 2.1.

If the virus has stealth or semi-stealth properties, the detection of infected samples is somewhat more complex. The best approach is to preserve all goat files, involved in the test and inspect them after a clean reboot (copy them to a floppy disk if your HDD is disabled as described in 2.2). More simple, but not that a reliable method, try to remove the virus from the interrupt chains using, say MARK/RELEASE programs by TurboPower Software. MARK should be installed before the first start of the virus, it remembers the whole interrupt table; RELEASE should be started after the attack to restore the old interrupt table and remove the virus from the interrupt chain). Unfortunately, this approach might not work if the virus uses tunneling.

In principle, we can use an integrity checker to compare test files before and after the virus attack. This generic method can detect almost all stealth viruses if used in the low-level disk access mode. For example, this mode is available in the Russian integrity checker ADInf.

3 'POLYMORPHICS DETECTION RATE'

3.1 HUGE QUANTITIES OF GOATS

In product reviews, we frequently read something like the following: "... the 'Polymorphic' test-set contains a mammoth 4796 infected files" [TOP] or "When tested against the 500 positively replicating *Mutation Engine (MtE)* samples, all but two were correctly detected as infected" [Jackson]. Why all these tests need so many samples of the same virus? The answer is simple because of the great variability of polymorphic viruses (more correctly - because of the variability of the virus decryptor). Any scanner coping with polymorphics has to decrypt the body of the virus and locate a search-string. Another approach is to try to distinguish the viral decryptor from normal non-viral code. Both methods can produce both false positives and false negatives. They are, of course, rather rare, but practically (and even theoretically) unavoidable. To find out the weaknesses of the scanner, the number of tested samples should be very high. That is why almost all comparisons of scanners are performed using very large samples. That is, unavoidably, of course, rather time consuming and not very convenient practice.

How can we speedup the tests and preparation of samples? The first idea is to put virus samples on fast media - virtual disk looks the ideal selection. But can we enhance DOS' access to the drive?

3.2 DOS SLOWDOWN WHEN WORKING WITH LONG DIRECTORIES

When experimenting with the creation of hundreds of files, I have noticed a very interesting peculiarity. After creating a number of files in the directory (in my case, around 700 files) all additional files needed much more time to be created! Obviously, some internal resource of DOS was exhausted. To shed the light on this effect I have run the same task - creation of 100*N goat files (N=1..10) using GOATS (with no size increase; i.e., all goats were identical), but thevaried number of BUFFERS (as written in CONFIG.SYS). Note, that the disk cache (SMARTDRV) was not active, because files were created on the virtual disk. Collected data is given in the table:

Time needed to create given number of files (in seconds +/-1).

FILES	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
BUFFERS										
15	6	12	19	28*	40	51	64	80	96	118
48	6	12	19	27	35	45	55	70*	90	112
58	6	12	19	26	35	45	55	70	82*	103
68	6	12	19	26	35	45	55	70	82	98

Note: '*' - shows number of files, when significant slowdown occurs.

1. We see that total time greatly depends on the number of BUFFERS.
2. At some place significant slowdown always occurs (compare columns to see).
3. The moment of this slowdown depends on the number of BUFFERS.
4. For creation of 1000 files, 68 BUFFERS are sufficient.
5. For 48 BUFFERS slowdown occurred at around 720 files.
6. For 58 BUFFERS slowdown occurred at around 870 files.

Thus, addition of 10 BUFFERS (10*512=5120 bytes) shifts the limit on (870-720=150) files. We can calculate how many bytes are needed per file - 5120/150=34.1. Surprisingly, this is very close to the directory entry size! This is additional evidence that slowdown occurs when there is no more space in BUFFERS to store current directory (and DOS needs to reload it from disk).

I have also found an interesting fact (not yet known to me) - the creation of files in a fresh directory takes much less time, than the creation of the same number of files in the same directory after removing 1000 files! And the creation time for 1000 files in used directory is approximately **three times more** than in a fresh directory! That is because DOS scans a directory only until it encounters zero entry. And for a used directory there are no such entries (at least near the beginning) and DOS has to scan the whole list of deleted entries.

Thus, we have to create bait files in a set of fresh directories of moderate size. The same applies to the testing of scanners against huge virus collections - fresh and short directories will be scanned faster.

4 GOAT SOFTWARE PACKAGE

After discussing some theoretical points, let's turn to the realization of these ideas in the GOAT package [GOAT]. This package is a set of tools for anti-virus researchers, which help to create bait files (also called sacrificial goat files or, simply, goat files).

The purpose of the programs can be explained using the following table:

You need	Use
Bait file with some special internal structure	GOAT.COM
A series of bait files of different sizes	GOATS.COM
Files of the same size, but with different contents	GOATSET.BAT
Many identical files to infect them with polymorphic virus	FLOCK.COM

Using GOAT.COM you can manually select the size, the name of a sacrificial goat file and vary its internals to meet the criteria, which the virus uses when deciding 'to infect or not to infect' the victim file. You can enter the size of a sacrificial goat file in any given format: decimal, hexadecimal or in kilobytes. Size of the victim files can be as small as 2 bytes and as large as many gigabytes (it is stored in 32-bit variable). GOAT.COM is very flexible - it can create COM, EXE, SYS(COM) and SYS(EXE) files, with code at the beginning, in the middle, or at the very end of the goat file. Files can be filled with zeroes, NOPs, two types of pattern and even filled with random garbage. You can add stack segment for the EXE files, vary header size, and ... many other options are available. The GOATS.COM file is intended to create a series of bait files with linearly increasing length. Length increase step is changeable. GOATS.COM has the same flexibility as GOAT.COM.

FLOCK.COM creates up to 1,000,000 identical files. You can infect them in a polymorphic virus to test its behaviour and properties. FLOCK.COM uses the same engine as GOAT.COM and GOATS.COM. Thus, the same flexibility as GOAT.COM is available too.

GOATSET.BAT produces some sort of 'a standard set' of files of the same size. These files are different (internal contents or attribute is variable). GOATSET.BAT needs GOAT.COM for execution. GOAT.COM should be located in the current directory accessible via PATH environment variable.

A small batch file RUN-ALL.BAT will help you to run (or infect, if you have a resident virus) all generated bait files.

4.1 SYNOPSIS AND SWITCHES

Usage of the main program - GOAT.COM looks like this (others are similar):

GOAT Size [Filename] [/switch] [/switch] ...

Size - decimal, hexadecimal, or in kbytes
(Example: 10000, 3E00h, FF00h, 31k, 512K, 2048k)
Filename - file to create. If no, makes GOAT000, GOAT001, ...

Short reference of all available switches is given below in alphabetical order:

/Annnn	set device Attribute (default=0C853h)
/B	place code at bottom of file (default - at start)
/C[n]	set selfcheck level (by default equal to 2, the highest) (/C means /C0; i.e., no selfchecking at all)
/Dnnn	create maximum 'nnn' subdirectories (default=10) (recognized only by FLOCK.COM, ignored by GOAT and GOATS)
/E	create EXE file (if size > 65280 - done automatically)

/Fnnn	create maximum 'nnn' files in a subdirectory (default=500) (recognized only by FLOCK.COM, ignored by GOAT and GOATS)
/H, /?	Help screen
/Inn	use fill byte 'nn' instead of standard zero-fill (ex., decimal /i100 or hexadecimal notation /iE5h)
/J	remove JMP at code start (default - JMP present)
/Knnnn	add 'nnnn' bytes of STACK segment to the bottom of EXE file (stack segment is filled with 'STACK' by default)
/Mnnnn	place code in the middle of the file exactly at nnnn position ('nnnn' is 32-bit value, but see limitations below)
/N[nnnn]	fill goat file with pseudorandom bytes. The parameter (if given) is a random number generator seed. RNG uses a multiplicative congruental method with 2**32 period
/O	do not make long EXE (>256K) with internal overlay structure
/P	fill free file space with pattern 00, 01, .. FE, FF, 00, ..
/R	make file ReadOnly (default - normal)
/S	make short (32 bytes) EXE header (default - 512 bytes)
/Tnn	set timestamp seconds field = nn (<63, even: 0, 1Eh, 62, ..)
/V	set SS:SP equal to CS:IP
/W	make word pattern (0000, 0001, ...FFFF, 0000)
/X	suppress signature defined in the INI file using "Motto="
/Y	create device driver (SYS file)
/Z	make 'ZM' EXE header instead of 'MZ'
/9	fill free file space with NOPs (default - with zeroes)

GOAT.COM, GOATS.COM and FLOCK.COM programs use the same set of command line switches. Most switches are self-explanatory.

The pattern inside the goat file always reflects the current offsets in the file (i.e., it is 'anchored' to the absolute location in the file). For example, at the file offset 1A2Bh you will see bytes '2B', '2C', '2D', ... (for byte pattern). Word pattern at the same location will look like this:- '2B', '1A', '2C', '1A', etc. Sometimes pattern filling is very useful.

Switch /Knnnn adds stack segment at the bottom of the EXE file. Size of the stack segment is limited - 16 < nnnn < 65536. Obviously, SP always points on the bottom of stack segment (i.e., SP=nnnn). Small and odd values in /K switch should be avoided, because they can hang the computer or cause 'Exception #13' (QEMM frequent warning), when SP goes through the stack segment boundary (i.e., half of a word is written at SS:0000 and other half— at SS:FFFF).

Switches /Fnnn and /Dnnn are recognized only by FLOCK.COM (GOAT.COM and GOATS.COM simply ignore them). You can specify the desired number of files and subdirectories to create. By default, 10 subdirectories with 500 files in each are created.

4.2 SIZE LIMITATIONS

By default GOAT.COM, GOATS.COM and FLOCK.COM programs produce a sacrificial file of COM type. This applies to any given size, which meets the following criterion:

$$2 < Size\_of\_COM < 65280$$

The magic number 65280 is a maximum size of COM file, which must fit in a segment size (64k=65536) without PSP size (256):

$$65536 - 256 = 65280$$

When placing the code at the bottom of the COM file, which size is around 64K, the code may lay too close to SS:SP (SS=CS for COM files; SP=FFFE) and the program may hang when run, because stack will likely overwrite the code. Therefore, if the spacing between IP and SP is less than 64 bytes, the goat generation is aborted and the output file is not created (you will see a warning, 'Goat IP will be too close to SP. Abort!').

When the size specified in the command line is greater than 65280 (or equal to), an EXE file is generated automatically (you do not need to write /E or /S switch explicitly). Such a file will have a normal 512-bytes EXE header in the beginning. When you need to create an EXE file shorter than 65280 bytes, use /E (or /S, /Z or /Knnnn) command line switch.

4.3 INI FILE

You may like to put your preferences (signature, switches, filename templates, etc.) into a separate file - GOAT.INI (common for GOAT.COM, GOATS.COM and FLOCK.COM). Use any text editor to create or modify anINI file. The sample GOAT.INI file is given below:

GOAT.INI

```
Motto="Anti-virus test file."    ;all output bait files will carry this string.
GOATfiles=FPROT                  ;files will be FPROT000.COM, FPROT001.COM, ...
                                 ;(default=GOAT)
GOATSfiles=ESASS                 ;files will be ESASS000.COM, ESASS001.COM, ...
                                 ;(default=GOAT)
FLOCKfiles=S&S                   ;files will be S&S000.COM, S&S001.COM, ...
                                 ;(default=GOAT)
FLOCKdirs=HEAP                   ;directories created - HEAP000, HEAP001, HEAP002
                                 ;(default=DIR)
STACKfill="*MYSTACK"             ;fill stack with '*MYSTACK*MYSTACK*MYSTACK'
                                 ;(default=STACK)
SYSname="DRIVERXX"               ;this string is inserted into SYS header
                                 ;(default=GOATXXXX)
Switches=/F200/D50               ;make 50 dirs, 200 files in each. 10000 in total
Switches=/C1                     ;to turn off registers check and avoid
                                 ;warning "Your PC might be infected..."
Switches=/iF6h                   ;always fill free file space with 0F6h byte
Switches=/O                      ;never make overlaid EXE files
```

GOAT.INI may be located in the current directory or in the path of the started program. The first location has priority over the second. GOAT.INI may not exist. In that case programs use built-in defaults.

Filename and subdirectory templates are limited to 5 symbols, because programs always add '000' and then start incrementing this number until it becomes '999'. Any string exceeding the limit of 5 symbols will result in the following error message:

'Error in the INI file line #nnn'

4.4 BAIT FILE INTERNALS

The bait files created with GOAT.COM, GOATS.COM and FLOCK.COM (if they are the same size) are absolutely identical in their internal structure and properties.

A created sacrificial goat file contains a small program, which displays its type (COM, EXE or SYS), size in hexadecimal and in decimal (only when the goat file is of big enough, i.e., space for code itself is at least 70 bytes). Sacrificial goat files consist of the two parts: the small portion of code (70 bytes or, if space not allows, just 2 bytes) and a block of zeroes, NOPs or pattern of variable size (00..FF, 0000...FFFE or random pattern). Zeroes (or NOPs or pattern) take all space of the file, free from the code. EXE files have additionally an EXE-header. The non-used part of the EXE header is always filled with zeroes. SYS files have additionally a device header, strategy and interrupt routines.

The output of a sample goat file (the size of the sample was 100 bytes) is the following:

'Goat file (COM). Size=00000064h/0000000100d bytes.'

File type (COM/EXE/SYS) and real numbers are inserted into the goat file message at the moment of creation.

4.5 NAMING OF GOATS

Usually GOAT.COM, GOATS.COM and FLOCK.COM programs create output sacrificial files in the following order: GOAT000.COM, GOAT001.COM, GOAT002.COM, etc. The same applies to EXE files: GOAT000.EXE, GOAT001.EXE, GOAT002.EXE, etc. If some file in a row (say GOAT050.COM or GOAT050.EXE) already exists - the next file number is selected automatically (it will be GOAT051.COM or GOAT051.EXE). Thus, we cannot generate both GOAT050.COM and GOAT050.EXE in the same directory. This rule does not apply to SYS files (eg., GOAT000.COM and GOAT000.SYS are allowed). This naming strategy is used to give some freedom for companion viruses.

Note, that definitions, given in the INI file may change the default file (and subdirectory) naming.

4.6 BAIT DEVICE DRIVERS

There are two formats of DOS device drivers - the old format (à la COM, understood by all DOS versions >2.0) and new format (à la EXE, introduced in MS-DOS 5.0). Drivers of the old type can only be started from CONFIG.SYS using a DEVICE statement. The entry point is defined in a special SYS header. Drivers of the new (EXE) type can additionally be started as a normal executables from the DOS command prompt. Drivers of the EXE type have two entry points - one for invocation from CONFIG.SYS/DEVICE (as written in the SYS header, which goes after EXE header) and the other is defined by CS:IP fields in the EXE header (this one works only when the file is started from the command line). The other advantage of the EXE format driver - it is not limited to 64K, like the old type of drivers. The first example of a driver with EXE format was SETVER.EXE from MS-DOS v.5.0. Such drivers can exceed 64K, but pointers to Strategy and Interrupt routines must fit into the first 64k (they are limited to 16-bits).

To create a device driver (SYS) file use switch /Y. Goat drivers of the old (COM) style will print the message 'Goat file (SYS). Size=...' when DOS requests an initialization of the driver (during CONFIG.SYS processing). Files in the new format (SYS&EXE) will do the same, but will print this message also when run from the DOS command line as a normal EXE file. In both cases, this driver file prints the same message. Note, that the EXE device drivers bear a '(SYS)' designator inside, but are always named as EXE files (to enable start from the command line as a normal executable).

Minimal size of the device driver is around 150 bytes (including SYS header). This limit increases for SYS & EXE files (it should include additionally the size of the EXE header - 32 bytes for /S; 512 bytes for /E).

5 'A STANDARD SET' OF GOAT FILES

Let's imagine that we know that we have a sample of the virus (eg., we got the sample from a knowledgeable anti-virus researcher), but we have no information about the properties of the virus. This situation frequently occurs in practice. First, we test it against a set of files of different lengths (say, 1000, 2000, ...10000 bytes). Now we see that the virus infected 8 files (3000, ...10000) and conclude that the virus avoids short victims (<3000). The 'standard set' of goat files may help you to find out which files are preferred by the virus (eg., the virus may infect only COM files starting with JMP). Checking 'a standard set' after virus attack, you can easily understand which files are infectable.

Now we have another question. Does the virus infect all files longer than 3000 bytes regardless of their contents? We have to test the virus against a set of files of fixed size, but with different contents. To simplify this task, the GOAT package has the generator of 'a standard set' of baits of given size it is called:- GOATSET.BAT. Yes, this file is really a DOS batch file, issuing a series of calls to GOAT.COM with different parameters. GOATSET.BAT makes COM, EXE and SYS files. Files are filled with zeroes or NOPs (90h), with the initial JMP (0E9h) or without it. Some files carry the ReadOnly attribute. EXE files are with normal (512 bytes) and short (32 bytes) EXE headers, with MZ and ZM markers.

GOATSET.BAT needs only one command line parameter - size of the files in the set. After invocation 52 files of the same size are generated - 12 COM, 34 EXE, 2 SYS and 4 SYS&EXE files. GOATSET.BAT also writes a report file GOATSET.LOG and places a full description of the generated bait files set there.

Being a BAT file, GOATSET.BAT is fully customizable. It can be easily changed with any text editor.

6 FUTURE THREATS

6.1 ANTI-GOAT VIRUSES

Fortunately, there are only a few viruses that try to avoid infecting goat files. One of them is *Sarov.1400*. It uses primitive algorithm to avoid victims with many repeated bytes.

The corresponding code is:

```
0100 8B161C00  MOV    DX,[001C]       ;LOAD RELATIVE OFFSET IN FILE
0104 33C9      XOR    CX,CX
0106 D1EA      SHR    DX,1
0108 B80042    MOV    AX,4200         ;LSEEK TO CHECKED FILE AREA
010B E80F01    INT    21
010E BAD804    MOV    DX,04D8         ;BUFFER LOCATION
0111 B43F      MOV    AH,3F           ;READ 100 BYTES FROM FILE
0113 B96400    MOV    CX,0064         ;SIZE OF BLOCK TO CHECK
0116 8BFA      MOV    DI,DX           ;DI -> BUFFER
0118 CD21      INT    21
011A 268A05    MOV    AL,ES:[DI]      ;GET FIRST BYTE (ES=DS)
011D 47        INC    DI              ;SKIP TO NEXT BYTE
011E F3AE      REPZ   SCASB           ;COMPARE WITH THE FIRST
0120 7455      JZ     DON'T_INFECT    ; ALL BYTES ARE THE SAME!
INFECT_THE_FILE:      ...
```

Without any doubt, more and more anti-goat viruses will appear in the future. We can also expect the appearance of more viruses which avoid victims placed on virtual disk. Or viruses, which do not infect files with certain typical lengths (divisible with 10**N and 16**N). Fortunately, most virus writers have not yet realized that such features are a very strong weapon. I would say, comparable with polymorphicity, because in most cases full disassembly of the virus will be required and that takes time. Moreover, such anti-goat tricks are programmed much more easily than any polymorphic engine.

6.2 ARMORING TRICKS, VIRUS/TROJAN CONVERSION

There are a lot of viruses which try to complicate their investigation. Viruses use anti-tracing techniques: *SVC.4644, Ieronim, XPEH* (family of viruses), *Zherkov* (called also *Loz*), *Magnitogorsk, HideNowt, OneHalf.3544, OneHalf.3577, Cornucopia,* etc. A wonderful set of antitracing capabilities is found in *Compact Polymorphic Engine* (CPE 0.11b), which is actually a virus creation tool.

Some viruses, when they detect that they are being traced switch to the 'trojan' mode and try to damage files, floppies and/or hard disks. That looks like revenge by the virus writer for an anti-virus researcher s attempts to catch the virus. Many viruses have such behaviour - for example, recently found *RDA.Fighter.5871/5969/7408* overwrites random sectors on the HDD [Daniloff], whereas *Maltese Amoeba* destroys 4 sectors on each of the first 30 cylinders of all drives; *CLME.Ming.1952* overwrites 34 first sectors on all drives; *DR&ET.1710* erases 128 first sectors on HDDs; *Gambler.288* destroys first 10 sectors on drive C:; Kotlas removes original non-infected copy of MBR, *SumCMOS.6000* tries to corrupt HDD.

The most nasty idea is to use destructive capabilities (à la trojan) if the virus senses the anti-virus environment. For example, when a virus detects goat files.

REFERENCES

[Skulason]	Fridrik Skulason 'The Virus Glut. The Impact of the Virus Flood.', *Proceedings of EICAR'94 Annual Conference,* St.Albans, 1994, pp.143-147
[FAQ]	VIRUS-L FAQ, Available from ftp.informatik.uni-hamburg.de in pub/virus/texts/ viruses/v-l-faq.zip or also from 'comp.virus' newsgroup.
[Brown]	Matt Brown 'N8FALL: The Nightmare Bug', *Virus Bulletin,* May 1995, p.11
[Kaspersky]	Eugene Kaspersky 'Hiding in EMS — A Creative Crisis?', *Virus Bulletin,* January 1995, pp.8-9
[TOP]	'The 1995 Scanner Top Ten', *Virus Bulletin,* January 1995, pp.14-19
[Jackson]	Keith Jackson 'S&S: The Anti-Virus Toolkit', *Virus Bulletin,* May 1995, p.22
[GOAT]	Available from ftp.informatik.uni-hamburg.de in pub/virus/progs/goat30.zip
[Daniloff]	Igor Daniloff, Documentation to Dr.Web anti-virus scanner.

TABLES

Table 1. **A list of viruses avoiding infection of some files (usually, anti-virus programs)**

Virus name(s)	Avoids
2UP.6000	AID*, COMMAND*, ANTI*, AV*, HOOK*, SOS*, TSAFE*, -V*, SCAN*, NC*, VC*, TNT*, ADINF*
Armagedon	*ND.COM
Bastard.1979	SC*, CL*, F-*, TB*, VS*, VI*, IM*, MS*, CV* (also deletes MS*.* and CP*.*)
ChDir.523	C*.*
CLME.Ming.1952	*AN.*, *OT.*, *86.*, *PY.*, *EG.*

Cornucopia	PROT*, SCAN*, VIRU*, NAV*
Cpw	CNC*, GUARD*, EMS*, CPAV*, SCAN*, CLEAN*, FINDVIRU*, CHKVIRUS*
Cruncher	CO*, SC*, CL*, VS*, NE*, HT*, TB*, VI*, FI*, GI*, RA*, FE*, MT*, BR*, IM*
Daemaen.1894 (Talon)	SC*, CL*, VS*, F-*
Datacrime	*D.COM
DR&ET.1710	SC*, CL*, VS*, F-*, CP*, VI*
Erase-821	*AID*, *VIR*, *DINF*, *CHK*, *TEST*, *AUR*, *PAV*, *NAV*, *-V*, *SENT*, *ASM*, *SCAN*, *LEAN*, *ANT*, *SAFE*, *BOOT*, *STRA* (and deletes these files)
Grog.2075	*MBIO*, *MDOS*, *SCAN*, *LEAN*, *PROT*, *CPAV* (and deletes CHKLIST.CPS)
Grog.2825	IBMBIO*, IBMDOS*, SCAN*, CLEAN*, F-PROT*, CPAV*, MSAV*, NAV* (and deletes ANTI-VIR.DAT, CHKLIST.*)
Helloween-1376	SCAN.*, CLEAN.*
Jerusalem-113b	PHENOME.COM
Jerusalem-Nemesis	NEMESIS.COM
Metal-500	*ST.*
Migram-Smack	*ND.*, *AN.*, *HA.*, *HK.*
Migram-Cemetry	*ND.*
MPTI-1536	AIDSTEST.*, VL.*
MrD.1569	VIR*, *MKS*, *AV*, *NV*, *TB*,
PlayGame.2000	CO*, 4D*, SC*, CL*, VS*, NE*, HT*, TB*, VI*, F-*, FI*, GI*, IM*, RA*, FE*, MT*, BR*
Predator2448	*PROT*, *SCAN*, *CLEA*, *VSAF*, *CPAV*, *NAV.*, *DECO*
PS-MPC.Joshua	*ND.COM
RDA.Fighter	*ES?, *WE?, *AN?
Sh.983/988	AI*, WP*, HI*, DO*, KR*
Slovakia.1.0/1.	SCAN*, AVG*, VIR*, ASTA*, ALIK*, REX*, MSAV*, CPAV*, NOD*, CLEAN*, F-PROT*, TBAV*, TBUTIL*, AVAST*, NAV*, VSHIELD*, VSAFE*, DIZZ*
Squeaker	SCAN*.*, ??VU*.*
SumCMOS.6000	COMMAND.COM, GDI.EXE, DOSX.EXE, WIN386.EXE, KRNL286.EXE, KRNL386.EXE, USER.EXE, WSWAP.EXE,CHKDSK.EXE
Sverdlov-1064	AI*.*, SC*.*
Swami (Bhaktivedanta)	*ND.COM, *AN, *LD, *RJ
Thirteen Minutes (also called Thursday-12th)	COMMAND.COM, SCAN*, CLEAN*, VIR*, ARJ*, FLU*
Tired-1740	Has a checksum list of 16 programs (checksum includes name and contents)
TPE.Bosnia	CO*, SC*, CL*, VS*, NE*, MS*, TB*, VI*, FI*, F-*, IM*, CP*, NA*, -V*, RA*, FE*, MT*, BR*
TPE.Girafe	CO*, SC*, CL*, VS*, NE*, HT*, TB*, VI*,FI*, GI*, RA*, FE*, MT*, BR*, IM*

Union.1449	AI*, AD*, SC*, NC*
Uruguay-#6/#7/#8	COMMAND.COM, SC*.*, F-*.*, *V*
XPEH-4016	COMMAND.COM, AIDSTEST.*

Note: SC = SCAN, CL=CLEAN, TB=TBSCAN, F-=F-PROT, CP=CPAV, AI=AIDSTEST, VS=VSIELD/VSAFE, NA=NAV, FI=FINDVIRU, AV=AVSCAN/AVP/AVPRO, AD=ADINF, BR=BRM_SCAN, 4D=4DOS.

Table 2. Some unusual virus markers

Virus	**Infection marker**
Atas.1268	Minutes mod 8=0, seconds=34 in file timestamp
Com2con	Timestamp: time=11:19
Kela.2099	62s in file timestamp, JMP, 'KLM' at start
Kuku.448	20s in file timestamp
Taiwan.708	62s in file timestamp
Uruguay-#1	Filesize mod 13h=0
Uruguay-#2/#3/#4	mod 17h=0
Filedate 11 (V-537)	Timestamp: date=17.08.88, time=02:08:34
Vienna (DOS-62)	62s in file timestamp
Vienna.377	Filestamp date (certain bits)

AUTOMATIC TESTING OF MEMORY RESIDENT ANTI-VIRUS SOFTWARE

Dr David Aubrey-Jones

Reflex Magnetics Ltd, Units 1-2, 31-33 Priory Park Road, London NW6 7UP, UK
Tel +44 171 372 6666 · Fax +44 171 372 2507

ABSTRACT

Testing of Anti-Virus products has with very few exceptions been limited to static scanning, and has excluded testing of the memory-resident component programs, the parts on which many rely for their virus protection. The reason for the exclusion is simple: such testing is normally extremely time consuming.

A simple method for automating such testing has therefore been devised. Such a method is likely to prove useful to AV companies in quality testing their software. It should also help pinpoint security holes within protection provided by TSR scanners and behaviour monitors.

INTRODUCTION

A number of different techniques have been developed to combat the growing threat of computer viruses. However, in reviews and product tests only a small part of the whole product is commonly tested against viruses; namely, the non-resident scanner. Such tests, while undoubtedly useful, leave an important part of the products totally untested. To make matters worse, these are commonly the very parts on which users rely for virus protection.

The reason for this exclusion is simple. Such testing is extremely time consuming, and would be a logistical nightmare if performed in a large-scale test. In terms of published reviews, very few have included such tests.

POOR PERFORMANCE FOUND IN TESTS

In September 1993, Virus Bulletin conducted a comparative review of six memory-resident scanners. They tested the ability of these products to detect 83 'In the Wild' virus files, and 250 other virus-infected files as they were copied using the DOS copy command.

Interesting findings resulted. Only one product reviewed demonstrated equal performance between the non-resident scanner and the memory-resident version, with the memory-resident version in most cases trailing significantly behind. The review concluded 'The poor performance of all the memory-resident scanners came as something of a shock when conducting this review'. It is obvious from these results that it is not safe to extrapolate from the results with the non-resident scanners to their memory-resident counterparts.

Since this time, no such tests appear to have been published elsewhere. *Secure Computing* made an attempt in February of this year to look at memory-resident scanners, but limited themselves to observations in memory usage and the performance impact of copying files (normally very significant). They shied away from the really interesting tests; the tests against real viruses.

There are several methods which have been developed to protect against viruses: virus-specific scanning software, behaviour monitors/blockers and integrity checkers. All rely on being memory resident, and the ability to monitor various aspects of the PC to provide constant protection. They also normally alert the user if something is amiss.

THE ULTIMATE TEST

To test any of these methods and their ability to provide virus protection adequately, they must be submitted to a real virus attack. This is the ultimate test of any anti-virus product. However, such tests are normally far too time consuming and tedious ever to be conducted. Even copying infected files, as was done in the *Virus Bulletin* tests, takes a lot of time. Copying files is also limited in scope, and is unable to test products fully. Only memory-resident scanners can be evaluated in this way, and then only in a limited way. It is interesting that *Virus Bulletin* has never conducted further tests along these lines.

With these factors in mind, it is readily apparent that there is a vital need for some automated method to test protection provided against virus attacks. Companies producing anti-virus software require such a method to test their products fully, and to provide adequate quality control. Such a method is also required to evaluate products properly and to provide meaningful results, such as under the European *ITSEC* scheme.

REQUIREMENTS FOR TESTING AV SOFTWARE

Let us examine some of the requirements for testing virus protection. To start, the test computer must be virus-free, and exact information on file and system integrity must be established, so that any changes produced by a virus can be detected.

Next, the anti-virus protection must be activated, and then the virus sample must be introduced under controlled conditions. Following this, an attempt must be made to activate the virus, by executing the infected sample. At this point the resident anti-virus software may or may not protect the system, and may produce an alert message.

Lastly, several other programs should be executed to act as virus bait, or goats. Memory-resident parasitic file viruses commonly infect on file execution, file open or file close when a suitable target file presents itself. Again the anti-virus software may provide protection, and may provide an alert message. All traces of the virus must now be removed from memory, by ensuring a clean boot. The method normally adopted for this is a cold boot from a DOS system floppy disk, with a watch being kept for CMOS-modifying viruses such as EXEBug. A full integrity check must follow to discover if the virus succeeded in infecting the computer, or in making any other unauthorised changes to the system.

Finally, and most importantly, the PC must be completely cleaned of the virus (if it was infected) and all files etc must be restored to their original state. Using the Cleanup utility from an anti-virus toolkit is obviously not adequate.

If this procedure is strictly followed, a fairly good knowledge of the protection offered by a product against a particular virus threat will result, although even this is by no means an exhaustive test. A person could probably accomplish in the order of four tests an hour using this method, when one takes into account fatigue, etc. It now becomes obvious why no product tests are generally conducted using such a method!

POTENTIAL PROBLEMS WITH AUTOMATED TESTS

ATOMS (Automated Testing Of Memory-resident Software) is a method we developed at *Reflex Magnetics* to automate such a test procedure. From the outset, it was clear that an adequate system could not be constructed from a single computer on its own without a large amount of extra hardware. It was far better to use two separate PCs, with various links between the two (see Fig. 1). One of the PCs would control the testing process (the Control PC), and the other would act as the dirty PC, with active viruses on it (the Test PC). With such an arrangement, the Control PC can remain in full control, since it can be kept totally free from viruses at all times.

CLEAN BOOTING

When constructing ATOMS, one of the first questions to be decided was how to clean-boot. This is a necessary requirement for accurate analysis of infected items, to avoid interference from virus stealth techniques, and to ensure cleanup after a virus. The most common method of clean-booting is via a system floppy disk, but this method was far from ideal for use by ATOMS, for a number of reasons. We therefore decided to clean boot from a network, using a boot image. This was faster and had the advantage that, by changing the boot image on the network, different modes of operation could readily be initiated.

HANGING

Another potential difficulty we had encountered was the propensity of many virus samples to hang a computer. Similarly, considerable numbers of so-called virus samples often turn out to not be viruses at all, but Trojan horses, which immediately cause damage, resulting in a hang when executed. Obviously, if the Test PC hangs during a test cycle, the Control PC must have a method to sense this and force a reset. Otherwise, when testing a number of virus samples, the process would be very likely to stall part way through, requiring operator input, and would not be fully automated.

A system such as ATOMS could be constructed using memory-resident code of one form or another to monitor the infection process and log or report results. Reliance on such code would be a mistake, however, in a DOS system environment, due to the ease with which viruses could cause interference.

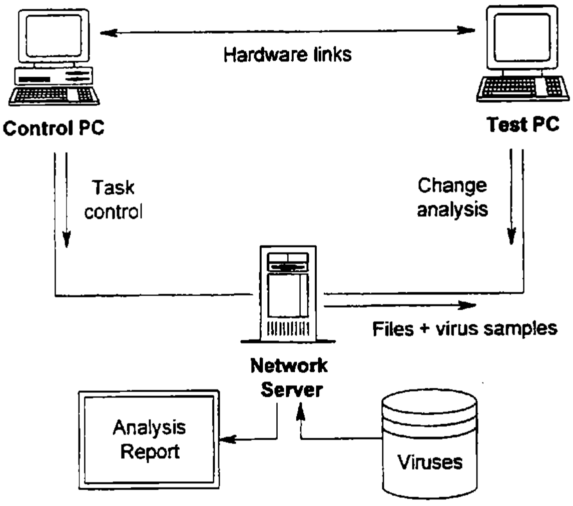


Fig. 1 ATOMS component links

ATOMS DESIGN AND OPERATION

ATOMS uses a Novell network to link the Control PC to the test PC, plus other proprietary Hardware links (Fig.1). All virus samples are stored on the network drive in a directory with restricted access. Each sample is selected for testing in sequence according to a list which is accessed sequentially from a database.

TEST PC SETUP

The test PC is normally configured as a basic DOS PC. MSDOS 3.3 was selected as being most 'virus friendly', there being some viruses which are very fussy about which version of DOS they will function on. An additional run using other versions of DOS can be used for such awkward viruses.

A Goat directory containing a number of different Goat files of differing lengths, etc is used on the test PC, and the virus sample is inserted after these. There is also a standard DOS directory containing a full complement of DOS files.

INITIALISATION

The testing procedure starts by the Control computer clean-booting the test computer (see Fig.2). The test files and programs including DOS, etc are loaded onto the test PC, and a copy is made for comparison later of various tracks on the Hard Disk which contain the MBR and partition boot sectors, etc. The main loop which is performed for each virus in turn is now begun. This consists of three main phases: the Infection phase, the Analysis phase and the Cleanup phase.

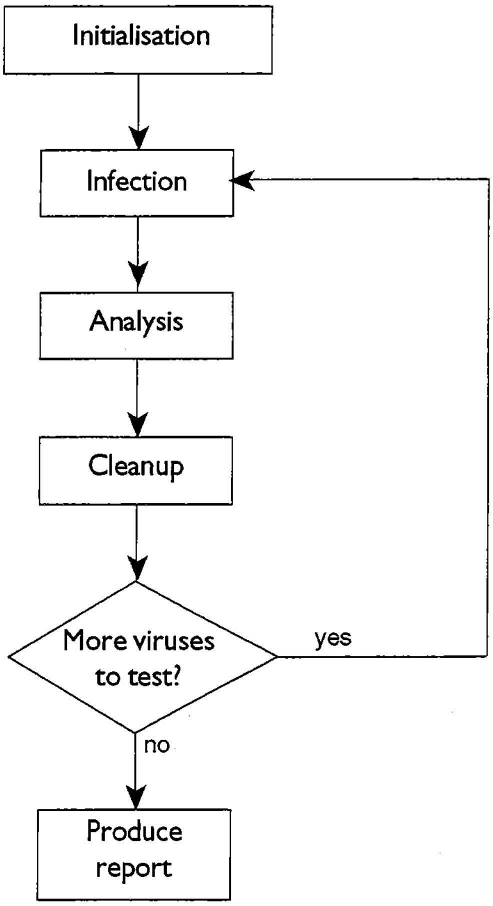


Fig. 2. Main stages in ATOMS testing

INFECTION

The Infection phase is begun by booting the test PC from a clean boot image. Included within this are the files CONFIG.SYS and AUTOEXEC.BAT which first execute the anti-virus TSR to be tested. The virus is then executed, followed by a series of goat files.

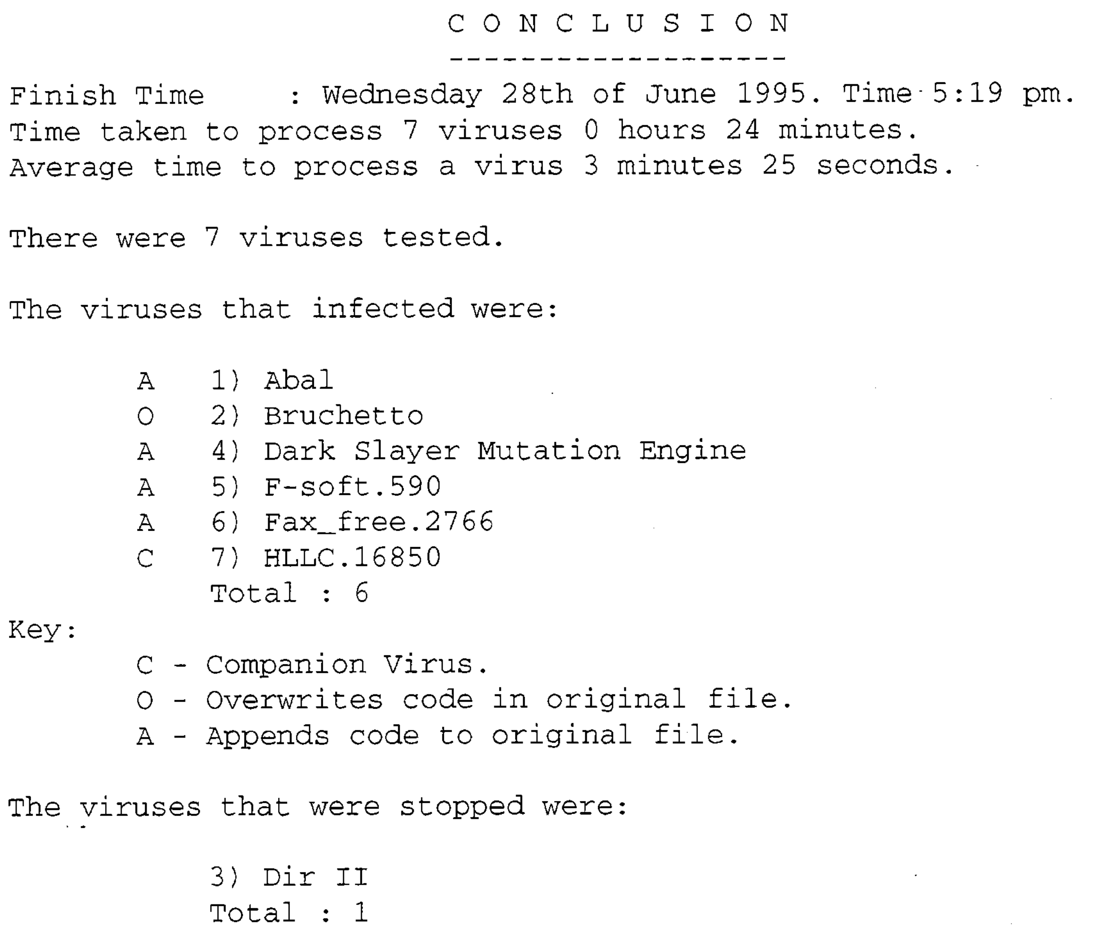
ANALYSIS

When all the Goat files have been executed, the test PC is rebooted, and the Analysis phase begins. A full comparison check is made for changes, and any found are logged to the report file.

```
ATOMS - Automatic Testing Of Memory-resident Software
- - - - - - - - - - - - - - - - - - - - - - - - - - -
Copyright (C) 1995 Reflex Magnetics Limited. Designed by David
Aubrey-Jones, written by Andrew Oaten and David Aubrey-Jones

Report Started on Wednesday 28th of June 1995. Time 4:55 pm.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Virus     : Abal.
Filename  : AL758.COM.
Number    : 1.
Changed Files:
       C:\TARGET\COMIT.COM
       C:\TARGET\A1000.COM
       C:\TARGET\A50000.COM
       C:\TARGET\A10000.COM
       C:\DOS\FORMAT.COM
       C:\DOS\MODE.COM
       C:\DOS\SELECT.COM
       C:\DOS\SYS.COM
       C:\DOS\ASSIGN.COM
       C:\DOS\BACKUP.COM
       C:\DOS\CHKDSK.COM
       C:\DOS\COMP.COM
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Virus     : Bruchetto.
Filename  : BRUCETTO.COM.
Number    : 2.
Changed Files:
       C:\TARGET\COMIT.COM
       C:\TARGET\A1000.COM
       C:\TARGET\A50000.COM
       C:\TARGET\A10000.COM
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Virus     : Dir II.
Filename  : DIR2M.COM.
Number    : 3.
No changes detected.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
Virus     : Dark Slayer Mutation Engine virus.
Filename  : DSME.COM.
Number    : 4.
Changed Files:
       C:\TARGET\WAIT.EXE
       C:\TARGET\COMIT.COM
       C:\TARGET\A1000.COM
       C:\TARGET\B1000.EXE
       C:\TARGET\B10000.EXE
       C:\TARGET\B50000.EXE
       C:\TARGET\A10000.COM
       C:\TARGET\A200.COM
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Virus     : F-soft.590.
Filename  : F-SOFT59.COM.
Number    : 5.
Changed Files:
       C:\TARGET\COMIT.COM
       C:\TARGET\A50000.COM
       C:\TARGET\A10000.COM
       C:\TARGET\AV1.DAT
       C:\TARGET\AV2.DAT
       C:\TARGET\AV3.DAT
       C:\TARGET\AV4.DAT
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Virus     : Fax_free.2766.
Filename  : FAM.EXE.
Number    : 6.
Changed Files:
       C:\TARGET\WAIT.EXE
       C:\TARGET\B1000.EXE
       C:\TARGET\B10000.EXE
       C:\TARGET\B50000.EXE
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Virus     : HLLC.16850.
Filename  : WORM.COM.
Number    : 7.
New files:
       C:\B1000.COM
       C:\TARGET\AV1.COM
       C:\TARGET\AV2.COM
       C:\TARGET\AV3.COM
       C:\TARGET\AV4.COM
       C:\TARGET\AV5.COM
       C:\TARGET\AV6.COM
       C:\TARGET\B50000.COM
       C:\TARGET\B10000.COM
       C:\TARGET\B1000.COM
       C:\TARGET\WAIT.COM
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```



```
                              C O N C L U S I O N
                              -------------------
Finish Time     : Wednesday 28th of June 1995. Time 5:19 pm.
Time taken to process 7 viruses 0 hours 24 minutes.
Average time to process a virus 3 minutes 25 seconds.

There were 7 viruses tested.

The viruses that infected were:

       A    1) Abal
       O    2) Bruchetto
       A    4) Dark Slayer Mutation Engine
       A    5) F-soft.590
       A    6) Fax_free.2766
       C    7) HLLC.16850
            Total : 6
Key:
       C - Companion Virus.
       O - Overwrites code in original file.
       A - Appends code to original file.

The viruses that were stopped were:

            3) Dir II
            Total : 1
```

Fig.3 Sample report produced by ATOMS

CLEANUP

Lastly, the Cleanup phase commences. The hard disk is wiped clean and formatted, and fresh copies of all files required are then copied onto the test PC from the network. Finally, to complete the cleanup, the next virus sample file specified in the database is copied into the directory with the Goat files, and the Infection phase begins again.

Based on a 486 DX2 test PC ATOMS is capable of performing around 30 tests per hour. In a day it can therefore manage nearly 800 virus samples. However, memory-resident scanners slow file access times considerably, and ATOMS performance when testing these products can be greatly reduced.

FINDINGS

Sections from a typical report produced by ATOMS are shown in Fig.3. Details on each virus which succeeded in bypassing the memory-resident protection is produced. This includes information on changes to hard disk boot sectors, and all files.

At the end of the report, there is a summary noting the time taken for the tests, the number of virus samples tested, and the number which produced a change on the test PC. A list of all viruses which produced a change is given, with a note on the type.

LIMITATIONS OF ATOMS

One must be careful when conducting any kind of anti-virus product test, and in this sense ATOMS is no different. There are several potential problems. One of the most important of these is the library of viruses which are used. Obviously, all these should be true viruses, but many viruses are very fussy about the conditions under which they will function, and must be 'spoon fed'. DOS version, BIOS type, disk type, type of Goat files, et. can all prove very important. Such viruses may well never infect on a given PC. Slow viruses, although uncommon, may also not infect. For this reason, all the viruses used in our final tests were first checked to make sure that they were functioning under the ATOMS setup. It should be remembered that if a change is found after a test with a given virus sample, it does not necessarily mean that a true infection has resulted. Changes could be caused by payload activation, etc and may not always correspond to an infection in the true sense of the word.

ATOMS has not been designed for use with boot sector viruses and cannot at present be used to perform any tests with them. This is something to be addressed in the future.

HUGE HOLES

Several memory-resident anti-virus programs have so far been tested with interesting results. These support the findings of *Virus Bulletin* two years ago, which showed that memory-resident scanners in general have a far poorer performance than their non-resident counterparts. In some cases the differences are striking, with huge holes in their security. For example, very few memory-resident scanners can detect many polymorphic viruses, which are becoming more and more common. By contrast, the performance achieved by behaviour blockers was generally superior.

THE FUTURE OF ATOMS

After initial teething problems, ATOMS soon proved invaluable. It is now possible to take a library of several thousand different viruses, and, with minimal operator input, check a memory-resident anti-virus product for any weaknesses. This is invaluable for good quality control.

It would also be very suitable for adoption as a tool to assist with evaluation of memory-resident anti-virus products. It enables tests with a meaningful number of viruses to be conducted which would otherwise be impossible, and reduces the cost of those tests. In addition, it has the advantage of being reproducible.

LATE SUBMISSION

The following papers are a late addition to the proceedings and, therefore, appear out of sequence.

COMPUTER VIRUSES IN HETEROGENEOUS UNIX NETWORKS

Peter V. Radatti

CyberSoft, Inc., 1508 Butler Pike, Conshohocken, PA 19428, USA
Tel +1 610 825 4748 · Fax +1 610 825 6785 · Email radatti@cyber.com

1 ABSTRACT

Unix systems are as susceptible to hostile software attacks as any other system, however; the Unix community is zealous in its belief that it is immune. This belief is in the face of historical reality. The first computer viruses created were on Unix systems. The Internet Worm, Trojan horses and logic bombs are all ignored milestones in this belief. Not withstanding these beliefs, there is a growing concern among computer security professionals about these problems: this is based on recognition of the complex nature of the problem and the increasing value of Unix based networks. Whereas the Internet Worm disrupted the Internet in 1988 the cost was relativity low; if this attack were repeated today, the cost will be very high because of the new-found importance of the Internet, electronic business networks using EDI, and private networks, all of which are Unix-based.

Traditional methods used against attacks in other operating system environments such as MS-DOS are insufficient in the more complex environment provided by Unix. Additionally, Unix provides a special and significant problem in this regard due to its open and heterogeneous nature. These problems are expected to become both more common and more pronounced as 32-bit multi-tasking network operating systems such as Microsoft NT become popular. Therefore, the problems experienced today are good indicators of the problems and the solutions which will be experienced in the future, no matter which operating system becomes predominate.

2 THE EXISTENCE OF THE PROBLEM AND ITS NATURE

The problem of software attacks exists in all operating systems. These attacks follow different forms according to the function of the attack. In general, all forms of attack contain a method of self-preservation, which may be propagation or migration, and a payload. The most common method of self-preservation in Unix is obscurity. If the program has an obscure name or storage location, then it may avoid detection until after its payload has had the opportunity to execute. Computer worms preserve themselves by migration, while computer viruses use propagation. Trojan horses, logic bombs and time bombs protect themselves by obscurity.

While the hostile algorithms which have captured the general public's imagination are viruses and worms, the more common direct problem on Unix systems are Trojan horses and time bombs. A Trojan horse is a program which appears to be something it is not. An example of a Trojan horse is a program that appears to be a calculator or other useful utility which has a hidden payload of inserting a back door onto its host system. A simple Trojan horse can be created by modifying any source code with the addition of a payload. One of the most favorite payloads observed in the wild is '/bin/rm -rf / >/dev/null 2>&1'. This payload will attempt to remove all accessible files on the system as a background process with all messages redirected to waste disposal. Since system security is lax at many sites, there are normally thousands of files with permission bit settings of octal 777. All files on the system with this permission setting will be removed by this attack. Additionally, all files owned by the user, their group, or anyone else on the system whose files are write-accessible to the user will be removed. This payload is not limited to use by Trojan horses, but can be utilized by any form of attack. Typically, a time bomb can be created by using the 'cron' or 'at' utilities of the Unix system to execute this command directly at the specified time.

While the 'bin remove' payload is a favorite of many authors, there are other traditional attacks which are not as overt in their destruction. These other attacks are more important because they bend the operation of the system to the purposes of the attacker while not revealing themselves to the system operator. Attacks of this form include the appending of an account record to the password file, copying the password file to an off-site email address for leisurely cracking and modification of the operating system to include back doors or cause the transfer of money or property. It is extremely simple to email valuable information off site in such a manner as to insure that the recipient cannot be traced or located. Some of these methods are path dependent; however, the path selected is at the discretion of the attacker.

One of the most simple methods of inserting a back door is the well known 'sticky bit shell' attack. In this attack, a Trojanized program is used to copy a shell program to an accessible directory. The shell program is then set with permission bits that allow it to execute with the user ID and permission of its creator. A simple one line sticky bit shell attack can be created by adding the following command to a user's '.login' or any other file they execute. Example: cp /bin/sh /tmp/gotu ; chmod 4777 /tmp/gotu.

Trojan horses and time bombs can be located using the same methods required to locate viruses in the Unix environment. There are many technical reasons why these forms of attack are not desirable, the foremost being their immobility. A virus or worm attack is more important because these programs are mobile and can integrate themselves into the operating system. Of these two forms of attack, the virus attack is the hardest to detect and has the best chance of survival. Worms can be seen in the system process tables and eliminated, since they exist as individual processes, while virus attacks are protected from this form of detection by their host programs. All of the methods used to detect and prevent viruses are also effective against the other forms of attack; therefore, the remainder of this paper will deal with the more serious problem of viral attacks.

3 UNIX VIRUS ATTACKS

The promotion of the concept of 'magical immunity' to computer viral attacks surfaces on a regular basis. This concept, while desirable, is misleading and dangerous since it tends to mask a real threat. Opponents of the possibility of viral attacks in Unix state that hardware instructions and operating system concepts such as supervisor mode or permission settings, security ratings like C2 or B1 provide protection. These ideas have been proven wrong by real life. The use of supervisor mode, the additional levels of protection provided by C2 and the mandatory access control provided by security level B1 are not necessary for viral activity and are therefore moot as a method of protection. This fact is supported by the existence of viruses which infect Unix systems as both scripts and binary.

In fact, virus attacks against Unix systems will eventually become more popular as simpler forms of attack become obsolete. Computer viruses have significantly more virility, methods of protection and opportunity for infection. Methods of protection have been highly refined in viruses, including rapid reproduction by infection, migration though evaluation of its environment, (boot viruses look for uninfected floppy diskettes), armor, stealth and polymorphism. In addition, the host system itself becomes a method of protection and propagation. Virus-infected files are protected just as much by the operating system as are non-infected files. The introduction of viruses into systems has also been refined using so-called 'droppers'. A dropper is a Trojan horse that has a virus or viruses as a payload. Finally, extensive networking technology such as NFS (Network File System) allows viruses to migrate between systems without effort.

All of these reasons point to viruses as the future of hostile algorithms; however, the most significant reason for this determination is the effectiveness of the virus as a form of attack. Past experiments by Doctor Fred Cohen [1984] used a normal user account on a Unix system, without privileged access, and gained total security penetration in 30 minutes. Doctor Cohen repeated these results on many versions of Unix, including AT&T Secure Unix and over 20 commercial implementations of Unix. The results have been confirmed by independent researchers worldwide. Separate experiments by Tom Duff [1989] demonstrated the tenacity of Unix viruses even in the face of disinfectors. The virus used in Mr. Duff's experiment was a simple virus written in script. It was believed to have been reintroduced by the operating system from the automated backup and restore system. Reinfection took place after the system had been virus-free for one year.

4 HETEROGENEOUS VIRUS ATTACKS

I have observed non-Unix personal computers attached to a heterogeneous network which were infected with computer viruses originating from Unix servers [1987]. The Unix systems were not the original point of entry for the viruses. They were dormant while on the Unix systems but became harmful when they migrated to their target systems. The Unix systems acted as unaffected carriers of computer viruses for other platforms. For the sake of simplicity, I have named this effect after an historical medical problem of similar nature, 'Typhoid Mary Syndrome' [1991]. Networks and specifically Unix servers which provide network file systems are very susceptible to this problem. I first observed this problem while investigating an infection of personal computers attached to a network with a large population of Unix servers and workstations. The virus was manually attacked on the personal computers using virus scanners. During the infection period, all of the personal computers were disconnected from the network and idle. Once all the computers were disinfected, all removable media was tested and the infection was unobserved for a period of time, the computers were reattached to the network. A few weeks later, a test of the computers using the same virus scanner indicated they had become reinfected with the same viruses. The source of infection was then identified as repositories of executables stored on the Unix file servers.

These repositories were organically grown centralized resources for all the personal computers because the Unix servers were effective at providing these shared services via NFS. In retrospect, this problem had to exist. The use of networked systems exported from the Unix platforms provided an easy, powerful method of transferring data, including executables. Some network designs provide all third party software from a network disk, for ease of maintenance and reduced storage requirements. This easy access provides an open door for viruses.

5 TRANSPLATFORM VIRUSES ATTACK UNIX

During late 1994 and early 1995, I observed multiple instances of at least three transplatform virus attacks on Unix systems. All of these attacks involved MS-DOS viruses which attacked PC-based Unix systems. The first attack involved a virus that corrupted the Unix file system every night. The attack was located using a virus scanner and indicated a Unix binary which was executed at midnight by 'cron'. The MS-DOS virus had become embedded in the Unix executable where it was executed. The virus did not perform as designed, in that the corruption was the result of the virus attempting to infect other files: it was not an intended effect. The virus was reinstalled every morning when the system was restored. The second attack involved an MS-DOS virus which executed, and was successful in infecting other files. Once again, the file system corrupted, but it took longer to do so, thereby allowing the virus to propagate. The final infection involved a boot sector virus. Since this type of virus executes prior to the loading of the operating system, the differences between Unix and MS-DOS are moot. The PC-BIOS and processor chips are the same in both cases, and the virus is able to execute according to design. In fact, two different viruses were observed performi ng in this way. The first virus was spread by an MS-DOS setup diskette, while the second virus was transmitted using a still undiscovered method. While we observed no boot sector infections of PC-based Unix systems during 1994, we received reports from system administrators who were requesting information on our Unix anti-virus product because they had experienced hundreds of infections during 1995. In one instance, a single multi-national company lost its entire international network overnight. The estimated cost in lost time, resources, and sales was in the millions of dollars.

Once it is understood that the BIOS and processor functions are the same for both operating systems, it is very easy to see how a transplatform virus could be designed by intention. The virus would be able to process correctly by inspecting the operating system using only common BIOS calls and then modify its basic behavior using a simple 'if' structure.

6 TRADITIONAL CATEGORIES OF PROTECTION AND THEIR FAILURE

There are three traditional categories of protection, none of which provide complete or significant protection as stand-alone methods of implementation. The categories are Control, Inspection and Integrity. Each of these methods has traditionally been used separately.

Control has been the primary intent of the U.S. national standards on computer security. They deal with the control of access to the system, its functions, resources and the ability to move or share data in the system. These national standards are codified in a library generally referred to as the Rainbow series (the name was given because the books have different color covers, making a library shelf look like a rainbow). While these standards are a valuable and important aspect of computer security, they do not provide a deterrent against software attack. A virus is an effective way of gaining control over a system, even a highly controlled system such as a B1-rated version of Unix. In this case, control does not provide protection against software attacks because of the viruses' ability to change permission sets with each new owner infected. A virus attack gains access to multiple users thought shared files. Access control is designed to allow the sharing of files. The ability to share files is a basic need of the user and cannot be eliminated without destroying the usefulness of the system. Discretionary Access Control (DAC) is not protection against software attacks, because it is a weak form of protection which can be bypassed and, as discretionary, is at the control of the end users who very often ignore it. Sites where the majority of the files on the system have no DAC protection are normal (many Unix sites have permission bit settings of 777, which allow anyone to read, write, execute or modify the file.) Mandatory Access Controls (MAC) also have little effect on virus activity for the same reasons, although MAC can be configured to be neither weak nor easy to bypass. Each time a virus attacks an executable file owned by a different user, it takes on the full privileges of that user, including access to files of other users whose permissions intersect the DAC and MAC permission sets of the infected user. On all systems, the need to share files forces the creation of users who exist in multiple permission sets. This multiple membership allows viruses to move between MAC compartments and levels. The reduction of multiple membership users will slow the advance of a virus but will not eliminate it. Finally, once a virus gains access to an operator account (root, operator, isso) it cannot be stopped by any form of control.

Inspection is the traditional way of locating both known holes in operating systems and locating known viruses. The key word here is 'known'. System audit tools such as COPS, SATAN and others can only locate holes known to them. Virus scanners can only locate viruses known to them. This means that a virus scanner or inspection tool is obsolete even before it is shipped from the factory. It can only deal with the past, never the present or future, since conditions searched for must exist at the time of coding. Virus scanner have to be constantly updated. This is becoming a problem with the explosion of viruses being created by new authors and virus computer aided design and manufacturing tools (V-CAD/CAM).

It has been proposed that audit tools such as COPS can be used to deter virus infections because they strengthen the system's ability to control access and data movement. These inspection tools only improve control. As stated, control does not provide protection against virus attacks. It attempts to keep outside people out and inside people within their areas of authorization.

The third category of protection is Integrity. Integrity systems are intended to detect change. In the MS-DOS world, early integrity systems used cyclic redundancy character, CRC, values to detect change. A virus was then created which countered this protection. The virus determined the CRC value of the target file, infected it, then padded the file until the CRC value computed the same. Many Unix users still use this method of change detection, or worse, they attempt to use the date of last modification as an indication of change. The date of last modification can be changed to any value on Unix systems with a simple user command. On many systems, an option of the 'touch' command provides this ability.

Any integrity tool which does not use cryptographic methods is of little value. In fact, if the integrity system fails to detect critical changes, the false sense of security created in the system operator can be devastating to the system. *CyberSoft* created an integrity tool, CIT, using the RSA Associates' MD5 cryptographic hash algorithm. Since the algorithm is cryptographic, it can detect even a single bit flip and cannot be misled by any known means. In addition, during the development of CIT, it was determined that it was necessary to detect additions and deletions to the file system, since these could be indications of non-infectious attacks such as performed by Trojan horses, worms and hackers. In this way, a rolling baseline can be created which will allow the system operator to recover quickly from any form of file system attack. Modifications to the protected file system created by unauthorized users or software attacks can be detected and removed. Using a tool of this type allows the administrator to locate the approximate time of attack, since the modification will have taken place between two known timed events, the last and current execution of the integrity tool. Finally, integrity tools can be used to determine if a third party file has been modified or tampered with prior to use. Some manufacturers of Unix operating systems now publish MD5 digests of their systems. Using these digests, it is possible to determine that the file on your system is exactly as it should be. There was no degradation from misreading the installation media, deterioration of the disk system, or intentional modification. If a manufacturer does not publish a list, end users can create their own by installing an operating system on multiple systems from different media sources. The created digests of each system should agree.

7 NON-TRADITIONAL CATEGORIES OF PROTECTION AND THEIR FAILURE

In the past, fencing systems were sold as a popular method of virus protection on PC platforms. A fencing system write-protects parts of the disk using a hardware board which is added to the system bus. Since a virus cannot infect a file that is write protected using hardware, it appears to be a good method. The obvious drawback is that the user cannot write to the disk if it is write-protected. The fencing system therefore had to create zones of protection so that the user could perform useful work. Viruses happily infected the unprotected zones. Fencing systems appear to have never been marketed for Unix systems. *CyberSoft* did provide fencing as a custom solution to an Internet service provider a few years ago. We suggested that their boot disk have the write-enable line cut and a shunt installed. The operating system was installed and logical links were created for all files which required constant modification to a second write-enabled disk. This method has been very successful against hacker attacks. The service provider has never had a write-protected file modified by an attack. Many people have tried, but the method has stood the test of time. This implementation method also suffers from the problem of zones of protection.

8 CURRENTLY AVAILABLE METHODS OF PROTECTION

CyberSoft, Inc. manufactures the first and oldest [1991] product in this category. The product is called VFind and runs on most Unix systems. Since I have not studied the other products available for Unix, I will deal with the product which I am qualified to discuss, VFind.

VFind provides protection in all three categories. It provides Control by supplying the COPS audit tool along with a proprietary audit tool called THD (Trojan Horse Detector). COPS was not developed by *CyberSoft* and is available free on the Internet; however, *CyberSoft* believes it is necessary to provide a certificate of traceability for COPS. It receives the program directly from the author, Dan Farmer, and supplies it to the end user without modification, other than packaging. This insures that the end user does not receive a Trojanised or corrupt copy of the program. The THD program makes use of the fact that many Trojan attacks use duplicate file names where the file name of the Trojan is the same as a popular Unix command in order to execute. The 'ls' command is normally stored in the '/usr/bin' directory. Since many users allow world read permission on their account control file, (aka dot-files), it is easy to learn the search path selected by that user to search for system commands. If an area that can be written into is in the search path prior to '/usr/bin', then a Trojan or virus-infected version of the ls command can be located in that directory and will be executed. The THD program looks for duplicate file names throughout the system. It also detects known high risk file names such as '/tmp/gift', which is the result of the Unix Usenix Virus (aka AT&T Attack Virus) running on the system.

Inspection is provided by a standard virus scanner. Since the Typhoid Mary problem affects Unix systems, the scanner simultaneously searches for Unix, MS-DOS, Macintosh and Amiga viruses on the Unix system. It has a user accessible pattern matching language called *CyberSoft* Virus Description Language, CVDL, which can be used to keep the scanner up to date. In fact, the end user can use legally-obtained scan codes from other vendors, or ones of their own creation, in order to provide independence from the vendor.

There were multiple reasons why *CyberSoft* felt it was necessary to develop a virus description language. The increasing sophistication of the problem was becoming difficult using standard scanning technology. Many of the viruses which attack Unix are written entirely in source code and executed in interpretative languages such as script. Scan codes cannot be easily designed to find a virus in which white space, the use of tabs, and variable names change. Normal scan codes depend on the fact that binary executables contain stable strings of code which can be searched for at specific addresses (excluding polymorphic and stealth viruses). This is only partially true in the Unix environment. Since VFind was designed to search for Unix, MS-DOS, Apple Macintosh and Commodore Amiga viruses on the Unix platform, addresses could no longer be specific, since the infected file might exist within a pseudo-disk or a compound file such as a tar file. In addition, the sequences of stable code values had to increase in size, to hold statistical validity and not generate false hits.

Scanning for viruses written in source code required several innovations in virus scanners. Many of the features required are normal parts of compiler parsers. Compiler parsers are the first step in the process of taking a computer program written in a source language and producing a binary executable. *CyberSoft* felt that a compiler parser could provide a solution to its technical goals; however, it would be necessary to define an entire language for the parser to work correctly. At the time this decision was being made, 1991, *CyberSoft* was unable to locate any standards for a virus description language. The language was defined in January 1992 and named the *CyberSoft* Virus Description Language, CVDL.

During the design of CVDL, several goals were defined. The first was to design a universal way of describing pattern matching. The second was that the language incorporate enough features that unforeseen future requirements could be resolved without changing the language or code. The grammar and versatility of the language must allow general programming within the pattern matching framework. These goals dictated many of the intrinsic features within CVDL, including the necessity to process any character or hex stream. Originally, we desired the capability of processing any length of pattern description; however, practical limits prevailed and a limit of 32,000 bytes per description was defined. A description of 32,000 bytes length can yield an actual pattern thousands of times longer, so the constraint was considered nonbinding. Boolean operators were defined, and upper/lower case sensitivity (or case insensitivity) was made a user selectable option. One of the hardest requirements to design efficiently was the ability to provide forward reference proximity scanning. This feature was a necessity to locate source code viruses. Proximity scanning allows the definition of a pattern which will not be affected by the ambiguity of the typist or white space.

One of the design features of CVDL is its ability to be used for the clean-up of data spills by searching a system for predefined patterns. While data spills are not a common problem with software attacks, they are a common problem with hacker attacks. A hacker will store interesting files in obscure locations. Many organizations caveat 'interesting' files using document headers such as 'TOP SECRET' or 'COMPANY CONFIDENTIAL'. Using CVDL, many different possible patterns of actual code can be pattern matched within defined constraints. In this way, CVDL is able to produce a basic model of a pattern which can match with a high percentage of accuracy and integrity.

Finally, an MD5 cryptographic integrity tool called CIT provides integrity to the entire file system. CIT identifies all files which have been modified, added to, or deleted from the file system. A side benefit to this ability is a reduction in help desk repair time when correcting system problems.

The use of tools from all three categories of protection, along with sensible policies and procedures, provides maximum protection against software method attacks in Unix by providing support in each area which is deficient in the other tools.

9 PROJECTION OF FUTURE PROBLEMS

I believe that the problem of attack software written for and targeted against Unix systems will continue to grow, especially now that the Internet has gained popularity. Unix systems are the backbone of the world-wide Internet. Viruses will become more prevalent because they provide all of the benefits of other forms of attack while having few drawbacks. Transplatform viruses may become common as an effective attack. All of the methods currently used in creating MS-DOS viruses can be ported to Unix. This includes the creation of automated CAD/CAM virus tools, stealth, polymorphism and armor. The future of viruses on Unix is already hinted at by the wide spread use of Bots and Kill-Bots, (slang term referring to software robots). These programs are able to move from system to system performing their function. Using a Bot as a dropper, or creating a virus which includes bot-like capability, is simple. With the advent of global networks, the edge between viruses, bots, worms and Trojans will blur. Attacks will be created that use abilities from all of these forms and others to be developed. There have already been cases where people have used audit tools such as COPS and SATAN to attack a system. Combining these tools with a virus CAD/CAM program will allow a fully functional virus factory to create custom viruses and attacks against specific targets such as companies which are disliked by the perpetrator. The information services provided by the Internet already provide sufficient information, in the form of IP addresses and email domain addresses, to identify, locate and attack systems owned by specific entities.

Finally, viruses and worms can provide the perfect format for a hostage-shielded denial of service attack. It is well known that an Internet-attached system can be made to 'disappear' or crash by flooding it with IP packets. Site administrators can protect their systems from crashing by programming their local router to filter out packets from the attacking source. The system will still disappear, because legitimate users will be squeezed out by the flood of attack packets, but filtering at the router can at least save the system from crashing. Unfortunately, anyone can masquerade as someone else on the Internet merely by using their IP address. This attack can send a barrage of packets to the target site, each of which has a different source IP address. It is not possible to use a router to filter from this type of attack, but the Internet service provider can trace the source of attack by physical channel without relying on the IP address. In cooperation with other Internet providers, the attacker can be isolated from the Internet for a short time. Hopefully, the attacker will become bored and go away, or can be identified for action by law enforcement. Another possibility is to use viruses to generate the attack. If a virus is successful in spreading to thousands of sites on the Internet, and is programmed to start an IP attack against a specific target on the same day at the same time, then there is no way to stop the attack, because it has originated from thousands of sites-all of which are live hostages. The site under attack will have to go off-line, since the Internet service providers will be helpless in the face of a coordinated dispersed attack. As the impact against each individual hostage system is low, the hostages may not even notice that there is a problem. The Internet service provider attached to the target system is in the best position to detect the attack; however, they are as subject to this attack as the target, since they may 'crash' from the excessive bandwidth usage flooding their network from multiple sources.

10 SCENARIO OF A VIRUS ATTACK AGAINST A SECURE UNIX NETWORK

The military and many other companies believe that they are protected against focused attacks because they employ a closed network configuration. In some cases these networks may also use highly secure 'B' rated operating systems [NCSC-TG-006]. Typically, the network will not allow modems, Internet connections or have any electronic connections to organizations outside of the immediate need. In addition, the networks are almost always heterogeneous because of legacy equipment, primarily PC systems. The network designers normally allow the PC systems to retain their floppy disk drives even thought their attachment to a network renders them nonessential. Networks of this type have been considered secure; however, they are open to information warfare attacks via focused virus.

Assuming that the perpetrator is an outsider without access to the equipment or premises, one possible method of attack against this type of network would take advantage of both the Typhoid Mary Syndrome and Transplatform Viruses to produce an attack which is targeted against the Unix systems but originated from an attached PC. A virus can be created whose payload is triggered by executing on a PC attached to the target network. This is not hard, with a little inside information about the configuration of the network. The perpetrator would then install the virus at all of the local Universities in the hope that someone working at the installation is taking a night class, or one of their children, will unknowingly infect a common usage home computer. At that point, the virus has a good chance of entering the target network. This is a well-known vector and is enhanced because the virus will not reveal itself. Once on the target system, the PC virus will act like a dropper releasing a Unix virus into the backbone. The payload virus may be necessary because many Unix backbone systems are not PC-compatible. The Unix virus payload can then install a backdoor which can be remotely directed. In addition, the virus can create a covert channel by making use of messenger viruses. While the use of messenger viruses are slow and have low bandwidth, they are bi-directional and can be used for command and control of more complex attacks.

11 CONCLUSION

I believe that the problem of attack software targeted against Unix systems will continue to grow. Viruses may become more prevalent because they provide all of the benefits of other forms of attack, while having few drawbacks. Transplatform viruses may become common as an effective attack. All the methods currently used in creating MS-DOS viruses can be ported to Unix. This includes the creation of automated CAD/CAM virus tools, stealth, polymorphism and armor. The future of viruses on Unix is already hinted at by the wide spread use of Bots and Kill-bots (slang term referring to software robots). These programs are able to move from system to system performing their function. Using a Bot as a dropper or creating a virus which includes bot-like capability is simple. With the advent of global networks, the edge between viruses, bots, worms and Trojans will blur. Attacks will be created that use abilities from all of these forms and others to be developed. There have already been cases where people have used audit tools such as COPS and

SATAN to attack a system. Combining these tools with a virus CAD/CAM program will allow a fully functional virus factory to create custom viruses to attack specific targets.

As these problems unfold, new methods of protection must be created. Research has hinted at several promising methods of protection, including real-time security monitors which use artificial intelligence for simple decision making. It is my hope that these problems never reach existence, but I am already testing them in an attempt to devise methods of counteracting them. If I can create these programs, so can others.

Even with the current problems and the promise of more sophisticated problems and solutions in the future, the one thing I believe to be certain is that Unix or Unix-like systems will continue to provide a payback well worth the cost of operating them.

WHY DO WE NEED HEURISTICS?

Frans Veldman

ESaSS GmbH, Saltshof 10-18, 6604 EA Wijchen, The Netherlands
Tel +31 8894 22282 · Fax +31 8894 50899 · Email veldman@esass.iaf.nl

One of the main disadvantages of most virus scanners is that they recognize only known viruses. In addition, about 100 new viruses appear each month. The exact number depends on how much spare time the virus writers have, on the season, and probably even on the weather, but an average of 100 viruses per month is a safe estimate.

This number is significant for products with a monthly update cycle because, the day before you receive the upgrade, there are about 100 viruses your installed version does not recognize. For products with a bi-monthly update scheme, there will be about 200 new viruses that may not be detected. In fact, such a product fails to identify on average about 100 viruses. Some will be detected as a new variant of something known, but many are not detected at all. Generic detection is therefore required, or at least desirable.

HOW DID HEURISTICS COME ABOUT?

Researching viruses is fun, but like everything else, a great deal of it is annoying. After having examined thousands of viruses, the fun is gone. A positive side-effect, however, is the tremendous speed of virus analysis today. About half the samples a researcher receives are non-viruses: false positives from a competing product, intended viruses, or files from a Virus Exchange Bulletin Board System. The first thing that needs to be done is to separate the viruses from the non-viruses. If you are able to do that quickly, you save valuable time.

HOW DOES HEURISTICS WORK?

Every anti-virus specialist can do it. Load the file into the debugger, and browse around. Does the file start reading command line parameters, initializing the screen, showing a copyright notice, allocating memory? Then it looks very much like an innocent program.

However, if it performs an undocumented system call to look for a strange response in order to find out whether it is already resident in memory, and if it contain routines to search for executable files, and contains a disk formatting routine, then it is something that deserves closer examination.

Heuristics attempts to perform this basic research automatically. Basically, a heuristic analyzer looks for virus-specific things, and increases a score with a certain value when it finds something very virus-like. If the score exceeds a predefined value, it yells ‘virus’.

WHY IS HEURISTICS SO DIFFICULT?

If heuristics is so simple, why don't we make more use of it? The answer is: it isn't simple! For a person, heuristics is simple, but for a person to instruct a computer in heuristics is anything but simple. The difference is crucial. The human brain is an excellent pattern recognizer. It works so automatically and unconsciously that we don't even know ourselves that we are using it and *how* we are using it.

How do we recognize people? Most of us are able to recognize hundreds, or even thousands of different faces. Even if someone has a different haircut or wears sun-glasses, we still recognize the face. From different vision angles, or on a black and white picture, recognition is no problem. Mistakes are very rare. But can anyone tell you exactly how he or she does it? Can anyone write a guideline for composing heuristics so a computer programmer can make a program for it?

Here's an example of what I mean: When the virus was new, I did an experiment at a conference. I showed a disassembly of the decryptor loop of an MtE-infected file on the overhead projector. Even before I finished the question, 'What is this?', a few people - virus researchers - yelled: 'MtE!'. Obviously, these virus researchers knew MtE and were able to recognize an infected sample immediately once it was loaded into a debugger. But the strange thing is that these same developers had severe difficulties developing a detection algorithm for MtE for their scanner!

The same applies to heuristics. For a virus expert, it is a trivial task to find out whether something is completely innocent or a virus. The problem is to find out how to identify the differences exactly. What we learned is that we see a lot of tiny details, and that we classify all these details as innocent, or virus-like, or very virus-like, etc. All this information is collated in a process that has more than a few similarities to fuzzy logic. Finally, we come to a conclusion.

Think of it like this: suppose you want to be able to recognize a bank robber. If you see someone rushing towards a bank with a nylon stocking over his head and a gun in his hand, the chances are you can assume something suspicious is going on. The fact that someone is rushing to a bank is not suspicious. The fact that he has a gun is something that triggers some attention, but since police officers (and, in the US, even civilians) carry guns, this is not a valid reason to classify someone as a criminal. The nylon stocking over his head is quite a unique thing; one you will rarely see among innocent people. But even then, children may use it to play an innocent game. But on the basis of these observations collectively, we can make a rule about identifying bank robbers: if someone is wearing a nylon stocking over his head, and is not a child, and also carries a gun but wears no uniform, and is also moving towards or away from a bank site, then it is probably a criminal. To detect other types of criminals, you will have to apply quite a lot of rules.

FALSE POSITIVES

The above example also shows that something can go wrong. The person described above is obviously a criminal, but the fact that there is a film crew on the opposite side of the street giving directions and filming the event may change the conclusion dramatically. Apparently, *one* fact can contradict a whole set of other facts. If that single quality is not in your field of vision for some reason, you may arrive at the wrong conclusion.

The same is true for heuristic scanners. A program can look very suspicious, but just one quality may make the program completely innocent: the difficult thing is to make sure that this one quality is in our vision field. This is not always the case, and results in a conclusion that we classify as a 'false positive'.

Nobody is happy with false positives: it costs time and money to find out what is going on; longer term, false positives may result in the famous 'wolf-effect'.

A false positive does not necessarily need to have a bad effect. One thing that can make the problem much easier to cope with is that the scanner says *why* it thinks that a program may be a virus.

If the scanner says a specific file is suspicious because it has memory-resident capabilities *and* a disk formatting routine, then the user is able to judge for himself the meaning. If the warning is issued about a word processing program, then chances are that something very strange is going on, because a word processing program is not supposed to be able to format disks and stay resident in memory.

Here's the dilemma: if the warning is issued about a resident disk formatting utility which provides the user with the opportunity to format a diskette 'on the fly' while still inside an application, then you know there is nothing to worry about. The same warning can have quite different meanings and consequences, depending on the kind of program it refers to. Therefore, it is important that a scanner tells you exactly *why* it thinks a file contains a virus.

If a scanner is doing this, the question also arises: what exactly is a false positive? If the scanner puts a statement in the log file that the resident formatting utility called RESFORM.EXE is able to stay resident in memory, and is also able to format a disk, is this a false positive? The information is accurate and correct. Only the conclusion that the file is a virus would be incorrect, but that is not a conclusion of the scanner. The human is the one who finally decides whether it is a virus, if he of course receives the appropriate information from the scanner.

Unfortunately, there are many products that just yell something like: 'This file may contain a virus'. What exactly is the meaning of this? If you get a diskette from someone with a new program, it *may* contain a virus, right? If you consult a scanner and it reports that the file MAY contain a virus, then you haven't gained very much.

There are three reasons why scanners do not tell you what they find:

- You are supposed not to understand it
- It may help competitors
- It may help virus authors.

The first argument is used the one most often used. A scanner should state that a file is infected; not issue a 'maybe'. The nature of heuristics, however, implies that the result is not black or white, but gray. The result will always be questionable. Supplying all information to the customer may help him. If he doesn't understand, too bad. If he actually *does* understand, then the information helped. Let everybody decide for themselves whether they understand something. In most business environments, new files are scanned in a footbath machine, so the average user is not bothered by heuristic results anyway.

The second argument is not often used, but is a valid argument. By telling the customer what you found in a file, you also tell your competitors what you are looking for. It may help them to design or improve their heuristics.

The last argument may be the best one. Supplying information about what you found in the file tells virus authors what you are looking for and may help them avoid detection in their next viruses. What actually happens is comparable to bank robbers who set up a film crew on the opposite side of the street to hide the fact that something illegal is going on.

HOW SERIOUS ARE THE COUNTERMEASURES OF THE VIRUS AUTHORS?

Of course virus authors don't like heuristics. It is annoying for them that a completely new virus is detected before it is even released. They try to avoid detection by heuristics, but that is not a trivial task for them. It *is* possible, but it has a price.

To avoid detection by heuristics, they need to hide most virus-like operations, which means globally that anti-heuristics activity has the following effects:

- Viruses become larger, slower, and more difficult to develop
- Some techniques cannot be used anymore at all
- Anti-heuristics may become a suspicious quality by itself.

For instance, virus authors used to set the seconds of an infected file's time stamp to an invalid value to allow the virus to find out quickly which files were already infected. Due to heuristics, they cannot use that method anymore, and must look for themselves inside the files. This is more difficult to program, making the virus larger and much slower. This is just one of the virus techniques that cannot be used anymore.

Also, anti-heuristics by itself is worth detection by a heuristic scanner! Some anti-heuristic methods used in the past by virus authors became, later, something that was excellent to separate viruses from non-viruses. In other words, anti-heuristics was used to define another virus-like quality of a program. For instance, virus authors used to encrypt viruses to hide what was inside and thus avoid detection by heuristic scanners. These days, with generic decryption engines, encryption is no longer useful to avoid detection by heuristics, and, in fact, has became a suspicious characteristic. An encrypted program by itself is already suspicious, and causes the scanner to look at the program more closely.

Also, there is a cross-relation between heuristics and other virus programming methods. Polymorphism, for example, makes it more difficult to create a signature for the virus, but also makes the virus look suspicious to heuristic scanners, because a polymorphic virus looks quite different from normal programs. It is one way or the other, and the virus authors have to choose. Anti-heuristics always has a price.

Virus authors have been trying to avoid detection by heuristics for quite some time. There are various documents written by the virus-underground explaining how to avoid detection by heuristics. Still, about 80% of the new viruses are detected by heuristics. Virus authors do not seem to be very successful with their anti-heuristics methods: I have seen original virus sources where the author of the virus states that he could have avoided detection by heuristics, but that it was 'too much work' and 'it finally gets caught anyway'.

Is that not part of our goal, to make writing viruses more difficult?

NEW DEVELOPMENTS

There are some new developments that will play an important role in the near future. Previously, heuristics was appearance oriented. This means that the scanner was looking for certain instruction sequences. It was like searching for criminals using the supposition that they look different from ordinary people. This is how customs officials decide whose luggage to open. It works, but not very well.

A new approach is to use an emulator to simulate execution of the virus, instead of just looking at the instructions. Instead of searching for virus-like code fragments, heuristics now looks for virus-like behaviour. We no longer search for bank robbers using the fact that they do not wear ties, but instead look for behaviour. If someone wears a tie and an expensive suit, but behaves like a criminal, he or she is still a criminal.

Why did we not use these methods before? The reason was simple: we didn't have emulators. We finally had to develop emulators in order to decrypt encrypted viruses. We can now use the same emulator to simulate execution of the virus. It doesn't matter how the virus authors hide what they are doing. If the virus is executed on a real computer, it finally has to execute its viral task, like infecting other files, no matter how it is hidden. By simulating the virus inside the scanner, it will finally reveal what it intends to do, no matter how it has been concealed.

The results of these new methods? They are excellent. Currently, about 80% of the viruses can be detected, without false positives. I expect that, with continued research, it will finally be possible to detect about 90% of the viruses, still without false positives.

This means that out of 10 new viruses, nine will still be detected! Virus authors will try to avoid detection, but it will be quite difficult for them to achieve that. Even if they succeed, they have to pay a huge price for it, and must refrain from using some convenient techniques. They also have to accept that the virus becomes larger, slower, and much more difficult to develop. Whatever the outcome, we have beaten them: isn't that what we are supposed to do?

A TESTING TIME

Paul Robinson

Secure Computing, William Knox House, Britannic Way, Llandarcy, Swansea SA10 6EL, UK
Tel +44 1792 324000 · Fax +44 1792 324001

If you have seen the film *Broadcast News*, you may recall the scene where Jane Craig, the character played by Holly Hunter, is preparing a news report for broadcast. The time available to get the taped report ready is quickly running out. People are screaming and shouting to get it ready in time, other people are shouting that there's not enough time. A production assistant snatches up the finished tape and runs with it through the obstacle course of the news room to the studio's control room. The tape is thrust into a machine and ... the news broadcast continues with the taped report prepared by Jane Craig but without any of the viewers realising the drama that has taken place behind the scenes.

Well of course that was just a movie and things aren't like that in real life – are they? No of course not. but there is more than a grain of truth in the movie about the way that the broadcast media works. To a lesser extent the same is true of the print media and this is a central characteristic of magazine publishing. To understand why things are the way they are and to understand why things are not as good as they should be, it is necessary to start here:

PRODUCT REVIEWS

There has been a good deal of criticism from one source or another about security product reviews – and in particular anti-virus product reviews. The substance of this criticism is that these reviews either are trivial; fail to address the real issue (whatever, in the opinion of the critic, that might be); are wrong in some aspect, or are based on tests which are flawed; draw conclusions which are not consistent with the test results; fail to explain their conclusions – and so on.

It is true that there have been poor product reviews. In this first part of my paper, I want to look at what happens when a magazine reviews an anti-virus product, what it sets out to achieve and why this sometimes leads to dissatisfaction in certain parts of the anti-virus community.

Magazine publishing is a funny sort of business. Unlike many products, a magazine is built from the ground up every month (in the case of a monthly magazine). This puts a lot of pressure on magazine editors to fill those pages each month, and many editors will tell you that it is a bit of a treadmill. Any of you that have been involved with a company magazine or newsletter will recognise how difficult it is to source the material which goes onto the page.

The pressure to fill the pages is compounded by a shortage of specialist journalists who can at once understand the technology and write coherently; there are many people who can do one thing or other but not both. I can count on the fingers of one hand (in the UK) the journalists who fit the bill – and from what I've seen of product reviews from other countries, the picture is the same all over the world. In these circumstances, an editor is often at a loss to source the necessary article. It could, of course, be said that it would be better if the editor chose not to cover the subject area rather than publish a less than first rate review. Such a proposition might satisfy some but it is not a solution which would survive in the real world.

Computer journalism, you would think, would include a high number of journalists who are technically accomplished. This is not, I believe, the case. The vast majority of computer journalists are *not* computer experts. If you look at the majority of product reviews they are not stern tests of the product, but are, by and large, descriptions of the product and its features. Now while this may be legitimate in showing how well a word-processor works, as far as anti-virus product reviews are concerned the question that everyone wants to have answered is how well does the product detect viruses. Therefore describing the features, the interface and so on does not significantly answer the question. To test the virus detection of a product requires some considerable understanding of (a) how viruses work and (b) how anti-virus products work. Even those computer journalists who *are* technically accomplished are only rarely sufficiently up to speed on computer viruses. Furthermore, there is considerable suspicion among journalists about the information offered by anti-virus product developers or their local distributors – clearly the developer/vendor's agenda is to promote his or her own product, so the information provided may be coloured to present the product in a favourable light. Editors could turn to journalists who have published in authoritative journals such as *Virus Bulletin* and *Virus News International* but there is concern that such publications and therefore their contributors were not impartial because of the association with their anti-virus producing owners *Sophos* and *S&S International* (though I cannot recall a single occasion on which anyone has established bias in the products reviewed in either publication). Anti-virus organisations such as *CARO* or *EICAR* could remedy the independence issue and would be well-placed to provide consultancy or documentation for journalists and magazines working in this field – sadly, however, this has not happened yet.

The majority of magazines world-wide are reliant on freelance journalists. There are advantages for magazines in using freelancers: for example, their costs are usually lower than staff journalists. In many cases, freelancers are a good option since, in principle at least, they are more likely to specialise in a certain aspect of computing. But increasingly the corporate computing experience is not replicable by freelancers who commonly will only have two or three computers to work on (instead of the several hundreds or thousands that typically exist in corporate organisations). Few freelancers will be running a network and those that are will most likely be running a peer-to-peer network rather than something like *NetWare* or *Vines* or *Pathworks*. And though there are some freelancers who are network specialists they will not have sufficient resources to have several networks, several servers or several different versions of network operating systems (few will even have several different versions of DOS running on their PCs). No freelancer I know of has sufficient resources to set up a proper lab which could be devoted to scientifically testing products. Another problem for freelancers is that they are paid on the number of words they produce. There is no additional element for the research which is demanded by the more challenging product reviews. In consequence, few freelancers have enough time to test products properly.

Some magazines have their own (or share) a test lab. Even where the lab is well-equipped and well-staffed – there are some problems which are specific to reviewing anti-virus products. For example, machines in the lab would have to contain computer viruses and for some tests would have the PCs' memory or boot sectors infected with viruses. Such procedures would make the entire lab a 'dirty' site – with all the problems that that brings – and this is made more difficult where the lab is shared with another magazine or where other product tests are queuing up for testing. Cleaning up the lab is a problem and special procedures are necessary (which procedures add to the costs) to avoid continued infection or reinfection of the lab hardware. Trying to construct a test environment which both recognises the situation which prevails for the readership and which recognises the demands of the anti-virus products being tested is very troublesome.

The lab supervisor needs to understand as much about viruses and anti-virus products as the reviewers – *and* should be in a good position to understand the day-to-day experience of MIS personnel.

Whether the anti-virus product tests are conducted by a freelancer or within the magazine's lab, all reviewers have to overcome the issue of a virus collection. We know of product reviews which have been conducted without viruses – this is analogous to a word processor review without words. The viruses have to be genuine viruses and therefore you cannot create dummy files with 'virus strings' in them – this is simply not a genuine test (we know that some anti-virus products themselves use several tests to establish a genuine infection including comparing a checksum of the alleged virus with one held on file). By the same token you cannot use simulated viruses (Rosenthal) such as were used in the NIST/*Byte* test – this is analogous to testing a spreadsheet but deciding to use words instead of numbers. If an anti-virus package doesn't detect simulated viruses as viruses, you cannot declare that the product has failed the test (since they're not viruses) and so the test is meaningless. One must use a full collection – the review should not be limited to just a few viruses or to certain carefully selected viruses otherwise once again the review is open for accusations of inaccuracy. The next problem that arises is: from which source should the reviewer obtain the genuine viruses. Setting aside the reservations that many anti-virus developers have about releasing a copy of their virus collection willy-nilly, each collection will contain a collection of files some of which are infected and some of which are not. Such 'dirty' collections will raise controversy since the product of the developer who provided the collection is likely to score higher results than other developers' products. The inaccuracy that this introduces to the test is likely to discredit the test completely. One solution would be for the reviewer to obtain an independent virus collection but few such collections exist – and even fewer would be available for the reviewer to use. Finally the reviewer could create his or her own collection, however, this is a significantly 'non-trivial' operation – and I will return to this issue later.

The economics of magazine publishing is also something that we need to consider in our look at why things are not better than they are. A principal (in some cases *the* principal) source of revenue for magazines is advertising. In the UK, the advertising rates are significantly lower than they are in the US – competition from a very large number of titles has forced big discounts to be applied to the published rate card. This pressure on revenue results in lower budgets for the editorial teams – I accept that this is only part of the problem because some publishing houses, while earning relatively good margins, are still penny-pinching when it comes to editorial budgets. The relatively low editorial budgets constrains what an editor can commission either from staff or from freelance sources. Given that many reviews are compromises between what is possible and what is achievable – all of the pressures and difficulties are further compounded by the failure of the anti-virus developers themselves (or their local distributor) to work sufficiently closely with the magazine reviewers and to ensure, for example, that the very latest version of the software is available. Considering the impact of the reviews and the criticism that has been levied against them by the anti-virus developers, it is startling to discover that some developers take such a casual approach to the knowledge that a review of their product is being considered and under way.

Finally, you have to look at how things work in the real world. When you are responsible for a publication which publishes on a monthly cycle, as most magazines do, you tend to work within that cycle. Your staff journalists and regular contributors tend to produce articles and product reviews within the framework of that month. Even where you are working with a month's or even a two month's lead time, you are still working to an allowance of time that is dictated by your publishing cycle. Therefore each month you have to produce a product review even if that review will not be published until the next month or the month after that. This means that you do not have much time to throw at a review – although to some extent as a professional reviewer (as is the case of a professional virus researcher) you learn to work quickly because you recognise how to take a product apart. At least this is true in a general sense, but it is a truism which is compromised when you enter a specialist field like security and even more so with a highly specialist field like anti-virus technology. So remembering our look at *Broadcast News* you find that the allowances of time are intolerably tight and when things go wrong, reviewers and editors have to abandon what they would ideally like to do and settle for what they can achieve *right now*. To break this cycle and step off the treadmill takes both a great deal of planning and a great deal of resources (later in this paper I'll talk about where these resources come from). To give you some idea of the scale of this operation, at *SECURE Computing* magazine we plan what we are going to publish (Editorial Schedule) about 15 months ahead of time. Major reviews take several months (in terms of lead time) of planning and background research and then several months to marshal the resources and conduct the tests. We have been planning the anti-virus product review that we will publish in the January issue since May. Our virus librarian has been working on the virus collection since June. We formally started some aspects of the testing earlier this month (September). Concurrent with the testing is the analysis of the results, retesting and checking with product developers where our tests show discrepancies with their claims. At the end of the day, tests (each of which may have taken days to complete) will appear as a simple table of results occupying no more than a few column inches. As an editor it is galling at times for me to see that such an effort is not accompanied with a fanfare of trumpets, the flash and dazzle of pyrotechnics or the glitter and glamour of an awards ceremony. But there it is.

These reviews typically occupy less than 25 per cent of the magazine but they suck up a considerable portion of our expenses. In the competitive world of magazine publishing any publisher who is seeking to trim costs will have to look at product reviews. It is relatively easy to slice away at the budget û for example cutting out a test which may have taken three days and cost a $1,000 and replacing it with a bit more description of the product or some more graphics may save the publisher as much as $900. The downside is that the review will be less stringent but in a market place where the majority of magazines are given away free what place does stringency have?

A MODEL REVIEWER

When I was preparing this paper, I rather rashly posed the question, 'Who is getting it right?' Looking at the world of anti-virus product reviews only, I feel the answer is, 'No-one!' I think there are good product reviews in the general product arena and I think there have been reasonably competent reviews of some security products. But as far as anti-virus product reviews are concerned I don't really think anyone has got there yet.

There have been some excellent efforts – Vesselin Bontchev is one anti-virus researcher who has produced laudable efforts and much as I respect his efforts, he is on record himself as criticising his own reviews. Both *Virus Bulletin* and *SECURE Computing* have in their time produced good attempts at anti-virus product reviews – and yet I think that their efforts have been some way away from the ideal. As far as the other testing and publishing organisations are concerned, I believe that they are some way behind even the best efforts produced by the examples I have cited so far.

What are the deficiencies of product reviews that have reached print in the last 12 months? One of the major deficiencies (and I'm not sure how this could be addressed) is the lack of completeness. Anti-virus product testers have *carte blanche* to decide which *products* and what *features* of those products they test. If you look at even the best reviews in the last year, you will see that not all the products available world-wide have been tested and that of those which have been included not all of their features have made it to the review page. I know why this is because as an editor, I have had to make the decision myself on what goes in and what is left out – and therefore speaking only for my own publication, which I feel I am entitled to criticise, I believe that the anti-virus reviews we have published this year have been incomplete because we left out (for example) the repair function that is offered by many products. I believe that the solution about what goes in and what is left out is something that we ducked last year when we spread our anti-virus product review over three issues – delivering network anti-virus product reviews in December, scanner reviews in January and TSRs and checksummers in February. I think it was a noble effort on our part but it did not deliver a comprehensive verdict on which is the best all-round anti-virus product.

There are I believe a number of publications who are getting it wrong. I have seen several reviews which frankly should not be given house room. One of the biggest problems for reviewers is the absence of a virus collection on which to test the products. In the early days of anti-virus product reviews, I saw product reviews which had not used *any* viruses. Later l saw product reviews which had been conducted with only about a dozen viruses. I have seen product reviews which were conducted using a virus collection from just a single anti-virus developer. I have seen product reviews which were conducted with non-viruses or simulated viruses. All these reviews are flawed and the reason that they occur is the one I gave earlier, the reviewers and editors do not have sufficient time or money to do the job properly.

I also believe there are some real problems ahead for anti-virus product reviewers. The first question is how do we represent the complex subject of anti-virus product reviews on the page of a magazine without packing the information in so densely that people do not want to or are not able to read it? How do we deal with the information overflow, for example should we make additional information available to people and what mechanisms do we use to do this – supplements, world wide web pages, ftp and so on? How do we review memory resident anti-virus products? And finally who pays for these product reviews in the future?

A MODEL FOR REVIEWS

Let me try to deal with what I think product reviewers need to do to conduct an anti-virus product review.

First, you need to establish a test protocol. This doesn't just mean devising a set of tests, it means working to a set of principles which you can discuss, publish and defend. It may well be that you will invite assistance or advice from experts in the area that you are testing – controversially much expertise exists among the very people whose products you are testing. Do you approach them and invite them to talk to you? The pros and cons could be presented as two questions: If you talk to the product developers will you compromise your integrity? If you *don't* talk to the product developers will you compromise the quality of your review?

Second, you need to prepare the virus collection for the test. There are some principles involved here: (a) the collection should not be product developer specific, (b) it should contain second generation viruses (that is everything that's in there should be a genuine infection), (c) it should contain test suites for polymorphic virus with no fewer than 500 mutations of chosen polymorphic viruses, (d) it should contain an in-the-wild test set which sensibly reflects the viruses which are genuinely in the wild, (e) the collection (and suites) should be available in some form or other for product developers whose products have been used in the test. When it comes to publishing the results, there is a convention which says that the viruses used in the tests should be identified. However, this does not make particularly exciting reading and trying to balance attractive and appealing content versus the possible value of the information is a tough editorial decision. The decision is complicated by the absence of a global acceptance of virus names – the closest we have come to this is the *CARO* naming convention but this convention uses very long names which from a layout perspective causes considerable difficulty.

Third, you need to have an adequate range of hardware and software on which to conduct the test. Part of the test should be an indication of product performance and on that basis, the test should include both standalone PCs and workstations. From a hardware perspective, a reviewer may take a decision on what constitutes the typical user's machine (and clearly this will change from time to time) but the review should at least explore a wider range of hardware and identify if there are 'significant' problems. There has to be a consistent specification and set-up on the hardware which is used for the test. In other words, if you are looking at testing the speed of a virus scan on an uninfected machine, you must make sure that the machine is properly set up. From a software perspective, the computer should be populated with a conventional choice of user software – this will vary according to the type of user (clearly corporate users will have a different software profile from home users). Windows users may include software which hinders DOS scanners – for example does the caching software work for or against the software. There are a number of aspects of the set-up which will affect the performance of a scanner and these matters should be explored – where a decision is taken which disadvantages a particular product, this needs to be explained to the readers. However, even where this happens and is explained to readers, it raises the principle of fair play since the way in which that information is presented may not claim the attention of a reader to the same extent that the overall result does.

Fourth, you need to make sure that you obtain the very latest product from the product developer. This is no mean feat – even though you might think it would be simplicity itself. One of the problems in gathering together the latest versions of a product is that developers are at different stages of development at any one point. So it may not be possible to co-ordinate what may be 15 to 20 companies to provide their latest product. Where companies delay sending out a product, the knock-on effect may mean that you could have the September version of some products and the October version of other products. This situation can arise in spite of one's best efforts to get everybody moving to the same beat. The situation is further confused where a product developer is not local and the local distributor has agreed with the developer that he or she will be responsible for all press contact. In such situations it may be completely impossible to get the latest version of the software – and hostilities can break out if you try to deal directly with the product developer.

Fifth, you need to establish a procedure for identifying errors in the tests you have conducted. Part of this process will require an analysis of the results. This is not easy to achieve because it is not always clear whether results make sense or not. In order to analyse the results, you have to do more than simply run a scanner at a collection of viruses and then read off the number it detects and the number it misses. You have to know which viruses it misses – it may be that some of the viruses in your collection (even though they are genuine infections) may not replicate and are therefore not genuine viruses; you can only discover this if you have a full report of the results. To take another example, if you are considering the results for scanners on the polymorphic suite, you may wonder why an otherwise reliable product fails to detect some of the infected files. In the past we have used an older version of the software to check if *it* detects the infected files – sometimes products lose their ability to detect a particular virus reliably (it has happened); sometimes the product has introduced some new capability which improves its detection performance (such as generic decryption) and what you were confident were viruses (and had been identified by scanners as viruses) are now more accurately identified as non-viruses – in the first case the product developer was wrong in the second case we were. Checking to see whether the infected files will replicate is one sure-fire method of identifying whether this is a genuine missed shot or not; such non-viruses should, of course, have been weeded out of the collection already. Another procedure which we recommend is for anti-virus product reviewers to pre-publish their results to the appropriate product developer. (*SECURE Computing* employs this as one of its product review procedures and has published notes of guidance which explain what happens in this stage of the product review process.) You can only produce reliable results if you analyse them and ask questions about them – it is, however, all too tempting to look at a set of results and say, 'Hey, Product X has improved/deteriorated over the last year.' Such an approach accepts the validity of the test above the validity of the product – it may be that the test *is* valid or the converse may be true – such a conviction, however, leads a reviewer to be blinkered and makes the likelihood of errors very much greater.

Sixth, you need to think carefully how you will conduct the test on each product. This is not such an obvious statement as it might seem at first (he said, patronisingly!). The way that you conduct the tests has to remove as many of the variables as possible. This may mean, for example, running DOS scanners from the C: prompt, cutting out the screen display, piping the results straight to a report – in other words cutting down on those things which add variables to the results equation. It also means drawing some conclusions about on whom the report is based. Is what you are doing the action of a typical user? (And who is your typical user – is it the system supervisor or the end-user?) The decisions you take here may dramatically affect the outcome of the product review. Some virus scanners have a heavy resource-drain in the shape of the interface which either prevents or severely limits the scanning of a large number of files. Accommodating these scanners by switching them to quiet (no screen report) mode makes it possible to test them but it also tests them in a way which is contrary to the default mode of operation and (arguably) derives results which do represent the normal performance of the product. This is a matter which different people take widely different views upon. In this sense, your test needs to consider the design of the product you are testing though this is only rarely explained by the product developer. Frequently you discover features of the design when you try to run a test and it doesn't work properly; you then spend a lot of time tracking down the problem/feature through technical support who either don't know about the problem/ feature or *say* they don't know about it. Failing to recognise that your test has to accommodate product design may result in the misrepresentation of some products. For example, we have recently been looking at products which include a heuristic element within them. Some of those products have been designed for different users and this emerges in a fundamental aspect of the product namely how does the product report an infected or suspicious file? If the product is aimed at a user with a higher than usual level of understanding of computing then the report can contain information which would baffle a novice. Other products assume that the user knows little or nothing about computing and therefore the reports take a different character. Depending on which design you accept is valid will determine whether you characterise the reports on suspicious files as false alarms or not. Coming to a decision one way or another could leave you open to charges that you are simply ducking an important issue (false alarms). The job of the reviewer is to determine whether this *is* a legitimate design issue or whether it is an attempt by the product developer to win a more favourable treatment for his product.

Finally, you need to consider how the various tests are going to be marked. You will need to decide on the principle on which marks are awarded – will you award marks according to a firm mathematical model or will you use a sort of fuzzy award and totting up of pros and cons – based, for example, on a product being *Poor* or *Below Average* or *Average* or *Above Average* or *Good*? You may also need to combine the marks from the individual tests in some way to achieve an overall or final mark – this adding up of marks may employ some sort of equation which gives priority to one test over another. How will you make all of this clear to your readers? Will you present your results in one overall table or will you split the results across several tables – and will you explain why you award your final recommendations or opinion?

There are many further aspects of what you have to do as a product reviewer – but many of those things revolve around how you manage the review process. How we manage it – is a commercial secret. But what I can tell you is that it involves a very heavy investment in hardware. It also involves a great deal of planning, heavy investment in the preparation of systems and finally heavy investment in frustration avoidance.

... AND SOME OF THE PEOPLE ...

In preparing this paper, I conducted a number of interviews with the readers of product reviews. As it happens many of the interviews were conducted in the US which I thought appropriate since this paper will be given in the US. However, some of the interviews were conducted in the UK because of the difference in the magazine publishing trade in that country. All interviews are reported on a non-attributable basis because the individuals concerned either did not want their companies to be identified or they could not get clearance in time. (Only two of the interviewees could obtain the necessary clearances in time so I decided they should not be singled out.) I have condensed the information from all the interviews.

One of the things that surprised me was how avidly people read reviews. Typical of the people interviewed was Daniel who said that he sought out reviews when he was looking for support for a buying decision. Daniel is a well-qualified expert in the area of network comms and works in a company that produces a number of products in that area. In response to questions on the quality of product reviews he said that he formed judgements based upon how the review was written, what level of detail was included, the accuracy of the reviewer's knowledge (wherever that coincided with what he already knew). Daniel said that he had been prepared to approve purchases on the basis of reviews he had read but he qualified this statement saying that each case was taken on its individual merits and the product very much dictated the final outcome.

John is the head of a marketing department (for a household product manufacturer) who is particularly interested in products which assist his department in their work. John was less critical on the whole than Daniel of the product reviews he read. He said that he looked at any product reviews for software that he was thinking of buying but also read reviews (from time to time) on products that he had recently purchased or that he was using personally to see how they rated against other competing products. He said that he was particularly interested in reviewer's opinions.

Shamilla is the manager of an MIS department (within a large corporation) who said she had little time for reading product reviews. She said that she mainly used product reviews as a source of information about the marketplace and that she tended to skim through the reviews identifying products to be brought in-house for evaluation. She said that she utilised the reviewer's opinions to eliminate products rather than to choose products. She was particularly conscious of those reviews which confirmed her own opinion of the product (where appropriate). She said it was unheard of for a major decision to be taken as a result of a product review although if a review which was hostile to a product appeared at a critical period, this might cause the purchase to be delayed while further consideration was given. Where two products were neck and neck, a favourable review might swing the balance. Shamilla had telephoned editors and reviewers about product reviews where she was looking for additional information (which is how I first became aware of her existence).

Simon works in the accounts department of a major retail network. He regularly goes out on the road to visit different sites and uses a notebook computer. He purchased a computer for his own use at home and unlike many users does *not* use it for games and connecting to the Internet; he does not allow his children to play on it, using it instead for a little accounting freelancing. He is an avid reader of product reviews and purchases computer magazines regularly which he reads while commuting to work. Simon does not consider himself an expert in computing and pays great attention to the opinion of the reviewers – he is principally interested in the products that the reviewer gives high marks to, and doesn't always read those sections of the review about products which were given low marks. He also pays close attention to features tables to see which products contain the most features. Simon has asked for various products to be bought by his employers based on the information he has gleaned from product reviews. He is seldom successful in getting what he wants from the MIS department but has occasionally purchased software for his own use on the basis of the reviews he reads.

From the results of the interviews I conducted, we can see product reviews *are* read and considered by those people who have an interest in software or hardware purchasing or who have some interest or responsibility for the way their computers function. The picture is very different among ordinary users who are only rarely interested in product reviews (or any other aspect of their PCs apart from the speed at which the 'damn thing' operates). Corporates pay close attention to the product reviews but temper the advice they receive in the light of their own experience – they are less likely to accept product reviews at face value though this is mainly because the purchase of software or hardware is only a small part of the IT bill for a corporate; switching from one security product to another causes MIS departments great upheavals and they, on the whole, were extremely unlikely to embrace this unless there were significant benefits – a few percentage points on a virus detection chart (even where those viruses were in the wild) was not considered, by the MIS departments I spoke to, sufficient grounds to move from an existing product. Small and medium sized companies were more likely to buy a product or make a switch to a different product where they had experienced some problems and felt that a different product could solve the problem. Home users were the most likely to buy a product on the basis of a product review.

[I conducted 36 interviews among a wide and representational sample of computer users. The views which the characters above express are a generally a composite of the interviews I conducted.]

WHAT TO LOOK FOR IN A REVIEW

It isn't easy at first glance to be able to see whether you can put your faith in a product review. Many reviews are written in an authoritative style designed to give you confidence in the reviewers opinion – that is, of course, what they're in the business for.

Some reviewers will indicate that they have a history in product reviews of this type. They may indicate how a product has improved or deteriorated since they last looked at it. While it is difficult to generalise on these matters, I would be looking for depth in a review – a feeling that the reviewer was at home in the subject under review or had just completed a very thorough research of the subject. This is indicated by his or her understanding of aspects of the product being reviewed. For example, why were polymorphic viruses invented and why are they difficult to detect, and in this context what is the significance of generic decryption?

Secondly I am looking for what isn't reviewed in the product review. What features of the product have been ignored and why might that be so? One of the big problems for reviewers, particularly magazine reviewers, is testing memory resident (TSR) virus scanners. I can tell you that *SECURE Computing* has done a good deal of research into this area and there are no short cuts and no utilities which facilitate testing TSRs. Many product reviews do not have tests for memory resident scanners because they can't do them. Is this a problem? It is when you realise that TSRs and now VXDs are going to form, in my opinion, the front line in virus defences. You can ask the same sort of questions about the absence of tests for heuristics, repairing infected files and scanning inside compressed files.

Thirdly, I would want to look at the level of testing within a product review. Testing is difficult, expensive and time-consuming. It is much easier to describe the product, the interface and the user manual. If the number of tests are fairly limited and if the analysis of those tests is limited or missing, this is another indication that this is a less-than-fully-competent product review. What I would seek as an ideal is a good balance between the description of the product, its interface, manual and so on and the tests which examine the technical prowess of the product. It is also important to have descriptions of the tests and how they were conducted, where products failed and what was the significance of the failure, what viruses were used (in certain circumstances) and where they were sourced (I'd be extremely sceptical about a virus collection which suddenly materialised out of nowhere – say the *X Magazine* virus collection). Also as a part of what's ideal, I'd like to see some elaboration of the way that performance is marked (and the marks are combined) so that where I disagree with the marking, I can nonetheless still make use of the data.

Fourthly, I am looking for how the product reviewer examines issues like the interface, or the management aspects of the products, or the technical support, or the manual – I want to know what importance is put in such issues (if any) so that I can determine how such aspects contribute to the overall assessment of the product.

Finally, I think it is important that the product review presents its information in a way which is helpful to users. It is all too easy for publications to hurl information at readers with little or no thought to how it is going to be interpreted.

HOW CAN THE SITUATION BE IMPROVED

I've spent a lot of time in this paper indicating that the reviewers are at fault, where they're at fault and how they can change things. But this is not the complete story. There are two other players in this tragedy, the first is the reader and the second is the developer.

Let's deal with the developer first. I've been reviewing anti-virus products or editing reviews for a good many years – consistently more years than any other editor in the industry. In that time, I've come to know most of the developers of anti-virus products personally. Many of the developers could do a lot more to make the product reviewer's job both easier and more effective. Unfortunately, too many of the people who are dealing with the press are from the developers' marketing departments – and what they know about the technical side of the product (to be kind to them) is insufficient for the reviewer's purpose. Most marketing departments are working on the 'short-term principle' – that is they want the review to say good things about their product now, they're not interested in solving the problem long-term. The trouble is that to do good product reviews on anti-virus products requires a long-term commitment by the reviewers – I've indicated above the sort of time and computing resources that need to be put into the reviews.

Some of the developers *do* recognise the benefits of working with the press and their technical staff *do* make themselves available – but the technical staff (particularly the most accomplished among them) feel very much under pressure and they are reluctant to invest much time in addition to their other commitments in talking to the press. Clearly only the developers can decide on the priorities for their staff but this matter it seems cries out for such investment. In addition, the marketing department and PR agencies are extremely nervous when a techie gets up in front of a conference or user group audience or speaks to the press. They are never certain what a techie will reveal about their product (which they would prefer was kept secret). I once turned up for a one-on-one interview with the chief architect of an anti-virus product and found him chaperoned by *four* marketing people, it is extremely rare for me to get an interview with any techie these days without at least one marketing person present. It is a given fact that techies and marketing people neither understand each other or want to understand each other (neither much valuing what the other does) and this is an area which needs to be improved if the product reviews are to be improved.

One of the questions which is invariably asked by PR agencies, sometimes asked by marketing departments and almost never asked by techies when talking to the press is, 'What's your deadline?' This simple question will indicate the pressure that the journalist is working under. If something has to be given or sent to the journalist you need to know how much time is available. Many journalists are frustrated when their attempts to do a good job are thwarted because the company concerned does not respond within the time available. I can testify to the number of times, we have failed to get a response from a company to some question which results in the product receiving a less flattering review than it might otherwise have done. Remember the *Broadcast News* example I mentioned earlier, final deadlines in publishing mean just that – if you pass over the line you're dead. I remember my first editor chiding me (in that jocular and friendly way that is unique to editors) because I was running up against a deadline, 'I don't want it good', he snorted, 'I don't want it bad. I want it NOW!' If you want your products to do better, you'd better understand how journalists have to work – this isn't saying that their work is more important it's just saying that these are rules of the game that you're playing in once you produce a product.

In another sense developers have already started to respond to reviewers – by cheating. We came across one developer who sent us a special version of his software which was designed to detect every 'so-called virus' in a certain virus collection. Unfortunately for that developer we source our viruses very widely and we also employ a virus researcher to manage our collection. We spotted the 'deception', demanded a genuine copy and then wrote up the whole story. That was probably the most blatant episode – however we know of other developers who have tweaked their products so that they don't irritate reviewers (who of course subject the product to quite different pressures from those that a user will).

The other player in our little tragedy is the reader. To some extent, I feel you cannot blame someone for wanting something for nothing – and clearly it was the publishers who started this business of giving publications away for free. But the pressure of costs on publishers which this has caused, has caused some of the problems which I have spoken about here. There is a truism that you only get what you pay for – and it is tempting for me to suggest that this is true for all those magazines which come free. But I do not think such a direct link between cause and effect is accurate. There are some free magazines which are excellent and then there are also subscription-based publications which are dreadful. I believe, however, that the free magazines apply an unhelpful pressure on the publishing marketplace – which is unbalanced as a result. Some readers seem now to take the viewpoint that all publications should be free – and are therefore dismissive of *any* publication which charges a subscription. It seems to me that in the end all readers lose from this approach and are poorer both in the knowledge of what systems are out there and in the quality of software (whose developers, without a well-informed and active press, are likely to concentrate on pressing forward their marketing campaigns rather than their research and development programme). Readers need to be more demanding of magazines but their voice is not as powerful as it might be since as receivers of a freebie magazine they are not customers in the true sense – they have not *directly* bought into the product. Free magazine receivers cannot vote with their feet and indicate their dissatisfaction – they can't stop buying something they don't care for, it keeps on arriving whether they read it or throw it in the trash can. And because of the absence of this crude but effective 'approval indicator', editors and reviewers don't get the message. Sales of bought magazines and newspapers *do* fluctuate when readers disapprove of what they're being given. To give an example, the *Sun* newspaper (for those who may not know it, *The Sun* is a tabloid daily newspaper in the UK) reported a tragedy which involved Liverpool people in a way which the people of Liverpool regarded as unacceptable. The sales of the newspaper in Liverpool collapsed, plunging from 524,000 per day to 320,000 per day – a loss of 38.9 per cent.

IN CONCLUSION

While product reviews are improving in some senses, there are strong forces at work which discourage rigorous product reviews. Even where some publications are making valiant efforts to produce meticulous evaluations of products there is little or patchy support from the product developers or the reading public. It is easy to criticise magazines and deplore their lack of expertise and lack of thoroughness in conducting product reviews – but such criticism is on the one hand misunderstood and on the other is essentially a waste of time. To do the sort of thing that *Virus Bulletin* and *SECURE Computing* have been trying to do is buck the trend in computer publishing.

I hope that this paper will provide you, as a member of the product-review-reading public, and you, as product developers, with some insight into the way that magazine publishing works – and why things are not as good as they could be and how they could be better.

FENDING OFF VIRUSES IN THE UNIVERSITY COMMUNITY: A CASE STUDY OF THE MACINTOSH

Judy R. Edwards

Illinois State University, MC 3470, Normal, IL 61790-3470, USA
Fax +1 309 438 3027 · Email jedwards@rs6000.cmp.ilstu.edu

ABSTRACT

There is a potentially risky situation evolving with regard to the risk of virus infection and transmission due to the portability of telnet (Internet) clients coupled with the work habits of the university community. But rather than resist temptation to use the wealth of Macintosh telnet clients available, sound virus protection methods create a safer environment for obtaining and using them.

Experts estimate the number of MS DOS viruses at 5500 and growing [1], and the ease of writing and transmission seems to point to the conclusion that there is no real end in sight. While there does not appear to be exponential growth of the DOS virus population now, we have witnessed over the past few years an enormous increase in the number of DOS viruses. The same cannot be said of Macintosh viruses. However, there is a potentially risky situation evolving with regard virus infection and transmission due to the portability of telnet (Internet) clients coupled with the work habits of the university community.

The easily-installable and freely available Macintosh telnet (Internet) clients have made accessing internet resources easier than ever. It is common practice for members of the university community to use telnet clients to download and transport their email, and other personal configuration files, between their office and home computers on a diskette. Students who rely on public workstations have no choice in the matter.

This is also a population accustomed to sharing data with other institutions rather than shying away from doing so with anyone outside 'the company' This behavior has carried over into the free exchange of telnet clients, often without regard to exposure to viruses.

Yet the temptation to move files and applications across the Internet almost becomes irresistible. After all, this is the essence of computing: the sharing of information!

Should you resist using telnet clients? Of course not. A more appropriate response is to keep current virus protection in place and to educate end users on how to deal with the presence of a virus.

FILE EXCHANGES

Macintosh telnet clients have vastly increased the frequency of file exchanges of all types across the internet. Where once a user may have had only one shareware file, they may now easily pass through numerous files during a single month – obviously increasing the probability of exposure to a virus-infected file.

POPmail clients simplify mail transfers (and attached shareware application and text files) by physically moving mail from a Unix machine to a Macintosh, for instance. Text files, which can contract INIT 29 [2] or the MDEF A, B or C strain [3], can be exchanged electronically as end users collaborate on projects via popmail attachments, innocently passing a virus onto a colleague.

File Transfer Protocol clients eliminate the need to learn the command line ftp process to transfer even entire directories of data – including collections of additional shareware clients – to your Macintosh with lightning speed from anonymous ftp sites. Gopher clients not only locate information on the Internet, but automatically ftp and install new shareware onto your Macintosh as you continue to work. The basic default configuration takes place with initial use – and, if virus protection is not in place, so can contamination of your Macintosh.

Installation of the Macintosh clients that make all this possible requires merely that MacTCP, Apple Remote Access, or a PPP client be installed on the workstation. The main application can either reside on the hard drive or the user's diskette – provided it remains unlocked when in use.

End users store their personal application settings files (INITs or Preferences), on their Macintosh, along with email, bookmarks, and hotlists storing Internet addresses for favorite sites for downloading information, and subscriptions to UseNet newsgroups they regularly read on a non-write protected diskette. By saving these files to an unlocked diskette, the end user can access information resources they use frequently as they move from one Macintosh to another.

There are two inherent problems in this situation. The first problem is the security risk of randomly transferring files, either applications or mail attachments, without heeding where those files have been. Each time a file passes across another Macintosh desktop, it is potentially exposed to a virus which it can contract or pass on to any other Macintosh desktop it passes across.

The second problem pertains to end users who move from one workstation to another, including students who use open labs as well as staff/faculty who carry client settings files between the office and home computer on a diskette that is not write-protected. Removing write-protection from a diskette creates an environment ripe for passing viruses.

RANDOM FILE TRANSFERS

From a support viewpoint, the ease of randomly transferring files is a tremendous boon to a university's ever changing flow of new end users. It has become commonplace for universities to pre-configure telnet client applications and utilities for end users and then make the clients widely available. This process saves the Help Desk staff from having to personally assist every end user through the configuration process.

The original 'clean' copy will be opened and the site-specific information entered, such as the names of the university's gopher, news, web, mail servers. MacTCP can be pre-configured for the correct domain name and gateway, before being passed on to users, as part of a site licensing arrangement. Communications software can be pre-configured with the local dialup number.

Files are then compressed, usually with a product such as BinHex or Compact Pro, among dozens available, that have also been downloaded from an anonymous FTP site. Once compressed, the pre-configured files are stored at the university's anonymous ftp site or on various Macintosh servers or public workstations where end users are encouraged to take their own copies for personal use.

But the question always remains as to how clean an application transferred via the Internet may be, and how clean the server that application resides on may be. Anyone can offer up applications from their server – and that web server or ftp server may actually reside on a Macintosh which could have virus infections on it and, by association, pass those viruses on.

There exists the temptation to use a Macintosh as a file server, rather than a unix machine, because files can be transferred either via ftp or AppleShare, offering users greater choice in how they access the server. It also introduces more contamination points, beyond just the traditional unix-based Internet server.

Anyone maintaining shareware on a Macintosh fileserver, must maintain stringent virus protection on that fileserver in order to avoid the risk of infecting clients being made available to end users. This is especially true for those individuals who open and pre-configure them. Infected applications don't even need to be run in order to pass on viruses such as ZUC and MDEF [3].

In addition, the clients make it so easy to begin participating in this file exchange with other end users, further increasing the end user's vulnerability. The nature of the world wide web contributes to the need to participate in this free exchange of applications. The plethora of 'helper applications' for viewing graphics, listening to sound files, or viewing movies within the web clients are also transferred over the Internet.

As users gain greater exposure to the wealth of data files available, they may also be exposed to a destructive infection. One Macintosh Trojan horse, CPro 1.41, masquerades as the commonly-used decompression program Compact Pro 1.41 and erases the System and floppy diskettes [4].

It is crucial that virus protection be installed and kept up to date on every single Macintosh and that anyone pre-configuring clients vigilantly maintains a sterile environment to avoid putting end users at risk.

TRAVELLING END USERS

The mobile nature of end users adds yet another entry point for infection. Faculty often work from both home and office. Students who don't have their own computer have no choice but to travel between computer labs. In order to make this possible, they carry clients on an unlocked diskette from one workstation to another in order to maintain access to their personal information. As these end users move files from one workstation to another, they may inadvertently encounter a virus on one of those machines or pass it on to another.

PROPER USE OF VIRUS PROTECTION

Commonly used Macintosh virus protection products include Disinfectant, Gatekeeper, MacTools and Virex. Each require that a Control Panel be stored in the System folder. It is also necessary to open the application and configure the software. Many end users skip this step, causing the virus protection to fail to scan disks or notify the end user when a virus has been detected or removed.

The ease of installing most Macintosh applications is contrasted by the fact that one of the more effective tools, Gatekeeper, does require some technical expertise to install properly. Even Disinfectant and MacTools require some installation and, if not configured properly, can fail to scan floppy diskettes. Some products such as Virex sometimes must be disabled when installing other applications and then automatically reactivated after a specified period of time.

Disinfectant is possibly the most commonly used – and one of the more effective – tools, partly because it is a freeware product itself. However, it does not address viruses which infect HyperCard stacks or trojan horses. Trojan horses, briefly defined, are 'stand-alone applications that must be launched in order to be activated, and that are similar to viruses in the damage they can cause' [5].

While the telnet applications themselves are not HyperCard-based, many of the helper applications for creating hypertext markup language documents are HyperCard add-ons. For HyperCard users, Disinfectant is not fully effective and they need additional protection such as Virex.

SCANNING FLOPPY DISKETTES

It is possible to configure Disinfectant, and other products, to skip scanning floppy diskettes. Since the telnet clients' settings file may reside on a floppy diskette, this is an unwise decision. Older versions of Eudora popmail, for instance, are small enough for the entire client, along with the end user's mail, to reside on a high-density floppy diskette.

It maybe possible, with any product, for an end user to cancel a scan. To counter this, Virex and MacTools can be configured to perform a scan at startup, shutdown – and at pre-selected intervals while the workstation is running. If a scan was cancelled during a virus infection, the next automatic scan should detect the virus.

COMPATIBILITY PROBLEMS

The Gatekeeper documentation stipulates that certain files such as 'communications programs, compression utilities, and electronic mail packages require File(Other) privileges when decoding downloaded applications and system files'. Not assigning these privileges within Gatekeeper can interfere with the way people actually work when they use the various telnet clients and decompression programs such as StuffIt Lite.

One advantage of Virex is that it protects against some trojan horses, in addition to viruses. However, a QuickMail Server will not transmit mail messages over a modem while Virex is running. The same is true for Apple's PowerBook Express Modem software for fax transmission. The solution is to disable the feature that diagnoses HyperCard stacks. These are not problems – unless HyperCard users are on the same network, or unless users are using a modem.

A novice is not always prepared to deal with these issues and may either stop using virus protection or spend a regrettable amount of time calling a Help Desk for assistance.

NETWORKS

Serving up applications over a server is a wonderful way to manage applications on every end users' workstation and include tools which prevent end users from disabling virus protection on the system. It is a wise decision on the part of the network administrator to ensure that the entire network's integrity is maintained.

Networked users must be made aware of upgrades that have been transparently installed for them, including an upgrade of their virus protection so that they also upgrade protection on their home computer. Site license purchases can include users who work from home in order to encourage all users to keep their workstations current.

KEEPING THE CAMPUS CURRENT

An ongoing problem for everyone is just keeping current of the latest product releases. Commercial vendors are sometimes unaware of the structure of the American university system. While there is often a central information services office on campus, individual departments generally have the freedom to purchase software independently.

Frequently, an individual department will purchase a site license for the software, unaware that another department has done the same. The contact name for the most recent purchase will then become the new contact name in the vendor's database, even though it may be the secretary who processed the invoice for an academic department, for instance. That person may not be responsible for actually maintaining software and therefore not comprehend that this is a crucial upgrade.

When an upgrade announcement is sent to the university, it may only also be sent to that single contact person, bypassing all other departments including the campus computing services. It may also be that anyone wishing to purchase the upgrade, may need to order through that department who was last-listed as the contact. Vendors are often unprepared for listing multiple contacts and departments as 'the' official contact point. Work with your vendor, and explain that your university may have multiple site-license contact points, if that is the case.

Through other means (such as trade publications), managers and computer services can remain aware of new upgrades even before – or in lieu of – receiving an announcement. Universities sometimes also keep users informed of new upgrades via a university-based listserv or mail list. John Norstad makes the following recommendations, among others:

- Join a user group such as BMUG (Berkeley Macintosh User Group), BCS (Boston Computer Society) or a local user group.
- Subscribe to the BITNET distribution lists VIRUS-L and INFO-Mac.
- Read the USENET news groups comp.sys.mac.announce and comp.virus.

DEFENDING AGAINST VIRUSES

The old adage 'keep your disks locked' is no longer sufficient, nor appropriate since doing so prevents the clients from updating personal settings or exchanging email. But there are reasonable responses.

- Install virus protection on every computer, not just scanning stations or at the office.
- Install and maintain current virus protection to immediately scan newly downloaded and decompressed files before using them.
- Configure virus protection to issue a notification if a virus has been detected and destroyed.
- Run a second scan just to be sure that you have eliminated the virus. If a virus such as INIT 29 is currently infecting one file it could reasonably be infecting others as well.
- Configure products like StuffIt Lite 3.07 to instantly activate resident virus protection to scan new files as they are decompressed.
- Rebuilding the Desktop when installing new applications is not only good maintenance procedure but it can remove the WDEF and CDEF viruses.
- Use Checksum Tools such as Checksum from Geoff Walsh or Virex and MacTools which have built-in checksums to perform integrity checking similar to that found in some MS DOS Anti-Virus programs.
- Download a 'clean' copy of virus protection and scan public or shared workstations before starting to work.
- Scan diskettes every time they are used at public or shared workstations.

- FTP 'clean' copies of telnet clients from an anonymous ftp site rather than accepting shareware that someone has been using as their personal copy or on public workstations.
- Scan software that has been pre-configured for your site.
- Notify end users when there is a new virus protection upgrade available to them.
- Network Administrators should be especially wary of running an infected client on a network server, thereby passing on an infection.

The privilege of sharing of information carries with it the responsibility for protecting yourself and others. Sound virus protection methods create a safer environment for obtaining and using the wealth of Macintosh telnet clients available.

BIBLIOGRAPHY

[1] Gordon, Sarah, 'Technologically Enabled Crime: Shifting Paradigms for the Year 2000', *Proceedings, Sec 94,* IFIP TC11. Curacao, Netherlands Antilles. 1994

[2] Spafford, Gene, 'New Macintosh Virus Discovered (INIT-29-B), 2 April 1994', Purdue University. 1994.

[3] Norstad, John, 'Disinfectant 2.5.1', Northwestern University. 1991.

[4] Harris, Kevin, 'Virus Reference 2.1.3', *Software Perspective.* 1994.

[5] Schnier, Bruce, 'Virus Killers: Macworld Lab tests virus software and survives', *MacWorld,* July 1994, p. 116ff.

ACKNOWLEDGEMENTS

The author would like to thank Sarah Gordon for countless hours of assistance in exploring similarities between DOS and Macintosh threats. It is my hope that collaborative work between researchers will lead to solutions to this ever growing threat.

RECENT VIRUSES, VIRUS WRITERS AND ROUTES OF VIRUS SPREAD IN HONG KONG AND CHINA

Allan Dyer

Yui Kee Company Ltd, Flat C & D, 8th Floor, Yally Industrial Building, 6 Yip Fat Street,
Wong Chuk Hang, Hong Kong

Tel +852 2555 0209 · Fax +852 2873 6164 · Email adyer@yuikee.mhs.compuserve.com

ABSTRACT

During the past two years, a number of viruses have emerged from Hong Kong and China. Some have spread internationally while others are only known locally. Hong Kong and China are simultaneously very close and widely separated in terms of business, communications, technical knowledge and availability of computing power.

The types of virus written and their spread are dependent in different ways on the environments in the two cultures. This paper will look at the viruses written and relate this to what is known about the virus writers and virus writing groups in Hong Kong and China.

The differences in virus reports from business and BBS users in Hong Kong show that different viruses are suited to spreading in these environments. Corporate indifference and teenage irresponsibility contribute to the virus problem in different ways.

INTRODUCTION

I think everyone knows that Hong Kong is a major international finance centre and the biggest entry port to China. How do viruses fit into this environment, and what effect does this have on the international virus scene?

BACKGROUND

Change is dominant. For Hong Kong, 1997 will mark the end of British rule and preparations are being made throughout society. In China, the use of computers is growing enormously. Last year, the estimated number of PCs in China was one million. This year it is double and the growth will continue.

THE LAW

Relevant laws exist in Hong Kong & China. Hong Kong has the Computer Crimes Ordinance, introduced in 1993 [1]. The first case prosecuted under this act earlier this year, was a 'cracker'.

In China, the National Congress issued a Guideline in February 1994 [2]. This places the Ministry of Public Security in charge of the whole nation's computer information systems security. More specifically, this Ministry is given responsibility to manage research and investigation of 'computer viruses and other data endangering social and public security and related harmful programs'. In addition, a permit is required for selling products specifically designed for computer information systems security. These permits are designed and administered by the same Ministry.

Paragraph 28 of China's guideline makes two definitions: a computer virus is a program or programs inserted into computer programs which can destroy the function of the computer and/or destroy data or influence the function of the computer. It is also a program that can be self-replicating. This definition appears to cover most malware: viruses, Trojan horses, logic bombs, and maybe even buggy programs.

Secondly, a specialised computer information security product is a hardware or software product specialising in protecting computer information system security.

Paragraph 23 of the guideline specifies punishments: deliberately introducing computer viruses or any other harmful data which endangers computer information system security, or selling a computer information system security product without a permit, can result in a warning from the Ministry of Public Security or a personal fine of up to 5000 yuen. Organisations may be fined up to 15000 yuen. Any gains from the violation would be confiscated and there may be a fine of from one to three times the quantity gained. As a comparison, the average monthly salary in China is 200 to 300 yuen, although anyone privileged with access to a computer could have a much higher salary.

This leads to a curious situation. People who spread viruses do not gain from their actions, so the penalty for selling a security product without a license is potentially higher than the penalty for deliberately spreading a virus. It seems that the National Congress failed to understand the nature of the problem when it issued this Guideline. The level of the penalty is also lenient for a country where corruption can be punished with the death penalty. For comparison, deliberately spreading a virus in Hong Kong can be punished with up to ten years imprisonment.

However, the level of punishment is irrelevant when no prosecutions have been brought in either jurisdiction.

THE VIRUSES

I have compiled a list of viruses for Hong Kong and China (table 1). This is different from my submissions to Joe Well's 'In the Wild' list because I have used a slacker criteria for inclusion: I must have received a sample of the virus from someone in Hong Kong or China. Some of these I received in collections; some I received via local BBSs, and while I could see that they were spreading, I could not confirm a single case of infection. Rather than indicating the viruses in the wild, this list indicates the viruses which are 'available'. Some of the older ones were probably in the wild in the past; the newer ones may turn out to be mere zoo specimens or the first indications of a wider spread.

The list is short: just 52 viruses for Hong Kong, and a handful for China. It is almost certainly incomplete; the China list should improve as stronger links are forged. For Hong Kong, it is possible that my contacts are insufficient, but it may also be because the local 'virus collector' community does not have frequent exchange with their European and American counterparts. This brings us to communications.

TABLE 1: HONG KONG & CHINA VIRUSES

Hong Kong

AntiCad.2900.Plastique.A
AntiCad.4096.Danube
AntiCMOS.A
AntiEXE
Burger.560.AY
Burger.560.BA
Cascade.1701.A
Crazy_Lord
Dalian
Green_Caterpillar.1575.A
Green_Caterpillar.1575.D
Green_Caterpillar.1575.H
Jerusalem.1808.Standard
Jerusalem.Clipper
Jerusalem.CVEX.5120.A
Jerusalem.HK.2358
Jerusalem.HK.2513
Jerusalem.HK.2880
Jerusalem.HK.2886
Jerusalem.Sunday.A
June_12th
Keypress.1232.A
Liberty.2857
Line
New or modified variant of Little_Red
MacGyver.2824.A
Mange_tout.1099
Ming.1017
Ming.491
Ming.CLME.1528
Ming.CLME.1952
NRLG.1030 - Generation 1
NRLG.776 - Generation 1
NRLG.992 - Generation 1
Ping-Pong.Standard
Possibly a variant of Devil's_Dance
Quartz
Sampo
Sblank
Shatin
Shutdown.644
Shutdown.698
Stoned.Dinamo
Stoned.Empire.Monkey.D
Stoned.Flame
Stoned.NoInt.A
Stoned.Standard
Timid.305
TV
Vienna.648.Reboot.A
Yankee_Doodle.TP.44
Yankee_Doodle.TP.44.E

China

Beijing
_1492
Changsha
Democracy.3806
Green_Caterpillar.1575.I
HideNowt
New or modified variant of Little_Red
Mange_tout.1099
Specified
Uestc

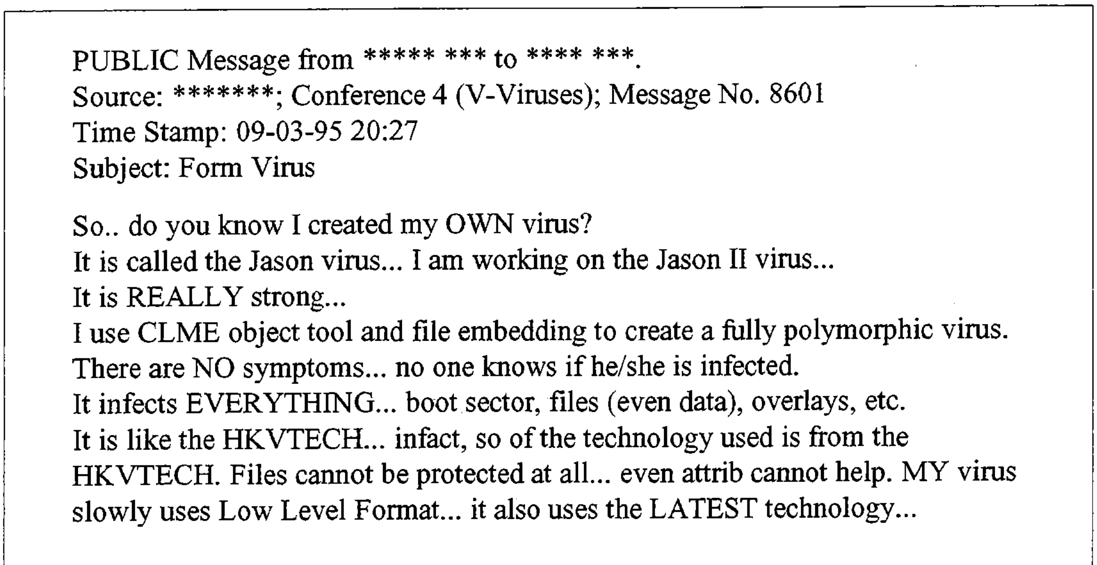
Shenzen

AntiCMOS.B

BBSs AND THE INTERNET

In Hong Kong, local phone calls are free, and, as HK is a small territory, everywhere is local. Consequently, there are large numbers of Bulletin Boards. However, international calls are expensive. A recent change is in access to the Internet. The first Internet Service Provider opened in 1994, and interest has rocketed in the last six months, whether this vastly cheaper method of reaching international sites will result in a wider range of viruses in Hong Kong zoos remains to be seen.

On the BBSs, there is a well-established Virus Echo, which is unmoderated. Most of the messages deal with combating viruses, but some are from self-proclaimed virus authors (fig. 1 is part of a typical message). There are hints in these messages that virus exchange BBSs exist, and some indication that there might be one or more virus writing groups, but I have been unable to verify this.



PUBLIC Message from ***** *** to **** ***.
Source: *******; Conference 4 (V-Viruses); Message No. 8601
Time Stamp: 09-03-95 20:27
Subject: Form Virus

So.. do you know I created my OWN virus?
It is called the Jason virus... I am working on the Jason II virus...
It is REALLY strong...
I use CLME object tool and file embedding to create a fully polymorphic virus.
There are NO symptoms... no one knows if he/she is infected.
It infects EVERYTHING... boot sector, files (even data), overlays, etc.
It is like the HKVTECH... infact, so of the technology used is from the HKVTECH. Files cannot be protected at all... even attrib cannot help. MY virus slowly uses Low Level Format... it also uses the LATEST technology...

Fig. 1: Part of message from a 'virus writer' on HK BBS virus echo

The virus writers take advantage of the BBS rules to spread their creations. Many of the BBSs specify an upload-download ratio, and check uploaded files against files already on the BBS. Those who upload new versions may also be awarded with extra privileges. As soon as a new version appears on one BBS, users will race to be the first to upload it to all the other BBSs they know. The virus writer merely has to infect a well-known package, give it a new version number and upload it to one or two BBSs with a tantalising description. The ensuing confusion will obliterate any trail to the instigator.

Some of the virus families which have emerged from the local BBSs are:

JERUSALEM.HK

The first in this family was Jerusalem.HK.2358. It appeared in March 1994. It displayed the text string 'HKs Vtech', which led to it being called Jerusalem.Vtech. This was later changed to Jerusalem.HK because a company manufacturing computers and other consumer electronics in Hong Kong is called Vtech. There is no connection between the company and the Jerusalem.HK family of viruses.

Jerusalem.HK.2880 appeared in May 1994, and .2886 and .2513 soon followed. These were initially distributed using the 'Trojanised new version upload' method. By June, reports of Jerusalem.HK.2880 were circulating Dutch BBSs. This is certainly a distribution method that can be very effective.

The .2880 and .2886 variants, after displaying 'HKs Vtech', also say 'using VTME v1.11' and 'Please ask S-S Chan to help you!'. These are polymorphic, and it seems likely that VTME is the name of the mutation engine developed for this family.

MING

Ming.1017 was first found in January 1994 and contains the text message: 'Nice To Meet You! Copyright(c) 1-11-1993 By Ming. From Tuen Mun, Hong Kong Version 3.10'.

It is a simple, overwriting file virus and is only notable because a minor variant reached Lapland two weeks later on video driver diskettes which had originated in Hong Kong. Even something as obvious as an overwriting virus can spread internationally with a lucky break.

In April, messages appeared on the local virus echo claiming to be from the Ming who had written this virus. He said that he was changing his name to Crazy_Lord and he that had written CLME, the Crazy Lord Mutation Engine. Different messages mentioned versions 0.5 and 0.6. He also thanked 'HKs Vtech' for assistance and challenged 'S-S Chan' to kill it.

It was July before a sample of a CLME virus was obtained. The decryptor used a simple XOR loop with a variable amount of junk code.

Ming.CLME.1952, Ming.1017 and Ming.CLME.1528 increased in samples uploaded to my BBS during the following months and are still occasionally uploaded now. A total of five Ming viruses are currently known.

The author appears to be a novice programmer, improving his skills with practice. One can only guess at the collaboration between this person and the author of the Jerusalem.HK family.

SHUTDOWN

Shutdown.644 appeared in January 1994. It contains the text message:

> Computers must be shut down to dedicate my sister

Reports of it and a second variant, .698, have appeared sporadically in the BBSs until recently.

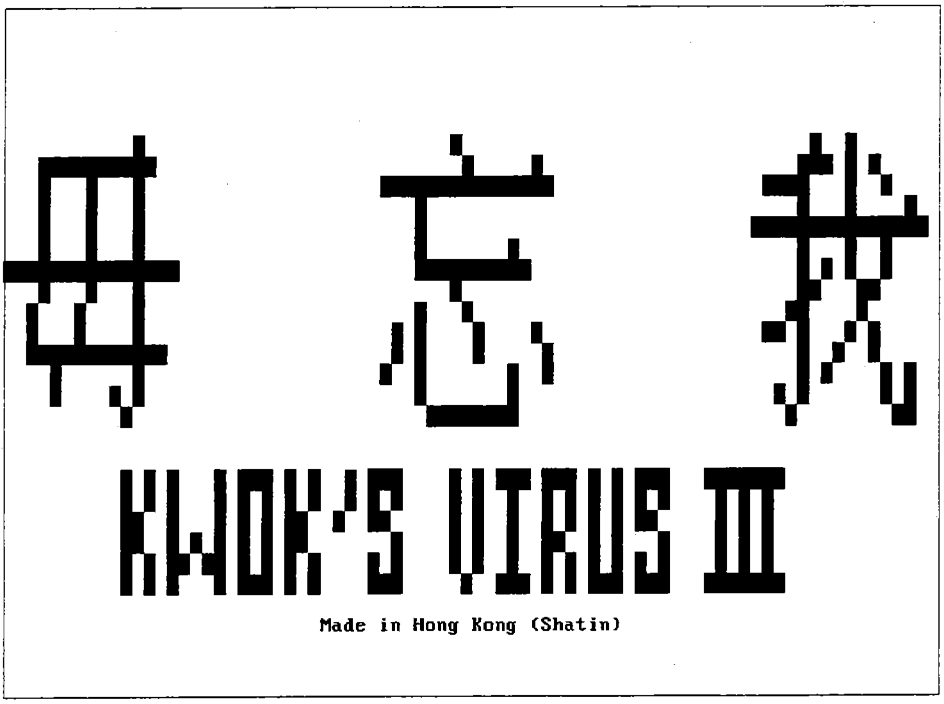
CHINESE LANGUAGE VIRUSES

Two other viruses known in Hong Kong BBSs are notable because they display, or attempt to display, a message in Chinese characters.

Jerusalem.J, which first appeared in October 1994, uses DOS calls to redefine four characters as two Chinese characters 'Death God'. The display routine fails on some video cards. Interestingly, in November 1994, a message complaining that it had been named incorrectly, and was actually called 'Jerry virus' was posted in the local virus echo. The writer of the message claimed it was not related to Jerusalem because it 'only has 30 bytes like Jerusalem'.

Shatin, which first appeared in Febraury 1995, takes a different approach to displaying Chinese characters. It uses block graphics to make the Chinese characters for 'Forget me not' as a full screen display (fig. 2). All these viruses have been widely reported in the local virus echo, but there have been no local reports from commercial sites.

I had hoped to give a detailed profile of Hong Kong virus writers, but firm evidence has not surfaced. The impression from numerous BBS messages is of a small number of actual writers, probably adolescent males, and a larger number of 'wannabe' virus-writers and hangers-on. One correspondent at different times claimed to be a virus writer and to know a virus-writing group, but not be a writer himself. He claimed to be concerned about what the group would do to him if he revealed what he knew. I could not work out if he was trying to fool me, or if he was really worried, or even if the messages originated from one or more people.



毋 忘 我

KWOK'S VIRUS III

Made in Hong Kong (Shatin)

Fig. 2: Screen shot of Shatin

THE ROUTE FROM CHINA

The commercial companies have more international interchange.

Literally thousands of Hong Kong companies have offices or factories in China. It is easy to imagine that this provides a channel for viruses written in China to reach the international scene. Two specific incidents suggest, but do not prove this:

ANTICMOS

In March 94, within a couple of weeksof each other, I received two samples of AntiCMOS.A. One was from an airline, the other a manufacturing company. Both believed that the virus had come from their China branches. These were not the first samples of AntiCMOS found internationally, but they were close to that date.

The next stage in the chain is a company without an anti-virus policy. In April 94, I received a sample from a large local bank. In May, a salesperson for an on-line banking service from that bank was found to have an infected demonstration diskette. The diskette in question was not write-protected, and the salesperson's standard procedure was to use a customer's machine to do a directory of the diskette, and then to boot the machine from it. This procedure seems designed to maximise the chances of picking up an infection and distributing it to the maximum number of customers. Since then, reports of AntiCMOS have risen.

I have had only one sample of AntiCMOS.B, in early May 94. This was collected from a computer company in Shenzen. This variant contains the text: 'I am Li Xibin'. Li Xibin is a plausible Chinese name, that and the dates of first appearance suggest that AntiCMOS is a Chinese virus.

MANGE_TOUT.1099

This was first found in January 1994 in a manufacturing company with a branch in China (as labour costs rose in Hong Kong, many companies moved their factories to China, and kept their head office in Hong Kong). Again, the possibility that this virus had travelled from the Chinese branch was mentioned. It spread in Hong Kong in the subsequent months and a trip to Beijing at the beginning of August 94 showed it to be well known there. Several samples were received, and local anti-virus software detected it as DA01. Near the end of August, it leaped to Norway via video device driver diskettes which had been duplicated in Hong Kong.

CHINA

China itself requires a lot more research, but of the ten viruses found, six have not been seen elsewhere. Of the others: AntiCMOS.B was found almost simultaneously in China and elsewhere; Mange_tout we have already mentioned; Hidenowt was found in Hong Kong soon after its original discovery (Chinese anti-virus software recognises it with the name of 'Dong'); and Changsha contains a name and address in China.

A perception noted in China was that foreign anti-virus software is unable to detect Chinese viruses, this sample shows how that idea could arise. The Chinese anti-virus software mentioned is called Kill and is produced by the Ministry of Public Security. It used to be distributed free, but it is now being sold. The apparent conflict of interest between the same organisation having responsibility for issuing permits to sell security software, and selling its own security product, does not seem to be a concern.

THREE ENVIRONMENTS

In Hong Kong, the corporate and hobbyist appear to form two separate environments (fig. 3). The corporate environment has more frequent Chinese and international connections but, for the most part, exchange only data diskettes. Thus, boot sector viruses are dominant and their prevalence assisted by the failure of many corporations to take the simplest of precautions.

Fig. 2: Major and Minor Routes of Virus Spread in Hong Kong and China

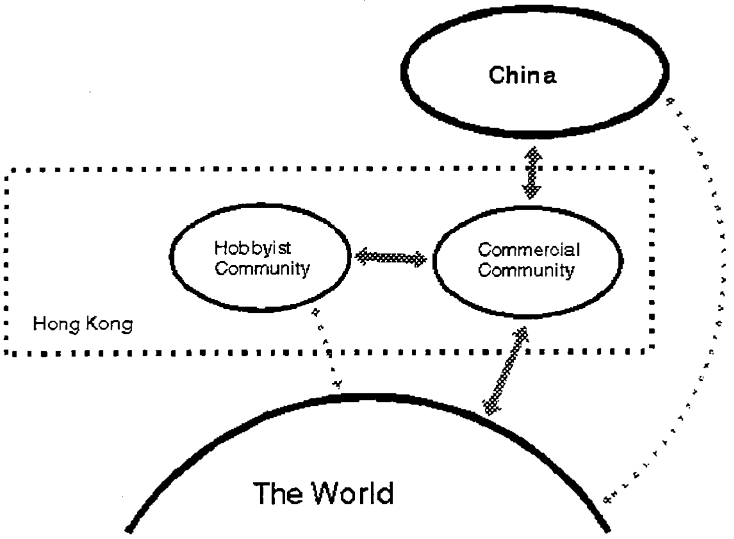
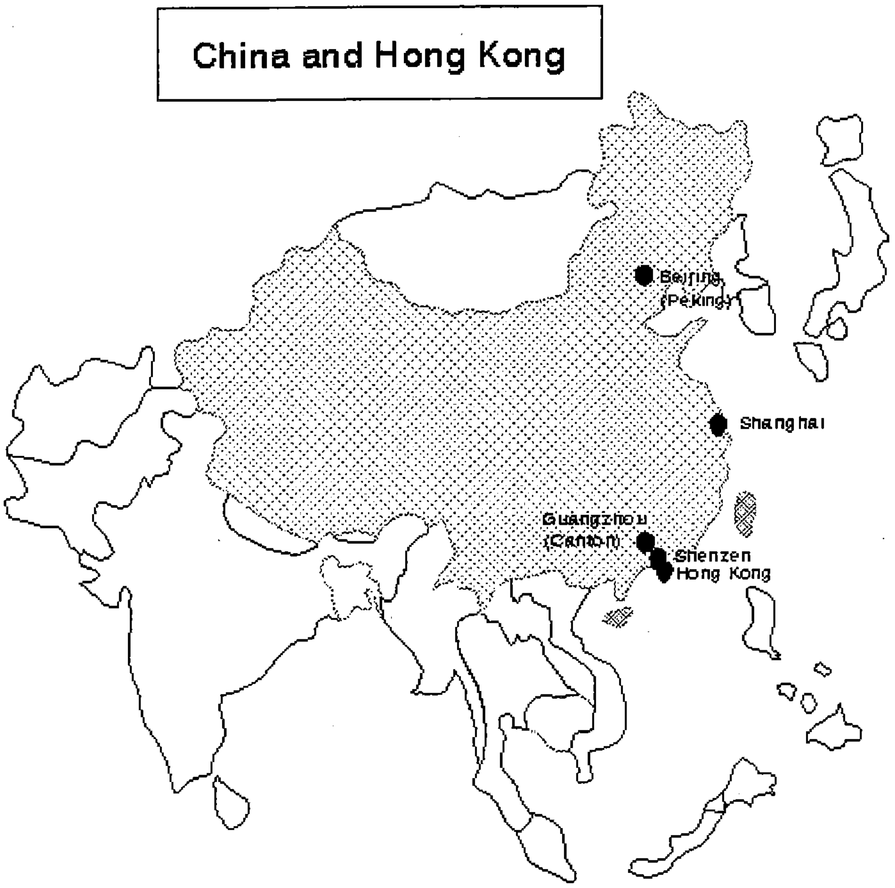


Fig. 3: Major and minor routes of virus spread in Hong Kong and China

The hobbyist environment frequently exchange files on BBSs, but these will generally be entirely within Hong Kong. So, file viruses are common, and locally written viruses are well represented, but some viruses, well-known internationally, are missing. This may change as the Internet boom allows exchange between previously isolated collectors.

China is largely isolated, but has its own virus writers. Who they are, and why they write, is currently unknown. However, it seems certain that we will encounter more from China as it develops and increases its links.



REFERENCES

[1] Dr Matthew K O Lee, 'Information Systems Security: legal aspects', *Hong Kong Computer Journal*, vol. 9 no. 11 pp. 19-22

[2] Title: 18th Feb., 1994 PRC NC no. 147 issue, available at Law-on-Line, URL:http://lawhk.hku.hk

CASE STUDY OF VIRUS CONTROL IN A LARGE ORGANISATION

THE UNITED NATIONS PEACE FORCES APPROACH TO VIRUS CONTROL

Lucijan Caric

Information Technology Services Section, United Nations Peace Forces, PO Box 870, Zagreb, Croatia
Tel +385 1 180394 · Fax +385 1 176288 · Email Lucijan_Caric_at_DPKO-UNPF@un.org

Phillip D. Kruss

Information Technology Services Section, United Nations Peace Forces, PO Box 870, Zagreb, Croatia
Tel +385 1 180327 · Fax +385 1 176288 · Email Phillip_Kruss_at_DPKO-UNPF@un.org

ABSTRACT

The United Nations Peace Forces (UNPF) operation in the former Yugoslavia includes a significant information technology (IT) component, with an IT Services Section (ITSS) staff of 80 personnel, several networks and 4,000 PCs. Despite initial similarities, the environment under which ITSS must control viruses is not a typical corporate one. There are fresh injections of viruses on a regular basis because each of the many troop contingents rotates every six months. Also, a high percentage of the PCs are standalone and located in war-torn locations that are difficult to access on a regular basis in order to update virus scanners.

Prior to the implementation of a comprehensive anti-virus program, the number of virus incidents had reached a critical level. Hence, all measures which could be brought to bear were utilized, including BIOS-in-built features, scanning, and the use of cryptographic checksums. It was also clearly necessary to develop a system which would defend against attack from users (e.g. changes to the DOS environment), which could cause as much damage and downtime as viruses, and also lead to the disabling of the virus protection. The constraints to the approach were software cost, maintenance cost, and the need to reduce the annoyance to the user to a minimum (the greater the impact on the user – such as long boot process – the more likely the user was to attempt to disable the virus protection).

An analysis of the virus population at large in UNPF at that time showed that 90% of infections were due to boot sector viruses, and therefore an initial emphasis was placed on the elimination of these viruses. Many of UNPF's computers possess so-called built-in 'Boot Sector Virus Protection', but, for a number of reasons, this was found to be totally ineffective. However a BIOS-configuration solution was found and implemented which drastically reduced the incidence of boot sector viruses and as an added bonus reduced the 'annoyance factor' by speeding up the boot process. This solution has often been overlooked.

In addition to the above, both virus scanning and verification of cryptographic checksums were introduced. This implementation covered memory and the most critical files, including those files which do not come under virus threat, but are subject to user attack; the checking of files subject to user attack was implemented to prevent deterioration of the DOS/Windows environments and disabling of the virus protection itself. In the execution of these measures, it was found that displaying a message to the user regarding the detection of a virus was simply not effective. Hence, on detection of a virus or a user-instigated change to vital system files, locking of the system was introduced so that an infected computer could no longer be used.

The end result of this exercise was that, at very low cost, the computers were very effectively protected against known and unknown viruses, and also against degradation of the DOS and Windows environments due to inadvertent or intentional attack by users.

Despite a continuing significant increase in PCs, and regular injections of virus-infected disks by incoming troops, the number of virus infections of PCs at UNPF has been dramatically reduced.

INTRODUCTION

One major task before the Information Technology Services Section (ITSS) of the *United Nations Peace Forces* (*UNPF*) in the former Yugoslavia at the beginning of the 1994 was the necessity to develop a standard and reliable user environment for the standalone PCs used within the organization. A primary criterion in this regard was to design an operating system environment which would prevent degradation at both the DOS and Windows levels. At the same time, the number of virus infections had reached a critical level, counting more than 15 separate incidents per month. A decision was taken to address both of these problems with one solution.

In the early stages of any *United Nations (UN)* mission, virus infections always present a difficult problem because many computers are moved rapidly into areas out of reach of any centralised information technology (IT) section. Further, there is a communications problem, which results in widespread use of diskettes for the dissemination of information. In addition, the rapid arrival of thousands of civilians and military personnel and the rotation of troops every six months results in a very regular inflow of viruses into the mission area.

There are currently over 4,500 civilians and 40,000 troops in the former Yugoslavia. The total *UN* PC count provided by the *UN* is over 4,000, although many nations, as well as other *UN* and non-governmental agencies, also possess their own PCs. A high percentage of PCs are standalone and are located in war-torn areas that are difficult to access on a regular basis to update virus scanners.

In designing the user environment, we were aware of the problems caused on machines by the users. Installing unauthorised software packages, either intentionally or unintentionally, changing the configuration of the DOS/Windows environments and introducing virus infected files and disks onto the PC were common actions. It was decided to design an environment which would be resistant to change and which would prevent operation of a PC whose configuration was significantly modified. At the same time, that environment needed to ensure trouble-free operation of the PCs, while providing the user with all necessary application software. It should also cause no virus false alarms or irritate users with long lasting and frequent system checks.

THE DESIGN ENVIRONMENT

When anti-virus measures were planned, ITSS was responsible for over 1000 PCs, with more than 3000 PCs projected to be in place within one additional year. More than half of these PCs were standalones and the rest were connected in several multiple-server networks. Most application software was located on the PCs, with only major applications (e.g. procurement) running as server-based. Since more than half of the PCs were standalones, and this percentage was expected to remain static (there is only so much network penetration that can be accomplished in areas such as Sarajevo), it was clear that it would be difficult to upgrade anti-virus software on any regular basis.

ANTI-VIRUS MEASURES

BIOS SETUP FEATURES

Before any anti-virus software is introduced, it is very important to identify and utilize all possible means of virus prevention present as built-in features of the PC. Most BIOS versions for the PCs used by *UNPF* possess several features which can be used to help prevent a PC from becoming infected. There are three main features which were used:

1 Boot Sequence

The ability to alter the boot sequence is a feature available on essentially all desktop PCs available on the market today. By altering the boot sequence, so that the computer boots from drive C: first, the time required for the system to boot up is shortened and the risk of the computer being infected with the boot sector viruses is greatly reduced since it is impossible for a virus to utilize accidental booting from drive A: in order to infect the PC.

2 Boot Sector Protection

Boot sector protection is another BIOS feature available for anti-virus prevention on today's PCs. The idea behind boot sector protection is to prevent a virus writing itself to the master boot sector (MBS) of the hard disk. If a virus tries to write to the MBS, a message is displayed on the screen warning the user and asking for confirmation before writing is allowed. Regrettably this virus protection feature is of very limited use because of the several reasons. One major loophole of this type of boot sector protection is that it typically protects only the MBS of the hard disk, leaving the DOS boot sector unprotected. For example, if infection is attempted by the Jack The Ripper virus it will be detected, but if infection is attempted by the Form virus, the user will not be warned and the virus will successfully infect the PC.

A second disadvantage of such boot sector protection is that it relies on the user's judgement and action. The user is typically confronted with a message like: 'Boot sector write – possible virus. Continue (Y/N)?' Most users as a rule choose the easier possible escape and press YES which leads to infection of the PC. It is our opinion that it would be much better if BIOS producers allowed for the possibility of writing to the master boot sector to be authorized from the BIOS setup in the ON/OFF form. In this case, if a write attempt is made to MBS, it would just cause a message to be displayed stating that writing to the MBS is not allowed. In order to write to the MBS, the user would need to change the BIOS setup first. Since writing to the MBS is a rare operation under most operating systems, imposing a more complicated procedure in order to allow somebody to perform it would not unduly complicate anyone's life since the advantages are obvious.

3 BIOS Setup Password

The availability of a BIOS setup password is a standard feature designed to prevent unauthorized access to the BIOS setup. This feature is of very limited use in professional data protection because passwords are easily disclosed by the many BIOS password crackers available today and many boards lose their password protection if the CMOS battery is removed. However, despite these limitations, this feature has been found to be very useful at *UNPF* through the use of a standard password to prevent a user from altering the BIOS setup, especially the boot sequence.

SCANNING

An anti-virus scanner was loaded onto each PC, with at-boot scanning of memory, boot sectors and selected files at various levels depending on boot action (see next section). However, in view of access difficulties as outlined above, and hence reduced update opportunities, the dependence on scanning was minimised. At this point we are not using a resident scanner because it appears that most resident scanners suffer from a lack of detection capabilities and as well cause significant overheads during operation of the PC.

Since in almost all cases the scanner started during booting was successfully detecting infections on the PC (and because the booting sequence was set in such a way as to prevent accidental infections with boot sector viruses and cryptographic checksumming was used to prevent installation of the unauthorised software to the PC), we did not consider the use of a proactive anti-virus component at this time. It was decided that the server based type of proactive anti-virus software would be considered for use and tested later. This process is now completed and all networked PCs will be protected with a server based anti-virus system.

CRYPTOGRAPHIC CHECKSUMS AND LOCKING

In order to strengthen the protection against unknown viruses, cryptographic checksumming was added to the virus protection offered by the scanner. The use of cryptographic checksums is the only possible way to discover all viruses, known and unknown and is particularly critical at *UNPF* where the scanners on many PCs cannot be updated regularly. Cryptographic checksumming was applied to the master and DOS boot sectors as well as to all executables for the purpose of virus detection. However, this concept was further extended to test the integrity of the main parts of the operating environment, including DOS and Windows configuration files. In other words, cryptographic checksumming was used to detect modifications not only to executable files but to other file types as well, which were not subject to virus attack but which were subject to user attack.

For the purpose of virus detection, the master and DOS boot sectors, all files invoked during the booting sequence, and all executables with the following extensions COM, EXE, SYS, DLL, OV? were checked using cryptographic checksums. For the purpose of detecting user intervention WIN.INI, SYSTEM.INI, PROGMAN.INI, AUTOEXEC.BAT, CONFIG.SYS and all .BAT files were checked.

A customized installation floppy-disk was designed and batch files provided for automated use. The creation and encryption of checksums is carried out through batch files on floppy-disk provided only for use by authorized ITSS staff. Users have no means of computing checksums on their computers. In addition, six DOS and Windows configuration files were automatically copied as part of the installation process for backup and recovery purposes.

Checking is performed automatically each time the computer boots, Windows is started or Norton Commander, Login or Logout are executed. If an integrity violation is discovered, the user is notified. If system configuration files have been modified, then, depending on which files have been modified, they will either be automatically restored from backup and control returned to the user, or the system will be halted and user will be advised to call the Help Desk. If executables are found to be modified, the system is halted.

CHECKING SEQUENCES

On first power up

Memory and hard disks are scanned for the presence of known viruses. Cryptographic checksums of all executable files are checked. If the integrity of any file has been violated, the computer is halted. This full check is performed only once a day on first power up.

Every time computer boots

The computer's memory is scanned for the presence of known computer viruses. Integrity of WIN.INI, SYSTEM.INI and PROGMAN.INI files are checked. If the integrity check fails then the original of each file is restored from backup, with modified files being stored for analysis. The integrity of NC.MNU (Norton Commander Menu) file is checked. If the integrity check fails, the original file is restored from backup. The modified file is stored for analysis. The integrity of all files invoked during booting is checked and if the integrity of any file used during booting has been violated, the computer is halted.

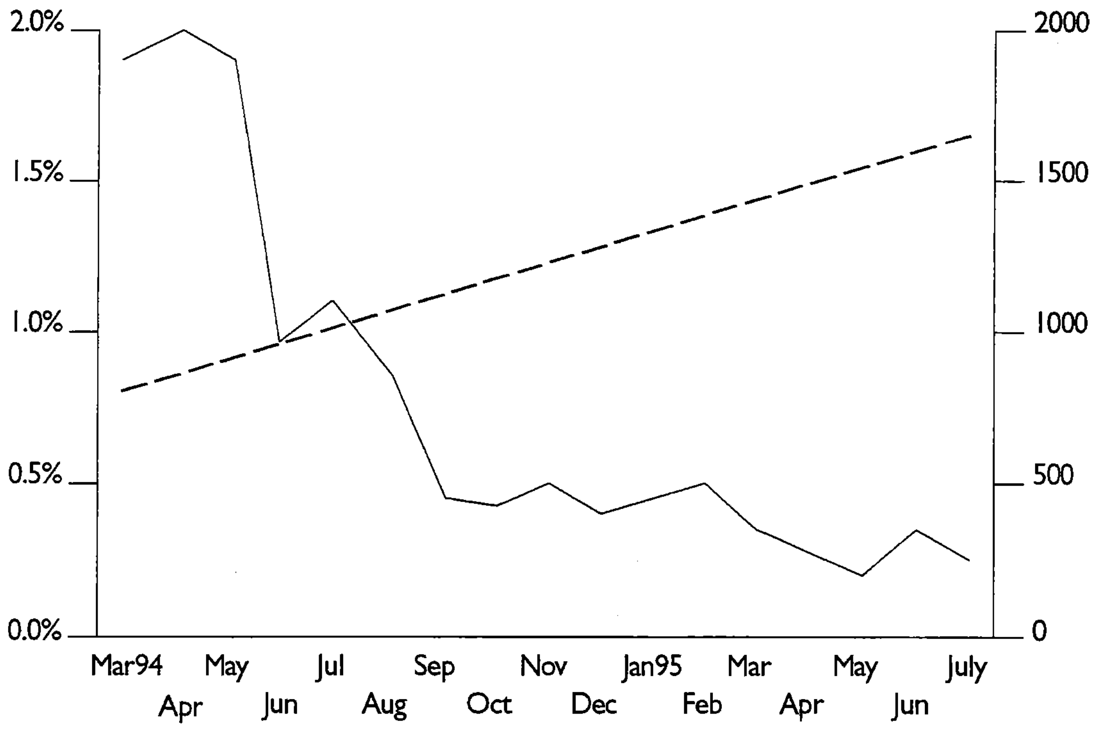
Every time Windows is started or Norton Commander executed

The integrity of AUTOEXEC.BAT and CONFIG.SYS is checked and if the integrity of either of these files has been violated they are restored from the backup. Modified files are stored for analysis.

ADVANTAGES

During the implementation and operation of the above measures, several objectives were reached. With the basic features implemented to prevent accidental booting from infected floppy-disks, over 90% of viruses present in the environment were covered. Implementation of cryptographic checksumming covered the problem of unknown viruses was proved worthwhile by detecting two previously 'unknown' viruses at that time (OneHalf-3544 and SillyCOM-290). At the same time, the boot process was speeded up and checking procedures were causing little or no delay. Finally, operating system setup was protected against tampering and degradation.

RESULTS



The graph shows the very rapid decrease in infections soon after the above measures were introduced (graph covers Zagreb sites only). The graph shows the percentage of PCs infected during a given month (solid line, left axis) as well as the number of PCs in use in Zagreb (dashed line, right axis). Over time, as the number of PCs used within the organization grew, the number of virus infections showed no increase. Maintenance was straightforward and effective since the computer setup was standardized and protection procedures were able to identify problems. The number of false positives was very low and detection accurate and effective.

CONCLUSIONS

The best anti-virus protection on a standalone PC is useless if disabled by the user, something which users tend to do in an environment where control is difficult due to lack of access.

Messages to users regarding possible detection of a virus are essentially useless – some disabling action is necessary, within our case, any infected machine being locked to prevent use.

Changing the boot sequence to C: then A: and subsequently password-protecting the BIOS setup is a very effective way of reducing the spread of boot-sector viruses in an environment where floppy-disks are in significant use.

Cryptographic checksum techniques designed for anti-virus purposes can be effectively used to limit users ability to modify Windows and DOS configuration files, thus enhancing environment stability and reducing maintenance costs.

In environments where policy enforcement is difficult, anti-virus techniques cannot be effective unless they are married with techniques for the preservation of the operating system environment, including anti-virus measures themselves – users tend to disable anti-virus measures.

The annoyance factor of any suite of anti-virus measures must be reduced as far as possible to reduce the likelihood of user attack against those measures.

COMPUTER VIRUSES: A GLOBAL PERSPECTIVE

Steve R. White, Jeffrey O. Kephart and David M. Chess

High Integrity Computing Laboratory, IBM Thomas J. Watson Research Center, P.O. Box 704,
Yorktown Heights, NY 10598, USA
Tel +1 914 784 7368 · Fax +1 914 784 6054 · Email srwhite@watson.ibm.com

1 INTRODUCTION

Technical accounts of computer viruses usually focus on the microscopic details of individual viruses: their structure, their function, the type of host programs they infect, etc. The media tends to focus on the social implications of isolated scares. Such views of the virus problem are useful, but limited in scope.

One of the missions of *IBM's* High Integrity Computing Laboratory is to understand the virus problem from a global perspective, and to apply that knowledge to the development of anti-virus technology and measures. We have employed two complementary approaches: observational and theoretical virus epidemiology [1, 2, 3, 4, 5, 6]. Observation of a large sample population for six years has given us a good understanding of many aspects of virus prevalence and virus trends, while our theoretical work has bolstered this understanding by suggesting some of the mechanisms which govern the behavior that we have observed.

In this paper, we review some of the main findings of our previous work. In brief, we show that, while thousands of DOS viruses exist today, less than 10% of these have actually been seen in real virus incidents. Viruses do not tend to spread wildly. Rather, it takes months or years for a virus to become widespread, and even the most common affect only a small percentage of all computers. Theoretical models, based on biological epidemiology, can explain these major features of computer virus spread.

Then, we demonstrate some interesting trends that have become apparent recently. We examine several curious features of viral prevalence over the past few years, including remarkable peaks in virus reports, the rise of boot-sector-infecting viruses to account for almost all incidents today, and the near extinction of file-infecting viruses. We show that anti-virus software can be remarkably effective within a given organization, but that it is not responsible for the major changes in viral prevalence worldwide. Instead, our study suggests that changes in the computing environment, including changes in machine types and operating systems, are the most important effects influencing what kinds of viruses become prevalent and how their prevalence changes.

Finally, we look at current trends in operating systems and networking, and attempt to predict their effect on the nature and extent of the virus problem in the coming years.

2 THE STATUS OF THE VIRUS PROBLEM TODAY

Over the past decade, computer viruses have gone from being an academic curiosity to a persistent, worldwide problem. Viruses can be written for, and spread on, virtually any computing platform. While there have been a few large-scale network-based incidents to date [7, 8, 9, 10] the more significant problem has been on microcomputers. Viruses are an ongoing, persistent, worldwide problem on every popular microcomputing platform.

In this section, we shall first review briefly our methods for monitoring several aspects of computer virus prevalence in the world. Then, we shall present a number of the most interesting observations. We will attempt to explain these observations in later sections of the paper.

2.1 MEASURING COMPUTER VIRUS PREVALENCE

We have learned much about the extent of the PC-DOS virus problem by collecting virus incident statistics from a fixed, well-monitored sample population of several hundred thousand PCs for six years. The sample population is international, but biased towards the United States. It is believed to be typical of Fortune 500 companies, except for the fact that central incident management is used to monitor and control virus incidents.

Briefly, the location and date of each virus incident is recorded, along with the number of infected PCs and diskettes and the identity of the virus. From these statistics, we obtain more than just an understanding of the virus problem within our sample population: we also can infer several aspects of the virus problem worldwide. Figure 1 illustrates how this is possible[1].

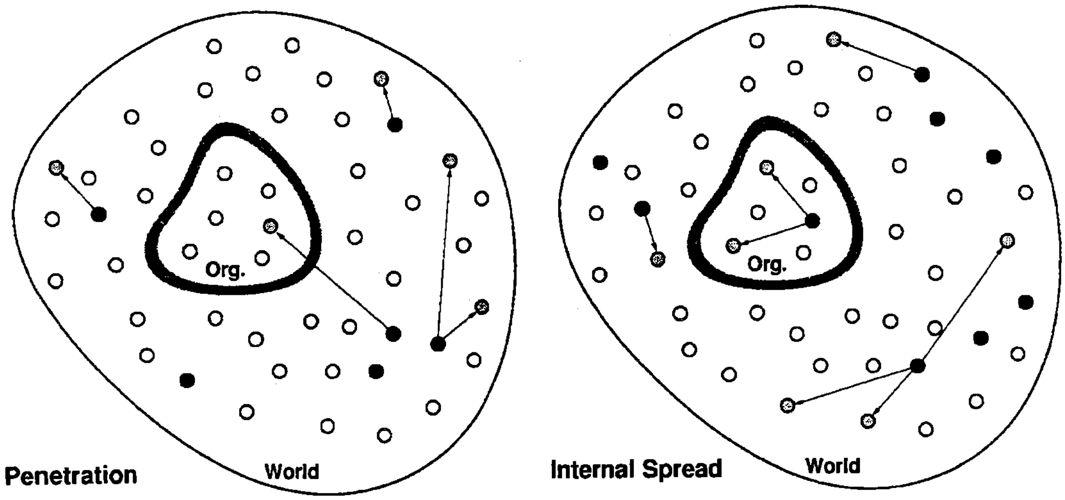


Figure 1: Computer virus spread from an organization's perspective. White circles represent uninfected machines, black circles represent infected machines, and gray circles represent machines in the process of being infected. Throughout the world, computer viruses spread among PCs, many of them being detected and eradicated eventually. Left: Occasionally, a virus penetrates the boundary separating the organization from the rest of the world, initiating a virus incident. Right: The infection has spread to other PCs within the organization. The number of PCs which will be infected by the time the incident is discovered and cleaned up is referred to as the site of the incident.

[1] Further details about our methods for collecting and interpreting statistics can be found in several references [2, 4, 5, 6].

From the perspective of one of the organizations which comprises our sample population, the world is full of computer viruses that are continually trying to penetrate the semi-permeable boundary which segregates that organization from the external world. At a rate depending on the number of computer virus infections in the world, the number of machines in the organization, and the permeability of the boundary, a computer virus will sooner or later make its way into the organization. This marks the beginning of a *virus incident.* Assuming that the permeability of the boundary remains constant, the number of virus incidents per unit time per machine within the set of organizations that makes up our sample population should be proportional to the number of computer virus infections in the world during that time period (in fact, our measure will lag the actual figure somewhat, since incidents are not always discovered immediately).

2.2 OBSERVATIONS OF COMPUTER VIRUS PREVALENCE

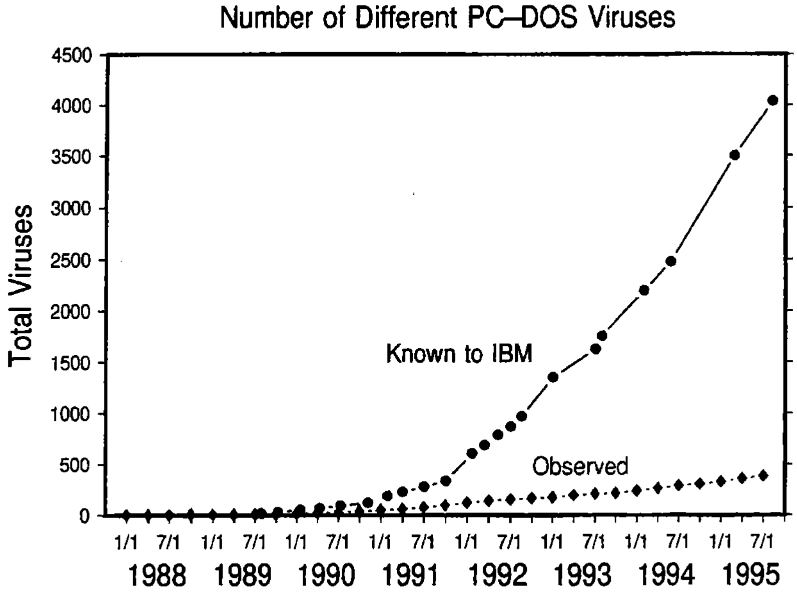


Figure 2: Cumulative number of viruses for which signatures have been obtained by IBM's High Integrity Computing Laboratory vs. time. There are thousands of viruses, but only a few have been seen in real incidents.

As shown in Figure 2, there are thousands of DOS viruses today. During the past several years, the rate at which they have appeared worldwide has crept upwards to its present value of 3-4 new viruses a day, on average (see Fig. 3).

Note that the number of new viruses is not 'increasing exponentially', as is often claimed [11, 3]. The rate of appearance of new viruses in the collections of anti-virus workers has been increasing gradually for several years. The number of new viruses is increasing at no more than a quadratic rate. In fact, almost nothing at all about viruses is 'increasing exponentially'. The problem is significant, and it is growing somewhat worse, but prophets of doom in this field do not have good trade records.

While there are thousands of DOS viruses, less than 10% of them have been seen in actual virus incidents within the population that we monitor. These are the viruses which actually constitute a problem for the general population of PC users. It is very important that anti-virus software detect viruses which have been observed 'in the wild'. The remainder are rarely seen outside the collections of anti-virus groups like ours. Although many of them might never spread significantly, viruses which are not prevalent remain of interest to the anti-virus community. We must always be prepared for the possibility that a low-profile virus will start to become prevalent. This requires us to be familiar with all viruses, prevalent or not, and to incorporate a knowledge of as many of them as possible into anti-virus software. We continue to monitor the prevalence of *all* viruses, regardless of how prevalent they are at present.

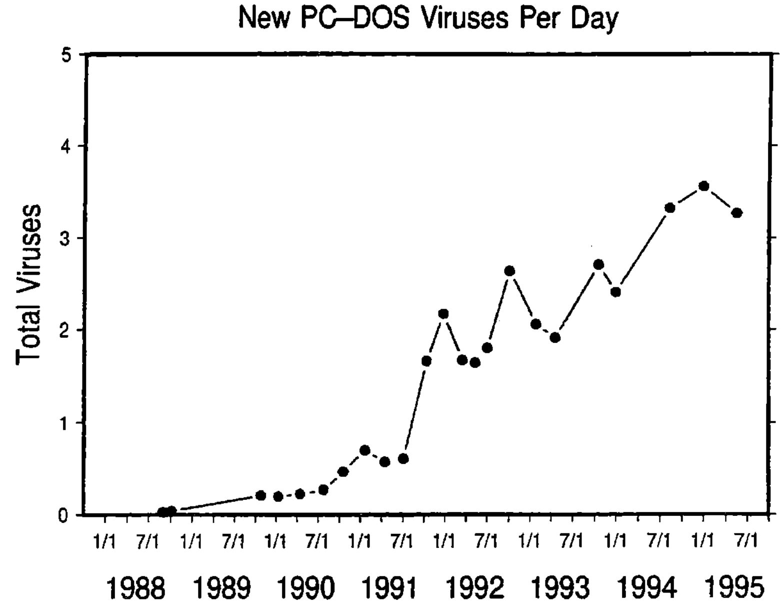


Figure 3: The number of new viruses appearing worldwide per day has been increasing steadily.

Out of the several hundred viruses which have ever been observed in actual incidents, a mere handful account for most of the problem. Figure 4 shows the relative fraction of incidents caused by the ten most prevalent viruses in the world in the past year. These ten account for over two-thirds of all incidents. The one hundred other viruses which have been seen in incidents in the past year account for less than third of the incidents. Most of these were seen in just a single incident.

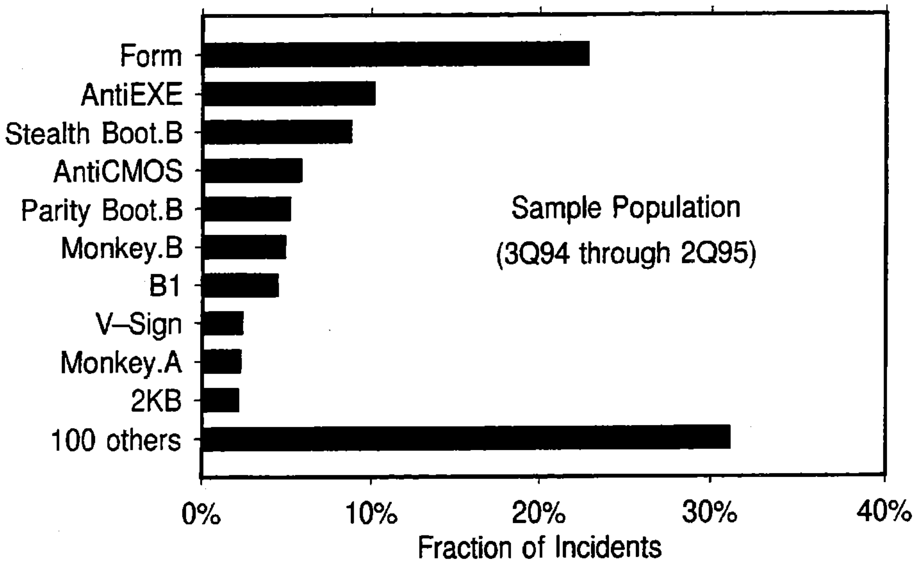


Figure 4: The top ten viruses account for two-thirds of all incidents, and are all boot infectors.

Curiously, the ten most prevalent viruses are all boot viruses. Boot viruses infect boot sectors of diskettes and hard disks. When a system is booted from an infected diskette, its hard disk becomes infected. Typically, any non-write-protected diskette which is used in the system thereafter also becomes infected, spreading the virus. The dominance of boot viruses is especially striking when one takes into account the fact that, of the thousands of known DOS viruses, only about 10% are boot sector infectors.

Boot viruses have not always been dominant. Three years ago, the second and third most prevalent viruses were file infectors, as were 4 of the top 10. The total incident rates for boot infectors and file infectors were roughly equal. Figure 5 provides another view of what has happened to the relative prevalence of these two types of viruses over time. Beginning in 1992, the incident rate for boot sector infectors continued to rise, while the incident rate for file infectors began to fall dramatically. We will attempt to explain this phenomenon in a subsequent section.

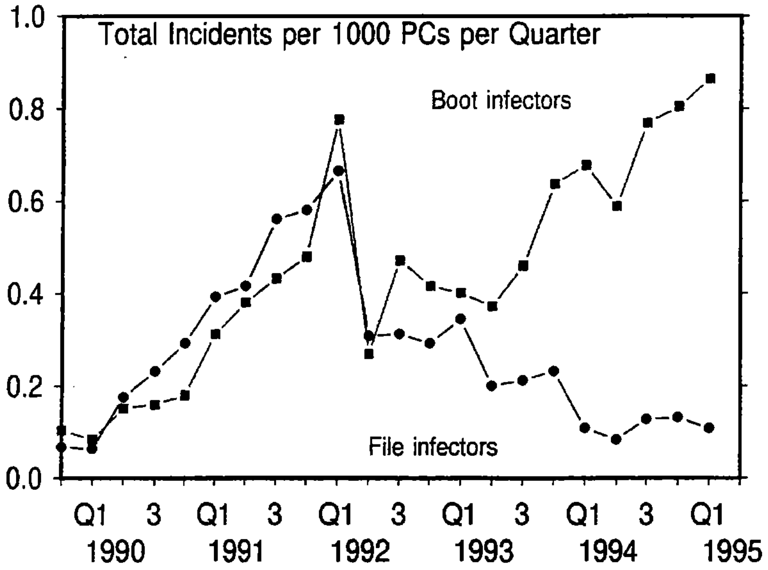


Figure 5: Boot viruses have continued to rise in prevalence, while file viruses have declined.

It is interesting to break up our incident statistics even further into trends for individual viruses. Figure 6 shows the incident rate for selected viruses. Note that some viruses have increased in prevalence, while others have declined.

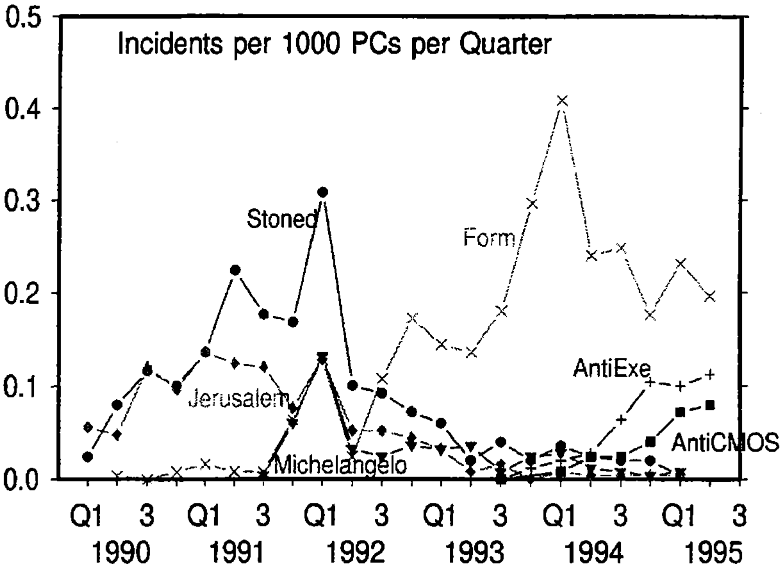


Figure 6: Some viruses have increased in prevalence, while others have declined.

Figures 2-6 raise several important questions:

1. Why are some viruses more prevalent than others?
2. Why do some viruses continue to increase in prevalence, while others plateau or decline?
3. Why are boot viruses so prevalent relative to file infectors, and why has their dominance increased over time?
4. Finally, can we predict what viruses are likely to become more prevalent in the future?

To begin to address these questions, we now review some of our previous theoretical work on virus epidemiology.

3 HOW VIRUSES SPREAD

Over the past several years, we have constructed theoretical models of how computer viruses spread in a population, and compared them against the results of an ongoing study of actual virus incidents [1, 2, 3, 4, 5, 6].

Our models are purposefully simple, in an attempt to understand the most important aspects of global virus spread. In these models, a system is either infected or it is not. If it is infected, there is some probability each day that it will have an infectious contact with some other system in the world, typically via exchange of floppy diskettes, or software exchange over a network. This is called the *birth rate* of the virus. Similarly, there is some probability each day that an infected system will be discovered to be infected. When that happens, it is cleaned up, and it returns to the pool of uninfected systems. This is called the *death rate* of the virus.

The birth and death rates are influenced by a number of factors. A virus' birth rate is governed by its intrinsic properties, such as the particular way in which it infects and spreads. Just as for biological diseases, its birth rate is also highly dependent on social factors, such as the rate of software or diskette exchange among systems. The death rate is determined by how quickly the virus is found and eliminated, which in turn depends on the extent to which people notice the virus, due to its behavior or through the use of anti-virus software. As we shall see, the birth and death rates also depend critically on the nature of the world's computing environment.

All our models show the same basic characteristics of virus spread. One fundamental insight is that there is an *epidemic threshold* above which a virus may spread, and below which it cannot. If the birth rate of a virus is greater than its death rate, the virus has a chance to spread successfully, although it may die out before it spreads much. If the virus does manage to get a foothold, it will start to rise slowly in prevalence. The rate at which it does so is governed by a number of factors, such as intrinsic characteristics of the virus and the overall rate at which software is exchanged. A second fundamental insight which has emerged from our research is that the growth rate can be much slower than the exponential rate that was predicted by one theory [11]. Our theory shows that, when software sharing is localized, the global rate of spread can be very slow, even roughly linear [1, 2]. At some point, the virus levels off in prevalence, reaching an equilibrium between spreading and being eliminated. Figure 7 illustrates the typical behavior of a system above the epidemic threshold.

If the birth rate is less than the death rate – if the virus is found and eliminated more quickly than it spreads – then the virus cannot spread widely. It may spread to a few machines for a little while, but it will eventually be found and eliminated from the population, becoming 'extinct'. Figure 8 illustrates this behavior.

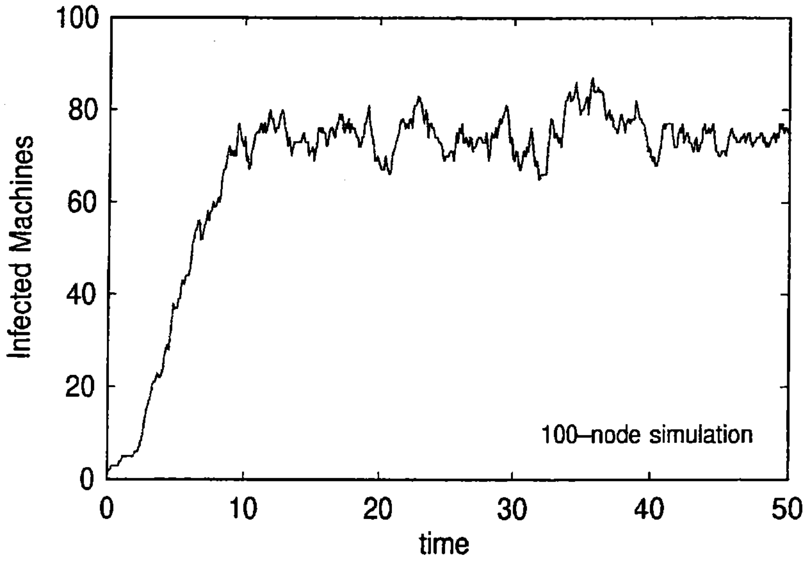


Figure 7: Above the epidemic threshold, a virus rises in prevalence at a rate which depends on a variety of factors, then plateaus at an equilibrium. In this simulation, the birth rate exceeded the death rate by a factor of 5.

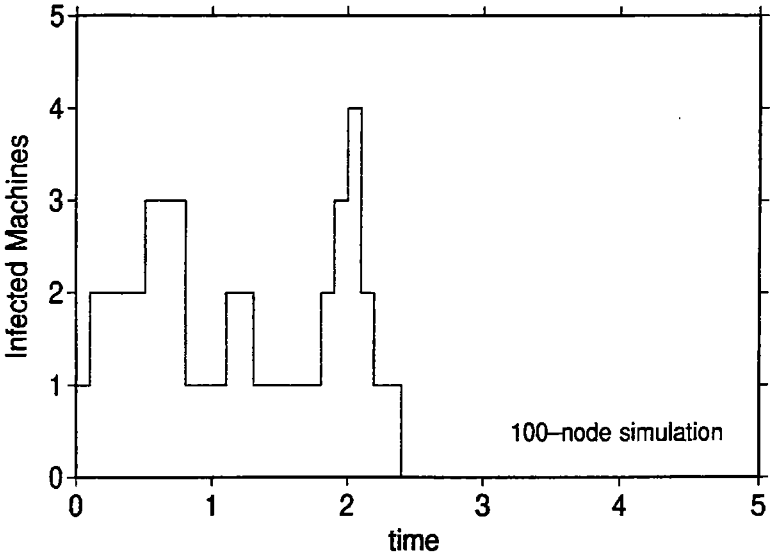


Figure 8: Below the epidemic threshold, very small outbreaks can occur, but extinction of the infection is inevitable. In this simulation, the birth rate was 10% less than the death rate. Note that the vertical and horizontal scales are very different from those of Fig. 7.

4 VIRUS CASE STUDIES

In this section, we illustrate the interaction between viruses and their environment by narrowing our focus to the behavior of selected, individual viruses. We relate changes and shifts in virus prevalence to theoretical findings and to our knowledge of relevant shifts in the computing environment.

4.1 MICHELANGELO MADNESS

The Michelangelo virus was first found in early 1991 in New Zealand. It is a typical infector of diskette boot records and the Master Boot Record of hard disks, with one exception. If an infected system is booted on March 6 of any year, the Michelangelo virus will overwrite parts of the hard disk with random data. This renders the hard disk of the system, and all its information, inaccessible.

The virus is named Michelangelo not because of any messages in the virus itself, but because one of the first people to analyze it noticed that March 6 is the birthday of the famous artist. The name stuck.

Finding a new virus is not unusual in itself; several dozen new viruses are found each week. Michelangelo was unusual in that it was found in an actual incident, rather than as one of the thousands of viruses gathered by anti-virus workers but as yet unseen in an incident. It was also unusual because it could cause such substantial damage to the information on peoples' PCs, and because that damage would all happen on a single day.

In the weeks which preceded March 6, 1992, something even more unusual happened. In a fascinating interplay between the media and some parts of the anti-virus industry, the Michelangelo virus became a major news event. News stories warning about Michelangelo's destructive potential were broadcast on major television networks. Articles about the virus appeared prominently in major newspapers.

As March 6 drew nearer, the stories grew ever more hysterical. The predictions of the number of systems which would be wiped out grew to hundreds of thousands, then millions [12, 13].

When the fateful date came, the predictions of doom turned out to have been a bit inflated. The Michelangelo virus was found on some systems, and probably did destroy data on a few of them. But the worldwide disaster did not occur. Indeed, it was difficult to find any verified incident of destruction of data by Michelangelo in most places [14].

This should not have come as a surprise. Our own research at the time showed that the Michelangelo virus was not very prevalent, and certainly not one of the most common viruses. We estimated that about the same number of systems would have their hard disks crash due to random hardware failures on March 6, as would have their data destroyed by the Michelangelo virus. It is important to keep the risks in perspective.

'Michelangelo Madness', as we came to call it, did have a dramatic effect, though not the anticipated one. Concerned about the predictions of widespread damage, people bought and installed anti-virus software in droves. In some locations, lines of people waiting to buy anti-virus software stretched around the block. In other places, stores sold out of their entire supply of anti-virus software during the week leading up to March 6. Around the world, a very large number of people checked their systems for viruses in those few days.

Figure 9 illustrates the effect of this activity. In the two weeks before March 6, 1992, reports of virus incidents shot up to unprecedented levels. Naturally, this was not because viruses were spreading out of control during those two weeks. Rather, infections which had been latent for days or weeks were found, simply because people were looking for them. In environments like that of our sample population, where anti-virus software is widely installed and used, it is likely that these same infections would have been caught anyway in subsequent weeks. But, since so many people checked their systems prior to March 6, the infections were discovered then rather than later.

People did find the Michelangelo virus, but they found far more viruses of other kinds. The Stoned virus, for instance, the most prevalent virus at the time, was found about three times more frequently than was the Michelangelo virus.

In the first few months after Michelangelo Madness, fewer virus incidents were reported than in the few month before it. This is easy to understand. First, virus incidents were caught earlier than they might have been because everyone was looking. Viruses found in the beginning of March might have been found in the beginning of April instead. So, you would expect fewer virus incidents to be reported shortly after March 6 of that year. Second, viruses were probably found and eliminated even in systems which might not have found them for a very long time. In just a few days, the worldwide population of viruses was decreased. We would expect that the virus population, and hence virus incident reports, would increase again in subsequent months.

Virus incidents did increase after that, but in a way which is rather complicated. We will examine this in more detail in a subsequent section.

Despite the beneficial effects of eliminating some viruses temporarily, the hysteria caused by this event was clearly out of proportion to the risk. Individuals and businesses spent vast sums of money and time warding off a threat which was much smaller than they were led to believe. We hope that those involved learned from the experience – that our friends in the anti-virus industry will be more careful in saying that they understand viral prevalence when they do not, and that the media will examine predictions of impending doom with a somewhat more critical eye.

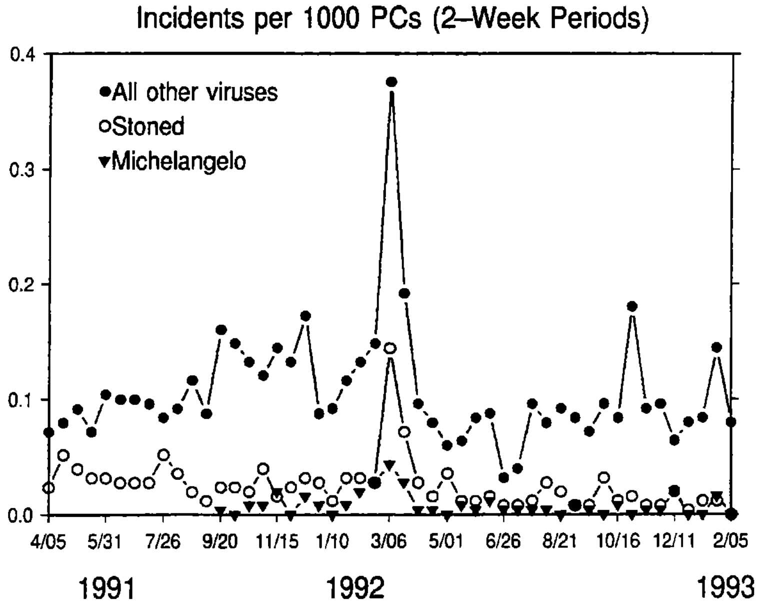


Figure 9: Michelangelo Madness resulted in many people finding viruses of all kinds.

4.2 THE MISSING BRAIN

The Brain virus was first observed in October 1987, making it one of the first DOS viruses seen in the world [15]. It infects diskette boot sectors, and becomes active in a system when that system is booted from an infected diskette. Unlike most boot viruses today, Brain does not infect boot sectors of hard disks.

In the early days of PCs, most PCs were booted from diskettes and did not have hard disks. This provided a perfect medium for Brain to spread. Diskettes used in an infected system became infected themselves, and could carry that infection to other systems. Brain spread around the world in just this way.

Beginning with the introduction of the IBM PC-XT in 1982, the PC industry made a transition to systems which have hard disks. Unlike their predecessors, these systems were not booted from diskettes as frequently. When they were booted from diskettes, it was typically for some special activity, such as system maintenance. Once that activity was concluded, the system was rebooted from the hard disk. It became very uncommon for a system to be booted from a diskette and then used for an extended period of time, with more diskettes being inserted into the system. This denied the Brain virus the opportunity to spread in most cases. The world became a much more difficult place for the Brain virus to spread, and its prevalence declined.

This decline in prevalence occurred before we started gathering accurate statistics about virus incidents, so we cannot illustrate it quantitatively. Anecdotal evidence and our own informal statistics from the late 1980s, however, suggest that the Brain virus was substantially more common than it is today. While Brain is still seen on rare occasions, it does not spread well today. We sighted the Brain virus several times from mid-1988 until mid-1990, but since 1990 it has only appeared in our sample population once, in early 1992.

4.3 NOT STONED AGAIN

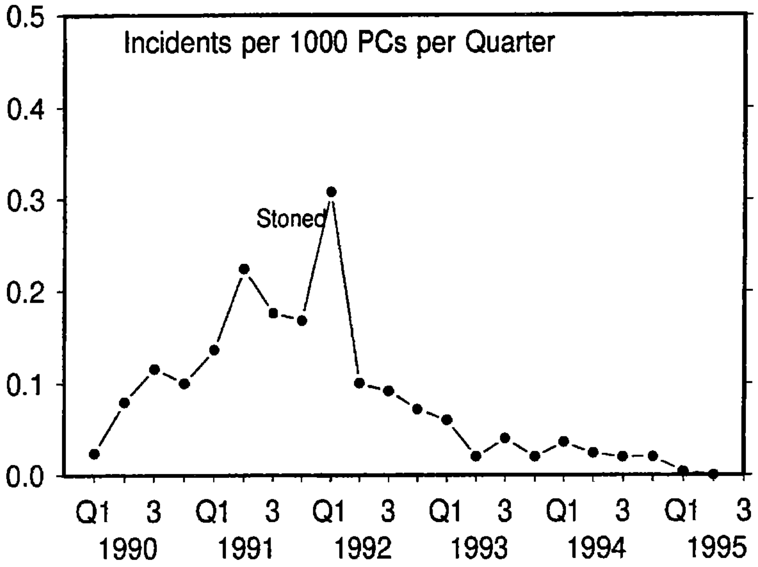


Figure 10: The Stoned virus, a boot infector, rose in prevalence and then declined.

The Stoned virus was first observed in an incident in 1989. It is a typical boot virus, infecting diskette boot records and Master Boot Records of hard disks. One time out of eight that a system is booted from an infected diskette, the message 'Your PC is now Stoned!' will appear on the display. The virus has no other effects.

This virus followed the expected pattern of rising in prevalence through 1991, at which time it had reached a rough equilibrium. After a large peak during Michelangelo Madness, it slowly declined in prevalence over the next several years. Once the most prevalent virus in the world, the Stoned virus is seen much less frequently today.

Its rise in prevalence and subsequent equilibration is what we expect of a virus. Its decline is a bit puzzling at first, until we notice that a system infected with the Stoned virus only spreads that infection to the diskette in the A: drive, not to any other diskette drive. The system became infected in the first place by booting from an infected diskette in the A: drive. The Stoned virus started its life on 5.25-inch diskettes. In spreading from diskette to system to diskette, it could only spread to other 5.25-inch diskettes.

Early in Stoned's life, most systems used 5.25-inch drives, so there was a fertile medium around the world which Stoned could use to spread. In the late 1980s, however, a trend began towards systems that used 3.5-inch drives as their A: drive. The fraction of systems which had 5.25-inch A: drives declined, and has been declining steadily ever since. With fewer and fewer systems that Stoned could infect and spread between, the virus too declined in prevalence.

4.4 JERUSALEM'S RISE AND FALL

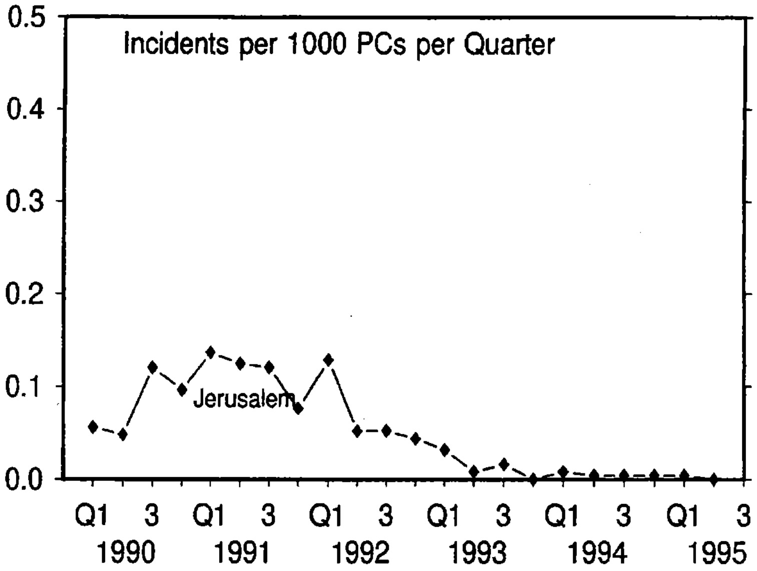


Figure 11: The Jerusalem virus, once quite prevalent, is seen much less often today.

The Jerusalem virus was first observed in December 1987, in the city of Jerusalem, Israel [15]. In many ways, it is an archetypical file virus. When an infected program is run, the Jerusalem virus installs a resident extension in DOS. Subsequently, when any other program is executed, the virus' resident extension will infect the program file.

Prior to 1992, the Jerusalem virus followed the expected pattern of a virus which is spreading around the world. It rose gradually in prevalence through 1990. At the end of 1990, it had reached an equilibrium level in most of the world. Through 1991, it maintained this same level of prevalence, neither increasing or decreasing.

After 1991, however, an odd thing happened. Fewer and fewer incidents of the Jerusalem virus occurred. What was one of the most prevalent viruses in 1990 declined to one of the least prevalent viruses in 1995. Indeed, we saw only five incidents of the Jerusalem virus in our sample population in 1994, and just a single incident so far in 1995.

What caused this decrease? It was not a change in diskette drive type, or the move from floppy diskettes to hard disks. File viruses like the Jerusalem virus spread to files on any kind of diskette, and persist in systems that boot from hard disks. We will return to the cause of this mysterious decrease in a subsequent section of this paper.

4.5 FORM FOLLOWS FUNCTION

The Form virus was first observed in an incident in the second quarter of 1990. It infects diskette boot sectors and system boot sectors of hard disks. When the system is booted from an infected diskette or hard disk, the virus becomes active in memory and infects essentially any diskette used in the system thereafter.

Unlike the Brain virus, the Form virus remains on the hard disk and can spread if the system is booted from the hard disk subsequently. Unlike the Stoned virus, the Form virus is capable of infecting diskettes of any kind in any diskette drive, so it did not remain limited to one kind of diskette. On the 18th of any month, the Form virus will cause a slight clicking when keys are depressed on an infected system. This is often subtle enough to go unnoticed.

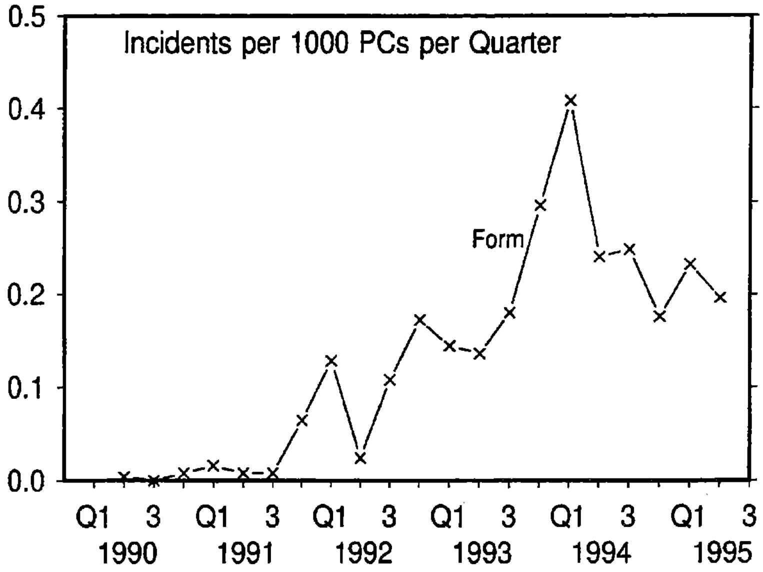


Figure 12: The Form virus, another boot infector, rose steadily in prevalence before reaching equilibrium

The Form virus does not possess the limiting features that caused the Brain and Stoned viruses to have difficulty spreading in the early and middle 1990s. It has exhibited what we expect to be typical behavior for a virus which has found its way into the world. It took over a year before it started rising significantly in prevalence. It rose steadily during 1992 and 1993, becoming the most prevalent virus worldwide. By the end of 1994, it had reached a rough equilibrium at about the same level as other mature viruses such as Jerusalem or Stoned. In the absence of environmental change, we might expect the Form virus to remain about as prevalent as it is today.

5 WHY ARE BOOT VIRUSES SO COMMON?

Boot viruses are by far the most common viruses today, accounting for nearly 90% of all incidents in the second quarter of 1995. File viruses, on the other hand, have decreased in prevalence. This is a remarkable change. Several years ago, file viruses accounted for around 50% of all incidents. What could be responsible for this dramatic change?

Was it Michelangelo Madness? No. That caused only a temporary depletion of viruses of all kinds. Michelangelo Madness explains the large peak in reported incidents, and the subsequent temporary decrease in incidents. It does not account for the difference in prevalence between boot infectors and file infectors.

Is it due to the increased use of anti-virus software? As anti-virus researchers and producers of anti-virus software, we would certainly like to think so. It is tempting to conclude that anti-virus software has made a difference in the world, given our experience with the sample population, in which we have found that widespread usage of anti-virus software and central incident management substantially reduces the size of incidents within an organization [4, 5, 2, 6]. Unfortunately, a closer look at our own data show that, while anti-virus software and policies can make a real difference within organizations, anti-virus software does *not* seem to have made as much of a difference to the world in general. All of the common viruses have been known for quite some time. All of these are detected, even by older anti-virus programs. If anti-virus software was responsible, we would have expected to see a decline in all viruses. The use of anti-virus software does not account for the difference in prevalence between boot infectors and file infectors.

To find the solution to this mystery, we look once again at changes in the computing environment, rather than events associated with the anti-virus industry. The biggest change in the PC computing environment over the past several years has been the change from the use of native DOS to the use of Windows 3.0 and 3.1. Windows 3.0 was released in 1990, and started to become a popular enhancement to the DOS operating system. Windows 3.1, released in 1992, accelerated this trend. Today, many PCs run Windows 3.1.

How does Windows affect the spread of viruses? Experiments carried out at IBM's High Integrity Computing Laboratory demonstrated that Windows is a fragile environment in the presence of typical file viruses. In many cases, if a file virus is resident in the memory of a DOS system, Windows cannot even start. On the other hand, Windows behaves very differently on a system infected with a typical boot virus. For many boot viruses, an infected DOS system can not only start Windows, but can spread the virus to diskettes from within Windows.

If Windows users get a file virus, Windows will typically be inoperable. This will cause the users to eliminate the virus one way or another, whether or not they realise that the system is infected. They might use anti-virus software. They might send their system out for repair. They might re-install everything from backups. Whatever they do, they will eliminate the virus because they cannot get back to work until they do.

If Windows users get a boot virus, however, they might not notice it at all. Windows will usually start and function as expected. Unfortunately, the virus will typically spread to non-write-protected diskettes that are accessed from within Windows. In this sense, most boot viruses are not affected by Windows, and spread in just the same way whether the user is running DOS or Windows. Unless users have good anti-virus software, they will not usually have any reason to suspect a problem, and hence will have no reason to get rid of the virus.

This environmental analysis led us to predict, in 1994, that boot viruses would continue to increase in prevalence, oblivious to the use of Windows. Similarly, we predicted that file infectors would continue to decrease in prevalence. Furthermore, we predicted that boot viruses that were not then very prevalent would become more prevalent, while few file viruses would [16].

This is exactly what has happened. Figure 5 illustrates the dramatic rise of boot virus incidents over the past several years, and the corresponding dramatic decrease in file virus incidents.

Several boot viruses which do spread from within Windows, including AntiEXE and AntiCMOS, were low in prevalence in 1994 but are now substantially more prevalent. As shown in Figure 6, they are approaching the prevalence of more common boot viruses like Form. Once they increase to this level of prevalence, we would expect them to reach equilibrium and not increase further in prevalence.

6 PREDICTING THE FUTURE

We have come to the surprising conclusion that the world's computing environment has been the primary factor in determining the change in prevalence of computer viruses. It is reasonable to assume that this will continue to be the case for some time.

If this is so, we can get some insight into future problems by examining current trends and the expected changes in the computing environment over the next several years. Some of these changes will tend to decrease viral prevalence, while others will tend to increase it.

If there were no changes in the world's computing environment, we might expect to see current trends continue. File viruses would continue to remain very low in prevalence. Boot viruses which have already reached equilibrium, such as the Form virus, would remain at about the same level of prevalence they have today. Other boot viruses would be expected to start becoming more prevalent, perhaps rising in prevalence until they too reach equilibrium. Since there are several hundred boot viruses, having all of them rise in prevalence to the level reached by Form would result in a huge rise in virus incidents worldwide.

There are, however, some environmental changes which we might expect over the next few years: 32-bit operating systems and networking. These changes could have a significant effect on the virus problem.

6.1 32-BIT OPERATING SYSTEMS

One of the significant environmental changes will be the transition from DOS to 32-bit operating systems for PCs, such as OS/2 and Windows 95. In the next few years, we expect that more and more systems will run 32-bit operating systems in order to better use the increasing power of newer PCs.

IBM's OS/2 is a 32-bit operating system which lets users run DOS, Windows and OS/2 simultaneously. The effects of computer viruses on OS/2 systems is described elsewhere [17]. Boot viruses do not generally spread from within OS/2 itself, though they can spread from systems which have DOS as well as OS/2 installed in separate partitions.

File viruses can often spread to other files when infected programs are run in Virtual DOS Machines (VDM) within OS/2. However, they remain active in the system only as long as the infected VDM is active, which is often only as long as the infected program is running. Some file viruses are likely to not spread in VDMs, simply because of differences between VDMs and DOS. This decreases the rate at which file viruses spread in collections of OS/2 systems [17]. In environments in which OS/2 predominates over DOS, we would expect this to lead to a decline in prevalence of all current DOS viruses.

Microsoft's Windows 95 is a 32-bit operating systems that supports DOS, Windows 3.1 and Windows 95 programs. Recent experiments with a pre-release version of Windows 95 suggest that DOS boot viruses will not in general spread well from Windows 95 systems [18]. File viruses were not tested in these experiments.

Preliminary experiments carried out at the High Integrity Computing Laboratory with a pre-release version of Windows 95 suggest that some DOS file viruses will spread as usual, some might not, and some might cause system problems. In environments in which Windows 95 predominates over DOS, we would also expect this to lead to a decline in prevalence of all current DOS viruses.

Not all of the news is good, however. Viruses can be written for 32-bit operating systems, and the first few such crude viruses have already appeared [17]. These operating systems offer new facilities which viruses can use both to hide and spread. The transition to these newer operating systems will change the virus problem, perhaps significantly, but it will not eliminate it.

6.2 NETWORKING

As more and more systems are connected to local and wide area networks, networks may become a more common medium for viral spread.

Of particular interest is the inclusion of networking capabilities in newer 32-bit operating systems. If people typically configure their systems to take advantage of these capabilities, and if that leads to more program sharing on local area networks, it could also increase viral spread in these environments. Currently, these capabilities are used primarily for workgroup computing rather than wide area networking, so the increased spread will result primarily in larger incidents (affecting an entire workgroup instead of a single PC), rather than a large increase in worldwide prevalence.

The final trend which bears watching is the rise of the Internet and global computing. This has the ability to increase the virus problem substantially over time.

There have been incidents of DOS viruses being transmitted on the Internet. Sometimes, they are posted to Internet newsgroups, which function much like bulletin board systems for anyone on the Internet. When the infected programs are downloaded and run, they can infect your PC just like any other infected program. So far, vigilance and rapid action have spread the word about infected programs in newsgroups quickly, and eliminated the problems as they have occurred.

The Internet can be used to support wide-area file servers. These are much like file servers on a LAN, but they can be accessed globally. A virus can spread to files on a LAN-based file server, and from there to the other client systems attached to the server. Similarly, systems which run programs from wide-area file servers can become infected if the programs on the server are susceptible to infection.

While boot viruses could be transmitted on the Internet as diskette images, which would be downloaded and installed onto diskettes, this seems unlikely to become a common means of transporting information. As more information is exchanged over the Internet instead of on diskettes, and the use of diskettes decreases, we would expect a decrease in the prevalence of DOS boot viruses. We would also expect that the increased use of the Internet to interchange and access programs would promote an increase in the prevalence of DOS file viruses.

There have been a few incidents of viruses and worms which are specifically designed to use world-wide networks to spread [7, 8, 9, 10]. These provide dramatic examples of how quickly and how widely viruses can spread on such networks. Fortunately, while these incidents have been rapid and large, they did not usually recur. After a matter of hours or days, when the virus was eliminated from the network and increased defenses put into place, the virus did not continue to spread. Unlike DOS viruses, which have continued to spread around the world for years, Internet viruses have (so far!) been episodic – they come, and then they go. But this need not always be the case.

7 CONCLUSION

The problem of DOS viruses continues to get slowly worse around the world. There are many more viruses than there were a few years ago, and they are appearing at a slightly higher rate. Virus incidents have also increased slightly, but we have to analyze the changes in prevalence of each individual virus in order to understand this trend.

Fortunately, we have made significant progress in this regard. We have achieved a good basic understanding of the spread of computer viruses. We know that a virus can either spread widely or almost not at all, depending upon how fast the virus spreads and how quickly an infection can be found and eliminated. If a virus does spread worldwide, it will rise slowly in prevalence, until it reaches an equilibrium level in the population.

For DOS viruses, this rise is very slow, often taking months or years. The equilibrium level is also quite low. Well-prepared organizations experience about one virus incident per quarter for every one thousand PCs they have, and this incident rate has not changed substantially for a number of years.

Our ongoing study of actual virus incidents had also demonstrated the remarkable effectiveness of good anti-virus software coupled with central incident management in controlling the virus problem within an organization.

This paper has focused on the causes of the major changes in viral prevalence worldwide. We conclude, perhaps surprisingly, that the use of anti-virus software does not play a major role in these changes. Rather, they are determined by the way in which specific viruses, and classes of viruses, interact with the world's computing environment.

We examine the history of several specific viruses to understand this interaction between a virus and its changing environment. The Michelangelo virus was never very prevalent, but media attention to it resulted in increased reports of viruses of all kinds, followed by a temporary decrease in reports. The Brain virus, which spread primarily among systems without hard disks, effectively died out as systems with hard disks became the norm. Virtually all file viruses, including the once-prevalent Jerusalem virus, have decreased dramatically in prevalence because of the increased usage of Windows, and because Windows is fragile in the presence of file viruses. The Form virus, along with other boot viruses, have increased substantially in prevalence, to the point where boot viruses account for around 90% of all virus incidents today. Their spread is not unusual. It is the expected behavior of viruses in a population. They have not died off as have file viruses because their spread is not limited by Windows.

If the computing environment did not change, we would expect that file viruses would remain very low in prevalence, while other boot viruses would increase substantially. If dozens of boot viruses became as prevalent as the Form virus is today, the total number of virus incidents would increase substantially.

By examining trends in the computing environment, however, we can analyze how these might affect computer virus prevalence in the next few years.

Increased use of 32-bit operating systems, such as OS/2 and Windows, is likely to cause a decrease in the prevalence of all current DOS viruses. This is not because they were designed to resist viruses. Quite the contrary: viruses can be written for, and spread by, these operating systems. Rather, the predicted decrease in DOS virus prevalence is simply because features which current DOS viruses use to spread changed in these newer operating systems.

Increased networking, and global networking in particular, will tend to increase the spread of file viruses and decrease the spread of boot viruses. Viruses written to take advantage of features of 32-bit operating systems, especially local and global networking, could become increasing problems. This is a worrisome prospect, as viruses can spread with remarkable speed on world-wide networks.

The technology required to deal with a world of rapidly spreading viruses will be much more challenging than current anti-virus technology. It will be required to respond very quickly, and globally, to new viruses – probably more quickly than humans can respond. While elements of this technology are working in the lab today [19, 20] the task of creating an immune system for cyberspace will occupy us for some time to come [21].

ACKNOWLEDGMENTS

The authors thank Alan Fedeli, Yann Stanczewski and many others for diligently gathering accurate information on worldwide virus incidents for many years. We also thank Joe Wells for his suggestion, later verified experimentally, that *most* boot viruses can spread from within Windows, while most file viruses cannot.

REFERENCES

[1] J.O. Kephart and S.R. White, 'Directed-Graph Epidemiological Models of Computer Viruses', *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, California, May 20-22, 1991, pp. 343-359.

[2] Jeffrey O. Kephart and Steve R. White. 'Measuring and Modeling Computer Virus Prevalence', *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, California, May 24-26, 1993, pp. 2-15.

[3] J.O. Kephart and S.R. White, 'Commentary on Tippett's ''Kinetics of Computer Virus Replication''', *Safe Computing: Proceedings of the Fourth Annnal Computer Virus and Security Conference*, New York, New York, March 14-15, 1991, pp. 88-93.

[4] J.O. Kephart and S.R. White, 'How Prevalent Are Computer Viruses', *Proceedings of the Fifth International Computer Virus and Security Conference*, March 12-13, 1992, New York, pp. 267-284.

[5] J.O. Kephart and S.R. White, 'Measuring Computer Virus Prevalence', *Proceedings of the Second International Virus Bulletin Conference*, Edinburgh, Scotland, September 2-3, 1992, pp. 9-28.

[6] Jeffrey O. Kephart, Steve R. White, and David M. Chess. 'Computers and Epidemiology', *IEEE Spectrum*, May 1993, pp. 20-26.

[7] Spafford, E. H. 1989. 'The Internet Worm Program: An Analysis'. *Computer Comm. Review, 19*, p.1.

[8] Cliff Stoll, 'An Epidemiology of Viruses and Network Worms', *12th National Computer Security Conference*, 1989, pp. 369-377.

[9] M.W. Eichin and J.A. Rochlis, 'With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988', *Proc. 1989 IEEE Symp. on Security and Privacy*, Oakland, California, May 1-3, 1989, pp. 326-343.

[10] D. Seeley, 'A Tour of the Worm', Proc. *Usenix Winter 1989 Conference*, San Diego, California, 1989, p. 287.

[11] P.S. Tippett, 'The Kinetics of Computer Virus Replication: A Theory and Preliminary Survey', *Safe Computing: Proceedings of the Fourth Annual Computer Virus and Security Conference*, New York, New York, March 14-15, 1991, pp. 66-87.

[12] J. McAfee, quoting expert sources on The MacNeil/Lehrer News Hour, March 5, 1992.

[13] Joshua Quittner, 'Michelangelo Virus: No Brush With Disaster', *New York Newsday*, April 5, 1992, pp. 68.

[14] Michael W. Miller, ''Michelangelo' Scare Ends In an Anticlimax', *The Wall Street Journal*, March 9, 1992, pp. B5.

[15] Harold J. Highland, 'Computer Virus Handbook', *Elsevier Advanced Technology*, Oxford, England, 1990, pp. 32.

[16] Steve R. White, Jeffrey O. Kephart, David M. Chess, 'An Introduction to Computer Viruses', *Proceedings of the Fourth International Virus Bulletin Conference*, St. Helier, Jersey, UK, September 8-9, 1994.

[17] John F. Morar and David M. Chess. 'The Effect of Computer Viruses on OS/2 and Warp', *Proceedings of the Fifth International Virus Bulletin Conference*, Boston, Massachusetts, Sept. 20-22, 1995.

[18] 'Viruses on Windows 95', *Virus Bulletin*, June 1995, pp. 15-17.

[19] Jeffrey O. Kephart, 'A Biologically Inspired Immune System for Computers', in R. Brooks and P. Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 130-139, MIT Press, 1994.

[20] Jeffrey O. Kephart, Gregory B. Sorkin, William C. Arnold, David M. Chess, Gerald J. Tesauro, and Steve R. White, 'Biologically Inspired Defenses against Computer Viruses', *Proceedings of IJCAI '95*, Montreal, August 19-25, 1995.

[21] *IBM*'s Massively Distributed Systems home page on the World Wide Web, http: //www. research. ibm. com/massdist.

VIRUS PROTECTION AS PART OF THE OVERALL SOFTWARE DEVELOPMENT PROCESS

Robin J. Kinney

Varian Oncology Systems, Mailstop C-080, 911 Hansen Way, Palo Alto, CA 94304, USA
Tel +1 415 424 6127 · Fax +1 415 424 4511

INTRODUCTION

In developing written processes for dealing with computer viruses in a commercial software development environment, there are two aspects to be considered: the successful elimination of the virus after an infection occurs, and the prevention of virus infection in the first place. The first is relatively straightforward and mechanical. It relies primarily on thoroughness for success. The second aspect can be more difficult, as it is more motivational and relies heavily on the culture of the organization. Developing a formal process for dealing with computer viruses provides a uniform way of training new employees and a mechanism of improving the organization's performance of both prevention and eradication of viruses.

An alternative to a written process is an *ad hoc* process. When a virus is discovered, why not just choose someone for the task, arm them with the best tools, and tell them to go forth? If the same person is chosen each time, they will likely develop their own process, whether it is captured on paper or not. In addition, they will probably get better at it each time. However, without the benefit of a written process, the upper boundary of process improvement is limited, management intervention is required, and the same person must be involved each time for the task to be efficiently performed.

Committing the process to paper does not provide the complete answer. People must be trained to know when and how to use the process. Using the example of system backups, an alternate system administrator who is shown the location of the development system, how to mount a tape, and what commands to type, can successfully perform that task on his own when using the written procedure for guidance. Effective training of people in how to use the process is key and is discussed in greater detail later.

The purpose of a software process is to support the mission of the organization. This mission is to make money through the development of software products, and to do this in a sustainable way. It is not enough simply to produce innovative products which meet customer needs. You must be fast in time to market and deliver a product of high quality. This allows the software developers to concentrate on new products which generate revenue, as opposed to working on removing bugs found by the customer. High quality and short times to market are expected by customers today: if you can't provide them, your competition will. Effective processes which deal with computer viruses support the mission of the organization, and should be part of the overall software development process.

When examining another organizations processes, that organization must be understood to a certain extent to best determine how those processes may apply to your organization. I will describe our software development organization at Varian Oncology Systems to provide that understanding. I will then move on to discuss the specific processes dealing with virus eradication and prevention; then finish with a case study.

Oncology Systems of Varian Associates produce medical devices used to treat cancer. Our major products are linear accelerators used to deliver radiation therapy to patients and data systems which manage information concerning patients' treatments, and also the business of radiation therapy, such as billing and patient scheduling. The business generates $350 million in annual sales, and employs a total of 1250 people with 40 people involved in software development. Currently we have 3000 accelerators and data systems in use at 2700 locations world wide.

The business is highly complex. Part of this is due to the highly technical nature of the products, and part to the safety aspects, since a coding defect could result in the death of a patient. Our business is regulated by the Federal Drug Administration, and an investigator could turn up on our door step without notice. We are also ISO-9001 certified as of 1992.

The data systems we produce use DOS workstations and fileservers in a Novell network. The linear accelerator also uses a DOS PC to provide the user interface and as a storage of configuration data. Several options available for the linear accelerator require additional DOS PC workstations for user interface and control.

The software development environment also consists of DOS workstations and fileservers in a Novell network. Programming languages are C, C++, assembly, and SQL. Software is maintained in a configuration management system and a defect tracking system is used. Processes are used to promote uniformity in our documentation to coordinate reviews of documentation, to organize software testing, and to conduct software releases, to name just a few.

VIRUS PREVENTION

Even in today's environment of networks, fileservers and workgroups, the floppy diskette is heavily relied upon. The floppy diskette is also the primary vector of computer virus infections. The challenge of virus prevention in commercial software development is to control the flow of software in and out of an organization without reducing productivity or stifling creativity. 'Into the organization' pertains to prevention of infection in the first place and 'out of the organization' pertains to increasing the probability that the virus is detected before distribution to the customer - a highly undesirable event.

There are two ways to prevent infections - the methods used by the Department of Defense (DoD), and everything else. In organizations under DoD contracts, it may be a federal offense to bring unauthorized media into a secure area. This may be a bit excessive for a commercial organization, but any time floppies move freely in and out of an organization, viruses are going to propagate. This means that a process to prevent infection in a commercial software organization is only going to be marginally effective. Still, marginal success is the best we have, short of tight security.

The goal, then, is to develop processes which provide the best possible results. At Varian, our prevention process is based on the statement, 'Do not place a floppy diskette in any computer unless you know that it is free of computer virus'. The purpose of this statement is to raise awareness, as it has obvious flaws if taken literally. For example, how do you know when a diskette is completely free of viruses? Virus detection software offers the best solution, and should be part of the prevention process.

Virus detection software available today is good, but there are no guarantees. Perhaps the most difficult problem is how to motivate people to use such software and use it correctly. The virus detection programs which execute as TSRs perhaps offer the best solution, but again, there are no guarantees. People must be motivated to use them properly, keep them at the latest revision (assuming the signature detection type), and continue to use them even when main memory conflicts arise. Awareness and motivation are key.

Having said that, I must admit that I know of no sure-fire way to motivate people regarding processes. So much of the motivational issue is governed by the organization's system of rewards. Often, software development teams receive their rewards by the timeliness and quality of their products, and not by their effectiveness at virus prevention. Perhaps the best which can be done is motivation on a personal level. Make people aware of the problem of computer viruses and what is expected of them regarding prevention. The written process is a good tool for providing awareness and can also describe how virus detection software is to be used, as well as good practices which decrease the likelihood of infection.

Some of the topics to consider in a process dealing with virus prevention are listed below:

- the vectors by which viruses enter an organization
- the types of viruses and how they infect systems
- the proper use of virus detection programs
- the simple rules of virus prevention:
  always buy from reputable dealers.
  never install shareware.
  'do you know where that floppy has been?'
- how might a virus manifest itself
- what to do if you suspect an infection.

If we acknowledge the fact that viruses are occasionally going to find their way into the development environment, then two issues are of primary importance: virus eradication and prevention of distribution to external customers.

VIRUS ERADICATION

The process of virus eradication goes hand in hand with the process of virus prevention. The lessons learned during the elimination of a virus can feed back to make the virus prevention process more effective. In our organization, neither process is so complicated that separate process documents are warranted, thus both topics are covered in the same document. This allows more cohesion between the two processes in both execution and education.

Becoming suspicious of a virus when computers misbehave is the crucial first step in the eradication process. If the virus is detected by a virus scanner, this first step is simplified. Equally important to this first step is the notification of a designated person on suspicion or detection. Even if the person finding the virus is qualified, and they usually are, to remove the virus from their system effectively, they must not do so. There are two reasons for this. The first is that it is difficult to say where else in the organization the virus exists. The second is, even though this one computer may be the only one infected (and it usually isn't), merely removing it deprives the organization of the learning experience which leads to process improvement.

At Varian Oncology Systems, we have a Systems Administrator who maintains the development environment. This is the designated person whom we notify, and the person empowered to form an eradication team. We have found the team approach to be the most effective means of removing the virus from the organization. Although the process of eradication is not particularly difficult, thoroughness is demanded, and speed is necessary to prevent further virus spread. A team is best suited for the task. Whoever is the focus of virus eradication must be empowered to assemble a team.

Who are the people who make good members of the eradication team? One logical person is the person who discovered the virus or first became suspicious of a virus infection. He/she is likely to take a personal interest in the eradication process, and is motivated to see it through to the end. Other people who make good team members are the leaders in the software development organization; often the more senior developers. Although you're using the most valuable resource for a task which may not warrant it from a technical standpoint, you're sending a very important message to the organization. You're saying that this is important, and that it requires a thorough and professional job. Using a highly-respected person decreases the likelihood that developers say, 'Gee, can you come back when I'm done debugging this module?' when the eradication team comes around. Just think of the message sent if only the most junior developers are assigned! Clearly, you don't need every team member to be your senior developers - one or two are adequate.

How many people should be on the team? This depends on the size of the organization, and on the type of virus. In our organization, we have three separate product teams simultaneously developing software, and a total of about 40 people involved in this effort. One or two people from each development team up to a maximum of perhaps 10% of the entire software organization, is a good rule of thumb. Be careful to not let your team become too large as it may cease to function effectively.

Now that an eradication team is assembled, what are they to do? Before they can do anything, the virus must be identified and an effective means of removal must be devised. Identification includes understanding the means of infection, of course. Virus scanners are your best resource here if the virus is detectable by this means. Virus Bulletin is also a source of information. Some developers of virus scanner software offer advice over the telephone as part of their product's licensing agreement. They see a lot more viruses in a month than most of us will (hopefully) see in our entire career. I frequently rely on this service. The virus eradication process is useless, unless a means of detecting and removing the virus is devised. Thus, these steps are essential before the team can proceed.

The team is now armed with a means of virus detection and removal, and is ready to go. If the virus can propagate by means other than floppy diskette, the development systems or fileservers should be taken off-line and not placed back on-line until all systems or workstations which have connectivity are cleared of the virus. A good tactic is to divide and conquer. Separate the organization by workgroups, departments, labs, etc., then proceed through that part of the organization checking for and eliminating the virus. Every floppy which could have been used in the recent past must be checked. When searching in people's offices, out of respect for their space, it may be best to engage them in scanning the hard disk and floppies. If you purchase pre-formatted floppy diskettes, and cannot determine the entry point of the virus, it would be prudent to check a diskette from each box. Be especially careful in shared work areas such as labs and testing areas. Don't forget those floppies kept in briefcases, and the systems people have at home. Thoroughness is the watchword during the phase of virus eradication.

As the eradication team moves through the organization eliminating the virus, good record-keeping is necessary. Every floppy diskette, every workstation hard disk, every fileserver checked for viruses should be recorded. The purpose of the recording is to answer three questions. First, how did the virus enter the organization? Second, how much did the infection cost the organization? Third, could the virus have escaped into the customer's environment and, if so, by what means?

A sample worksheet and checklist are included at the end of this paper. The worksheet helps organize information before the eradication effort begins. The checklist simplifies the record-keeping during the eradication process.

The process of eradication should specify checking all areas which could be infected, but must allow some flexibility. For example, if the team has checked a large quantity of floppies and workstations directly associated with the original infection and found no new infections, the search could end at this point. I would recommend that this decision be made on the conservative side, because a missed infected floppy is likely to reinfect the organization at a later time.

The team should attempt to follow the trail of infection by asking questions such as, 'Who has used this floppy?' and 'Who has most recently used this laptop?'. Remember, we want to determine the source as well as removing all instances of the virus.

Even when the team has completed the eradication to its satisfaction, the task is still not complete. For this process, and every software development process, areas of improvement must be explored. This exploration is performed through the development of a post-mortem report. Many of the written records kept by the team during the eradication process are used to generate this report, which is written by the team leader or anyone on the team, and archived for future reference. The post-mortem report should cover the following topics:

1. the chain of events from suspicion through eradication
2. the source of infection, or most probable sources if the specific source cannot be determined
3. the total cost to the organization
4. evaluation of the effectiveness of the prevention and eradication process
5. suggestions for process improvement.

The action necessary to affect the improvement must be assigned to someone for implementation. Imagine the effect on the team if their recommendations are ignored. A member of the eradication team is a logical candidate for the process improvement task. It is important that individuals are rewarded for process improvement efforts. At Varian Oncology Systems, process improvement is expected, and is one of the items considered during annual performance evaluations.

PREVENTION OF VIRUS DISTRIBUTION

There remains one other area in which a process is helpful in dealing with computer viruses: preventing the distribution of virus to customers. At Varian Oncology Systems and in most commercial organizations, this is a highly undesirable situation, to say the least. The nature of our business is such that we would make a service call to each infected customer to install 'clean' software. Each service call costs our business approximately $1000 domestically; even more if the customer is in another country. In addition, it is damaging to our reputation. From our customers' point of view, they are trusting the quality of cancer therapy to the quality of our software. A virus distributed with that software is inconsistent with that trust. Lastly, we are regulated by the Food and Drug Administration, and the distribution of a virus with our software may require explanations to that regulatory agency.

Having identified this as a threat, what processes are effective at mitigating the release of a virus to a customer? A well-defined process for how a software product is released is a good start. I will describe the highlights of the processes which work effectively for us at Varian Oncology Systems.

Our Software Quality Engineering organization, in preparation for software validation, builds the software using 'gets' of source code explicitly called by file name from the software configuration management system. It is unlikely that source code would become infected, except by malicious intent. Libraries in object form could be infected, but this probability is reduced by purchasing only from reputable vendors and never using shareware or objects downloaded from uncontrolled bulletin boards.

Software Quality Engineering, after validation testing is completed, creates distribution masters. These are never created on preformatted diskettes. The distribution masters are released to our manufacturing organization, along with byte counts for each diskette. The manufacturing organization will accept software only if it is properly released, and only from Software Quality Engineering. This provides a narrow controllable channel through which all released software must pass.

The first thing the manufacturing organization does with the distribution masters is to verify the byte counts and scan the floppy diskettes for viruses. The diskettes are kept in a secure location, and media is copied for distribution on an isolated system. This system is kept secure because only authorized people can gain access: these people are knowledgeable about the threat of virus infection, and are educated in the process. From each batch of distribution diskettes made, a floppy is checked to verify the correct byte count.

TRAINING

Education of the processes associated with virus prevention, eradication, and all software development processes for that matter, is essential if the process is to be effective. Integration of the processes associated with viruses into the processes directly associated with software development provides a uniform way of training new employees, and demonstrates an equal level of importance. Employees new to the organization should be provided with training which familiarizes them with the complete set of processes. This education should stress the employees' roles and responsibilities regarding these processes and to their continual improvement.

Clearly, the set of processes for an organization defines the entire collection of activities. Often, for small projects or those which are less critical, not all processes apply. Flexibility for sizing the process to the task at hand should be built in to the processes. This helps eliminate confusion, and empowers the team to determine what activities apply. On the other hand, if certain employees, projects, or teams are held to a different set of standards than those defined by the processes, then a larger organizational problem exists.

CASE STUDY

SUSPICION AND IDENTIFICATION

At 10:05 a.m. on Tuesday, August 9, 1994, Jeff, a software engineer on our data systems product, asked that I come to his office. We had only one systems administrator at the time, and she was off-site at a training course for the week. Jeff was aware of my interest in computer viruses and my involvement in software process improvement. Jeff told me that he had discovered a virus identified as 'Newbug' on a floppy he used to transport files between work and home. The virus was discovered on the floppy at his home, by Central Point's scanner, PCTools Anti Virus version 2.0 executing as a TSR.

DETERMINING THE METHOD OF REMOVAL

I began a preliminary investigation of the virus. I determined that the current version of Vi-Spy, the virus tool generally used by our organization, successfully detected and removed the virus. I was unable to locate our back issues of the Virus Bulletin so I gave a quick call to Ray Glath of RG Software. Ray confirmed that it was a simple boot sector virus which corrupts DOS executables, and that it has aliases of 'Generic', 'D3', and 'Newbug', but is more generally known as 'Anti-Exe'.

ASSEMBLING THE ERADICATION TEAM

With the Systems Administrator unavailable, I was hoping that the Software Operations Manager would assign someone to lead the eradication team. The data systems team was in the middle of a software release, so I volunteered to lead the team. This decision was made at 1:00 p.m. on the same day Jeff first called me.

As this was a simple boot sector virus, I decided to deviate from the process slightly, and begin with a preliminary search through this one project team. I engaged the help of Jeff for eradication during this preliminary phase. Also, due to the nature of the virus, I elected to not remove fileservers and workstations from the network.

ERADICATION OF THE VIRUS

Throughout the afternoon of August 9, and into the next day, Jeff and I scanned for AntiEXE. Below are the statistics for this preliminary phase.

total number of workstations checked	26
total number of workstations infected	1
total number of floppies checked	25
total number of floppies infected	7

Five of the seven floppies infected were the distribution masters of an official build. This was rather curious, given the mode of propagation of AntiEXE, because the build engine on which these masters were built was not infected. This led to two possibilities. Either someone discovered the infected build engine and removed the virus without proper notification, or the floppies were infected before files were copied to them.

Due to the significant number of infections discovered during the preliminary search and particularly, the inconsistencies in the information, the search was expanded with more rigor. I added one more person to the eradication team. During the first three days of the week of August 15th, a total of 19 additional floppies in the data system testing lab were found to be infected. At that time we used preformatted floppies for the distribution masters, so one floppy from each box from our in-house store was checked. These floppies were free of viruses.

Then, on Friday, August 19th, 36 infected floppies and one infected workstation were discovered in the office of one of the data systems technical writers. Many of these floppies were received from an outside vendor under contract to create customer training material for the data system product. This vendor was contacted, but they were very quick to stress that the virus could not have originated with them. It was impossible to tell if their floppies infected the technical writer's workstation or vice versa.

Also on Friday, August 19th, due to the large number of infections discovered thus far, I decided to check the building where the linear accelerator software is developed, and engaged five people to form the eradication team there. We moved quickly through 23 offices, 32 workstations, and 88 floppies, finding no instance of the virus. This effort required approximately three hours.

I had learned during the eradication process that, approximately two weeks before this discovery of the virus, AntiEXE was discovered on the workstation of the same technical writer who, this time, had the 36 infected floppies. At that time, the virus was removed from the workstation by a person who meant well but was outside the software organization, and was unaware of our process.

It was impossible to determine with certainty the vector of the infection. The two prime candidates were the vendor creating the customer training material, and the preformatted floppies.

The statistics for this infection of AntiEXE are as follows:

Elapsed time from virus suspicion to eradication	9 days
Total time to eradicate the virus	9 days
Total number of computers checked	52

Total number of floppies checked	316
Total number of workstations infected	2
Total number of floppies infected	61
Total number of fileservers infected	0
Total man-hours lost due to infection	116
Total cost to the organization	$3500

In examining the statistics for this infection, it seems unlikely that only two workstations would have propagated 61 infected floppies. Unfortunately, there is insufficient data to reach a conclusion.

LESSONS LEARNED

- It would serve us well to have a more comprehensive way of dealing with viruses for the entire enterprise. Had this been the case, we would have likely been spared the reinfection of AntiEXE.
- We have a process by which we evaluate outside software houses. This includes evaluation of their ability to deal with computer viruses. The development of the customer training material was managed by a department which was unaware of this process, and thus did not arrange for a vendor evaluation. Had this process been followed, the probability that the infection would have been prevented increases somewhat. This assumes that this vendor is the source of the virus, which cannot be positively determined. A better way of general process training may be beneficial.
- We should not use preformatted floppies for software distribution masters.
- A group of people working as a team can move quickly and effectively through an organization to search and remove a virus detectable by a virus scanner.

CONCLUSION

Written processes dealing with computer viruses can be of great value to a software organization. Although for a commercial organization the process offers only marginal benefit for prevention, it offers significant benefit for removal of the virus. An effective process can also help in preventing viruses from being distributed to customers with released software. Incorporation of these processes into the set of processes which define how business is conducted provides a uniform way of educating new employees in the area of computer viruses. This incorporation also provides a standard method for process improvement. The processes associated with computer viruses not only complement each other, they mesh with those processes associated with software releases, producing distribution media, development system backup, and development environment security, to name just a few.

Instituting written processes in an organization, and the continuous process improvement associated with this, can be a difficult undertaking. It is extremely hard to build that initial momentum, and there are always those in the organization who fear and resist change. Don't be discouraged. Begin with a small set which the organization is already using informally and build from there.

As for the processes dealing with viruses, each time you are required to respond to an infection, examine what works well and what doesn't. Concentrate your process improvement not only on the mechanics of virus eradication, but also on the infrastructure and communications needed to be effective in the process. It is the infrastructure and communication, as opposed to the mechanics, which will be most important when faced with one of the viruses not detectable by virus-scanning software.

VIRUS ERADICATION TEAM WORKSHEET

Virus Name ____________________

Aliases ____________________

Virus Type ____________________

First Suspicion

Date __________ Time __________

Person ____________________

Date Systems Administrator Notified __________

Time ____________________

Positive I.D.

Date __________ Time __________

Person ____________________

Scanner Name ____________________ Version Number ____________________

Virus Expression Characteristics ____________________

____

____

Virus Elimination Method ____________________

____

____

Is this the first infection by this virus? Y / N

Previous Date if 'No' __________

Eradication Team Members

____________________ ____________________

____________________ ____________________

____________________ ____________________

____________________ ____________________

____________________ ____________________

____________________ ____________________

VIRUS ERADICATION TEAM CHECKLIST

Name______________________________________ Date ____________

page_____ of_____

FOR EVERY ITEM CHECKED:

Type	Name	Location	Time Spent	Infected
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N
f h f/s				Y N

f - floppy **h** - hard disk **f/s** - fileserver

Name - any specific identification of the media or the fileserver (not necessary for uninfected floppies)

Location - name of the lab, office, server room, etc

Time Spent - total time to and from location and checking and/or removing virus and restoration of files

HARMLESS AND USEFUL VIRUSES CAN HARDLY EXIST

Pavel Lamacka

HT Computers Ltd, (Computer Accidents Research Center), Sliacska 1, 831 02 Bratislava,
Slovak Republic
Tel +42 7251 426 · Fax +42 7252 742

INTRODUCTION

Some virus authors and even some antiviral experts claim that not all viruses and programming techniques used in them are harmful and therefore bad [Cohen-85] [Cohen-92]. They argue that viruses which have the ability to execute no action, neither harmful nor useful, are harmless and therefore neutral, and that viruses which are able to execute beneficial actions are useful and therefore good. Discussions about harmless and useful viruses are still not finished as can be seen, for example, from [Kaspersky-93] or [Timson-93]. Neither are they academic, because our basic attitude towards viruses; the techniques of their implementation; and their originators and propagators depends on the results of these discussions. If viruses are really neutral in nature, it is necessary to discipline only those responsible for their unsuitable purposes and usage. But if we find out that viruses are bad in principle, we obtain the right to take a consequently defensive attitude towards their originators and propagators. The goal of this paper is the presentation of reasoning leading to a standpoint which is usable in practice regarding the existence and feasibility of harmless and useful computer viruses. The presented reasoning is based on a combination of known, lesser known and so far probably undiscussed facts and conclusions. Those of which are considered contributions of this paper *are indicated.*

HARMFUL VIRUSES

Before we start a discussion about the possible existence of harmless and useful viruses, we will take a look at harmful viruses. Viruses which are able to execute a harmful action, like destroying data or disabling the usage of a computer, are considered harmful. Generally a harmful virus V_H (Fig. 1) consists of at least the two following modules: a module of self-replication M_{SR} and a module of harmful action M_{HA}. The harmful action is usually executed by the virus on a certain condition. Other modules and functions of the viruses, for example, stealth, encryption or polymorphism, will not be considered here, because they are irrelevant to this paper.

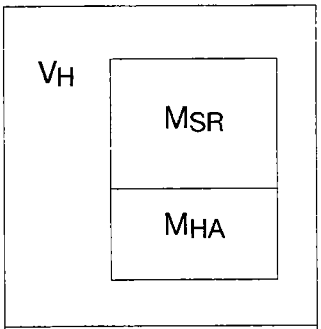


Fig. 1 Harmful virus

Many people claim that viruses have gained their bad reputation only due to the harmful actions which many of them execute. Let us therefore look at whether the harmfulness of a virus would disappear after removing the harmful action code from it, and then at whether it is possible to obtain a useful virus after it is given the ability to execute a useful action.

HARMLESS VIRUSES

Virus V_N (Fig.2), which can do nothing but self-replicate, consists of a self-replication module M_{SR} only. Several such viruses are known in practice. It is known that although this type of virus does not contain any harmful action module, it is able to damage the code of a program on which it is a parasite. This is often caused by the untidy implementation of the virus. It may happen that the virus is implemented in a competent way, but then it meets with a new program structure which it cannot infect properly and therefore it damages the structure. Users have no means of defending against such side-effects because until now it has been unusual to accept complaints about viruses, for example, on hot-lines.

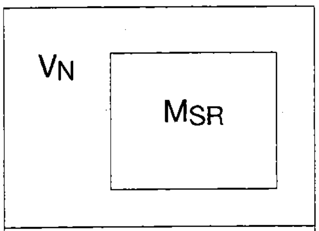


Fig. 2 No action virus

Moreover, it follows from the principle of the function of viruses that they always interfere with the integrity of infected programs by their activity. This results from the fact that all *viruses obtain control flow by theft*, that viruses steal control from the programs which they have infected. Usually, but not always, they steal control by modifying the code of the infected programs. An example of viruses which steal control without program modification, are companion viruses. *By the theft of control the viruses act as parasites* on the programs infected by them. This ability is given to each virus at the time of its origin. It is the inherently parasitic nature of the self-replication of computer viruses which interferes with the integrity of the programs affected by them.

Besides, by self-replication, *viruses waste computer resources*, particularly memory and processor time, which is also a form of doing harm. Although this form is often tolerated, it is *unpredictable* and in time-critical applications it can be substantial and is therefore intolerable.

It depends whether some of the given influences are demonstrated to be harmful ones. In all cases, by their ability to self-replicate and their parasitic nature, viruses violate the conditions of function of the programs affected by them, therefore the authors of the programs cannot guarantee the functionality of the programs, which restricts their author's rights.

From the above mentioned arguments, it is sufficient for everyday practice that the best which can be said about the simplest viruses, containing no code for harmful actions, is the following:

(I) *The harmlessness of computer viruses is not guaranteed.*

In other words, the usage of any virus is risky, because of the danger of violating computer activity. This riskiness of computer viruses results from their ability to parasitically self-replicate. Because without this ability the virus is not a virus, it follows that this riskiness is peculiar to all computer viruses, also in cases when they have no ability to execute harmful action, and even when they have the ability to execute useful action. It also follows from that, that the danger of harmful viruses does not rest only in their ability to execute harmful actions.

USEFUL VIRUSES

Generally, a useful virus V_U (Fig.3) consists of at least the following two modules: module of self-replication M_{SR} and a module of useful action M_{UA}. The useful action is usually executed by the virus on a certain condition.

A useful action which virus can execute is, for example, the compression of the code of an infected program, as is done by the virus 'Cruncher' [Kaspersky-93] [Timson-93]. Another example is the virus 'AVV' [Kaspersky-94], which detects the presence of other viruses.

It is problematic to evaluate the virus V_U as unambiguously useful, because it is unknown whether the usefulness of its action outweighs the riskiness of its self-replication. Moreover, it is problematic to compare the usefulness of the action to the riskiness of the self-replication. Even if the usefulness of the action was much greater, the riskiness of the self-replication, however low, might be intolerable, and therefore, the virus as a whole could not be evaluated as unambiguously useful.

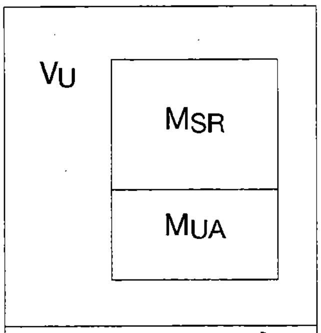


Fig. 3 Useful virus

Let us consider a virus V_C, whose useful action is a compression of the code of programs. Fig. 4 shows the situation when N programs P_1 - P_N have been infected by the virus. Each of the programs P_i has been compressed at the time of its infection by the virus. Along with it a part of the virus V_C acting as a parasite on it has been compressed. The rest of the virus, which is a decompression module D, has not been compressed and receives control at the time of activation of the program P_i. Module D decompresses the program P_i and the compressed part of the virus V_C to their original state, control is passed to the decompressed remainder of the virus V_C, and the rest of the process goes on as usual for viruses. This means that a program P_i infected by the virus V_C behaves like a self-expanding program.

From the user's point of view, besides the above mentioned problems, there are the following interesting matters. The compression module is present in each instance of the virus V_C, in our case it is N-times, which is not the case in common compression programs. Next, it is interesting that the *virus activity is uncontrollable*, because the virus itself finds the programs to be infected, fully *autonomously*, according to the rules built into it. Due to this reason, the user is unable to decide on which programs the compression should be applied and on which ones it should not. Finally, it is interesting that *viruses behave in an indeterminate way*, because their activity often depends on software configuration, sequence of executed operations and other parameters of random nature. Therefore it is generally *unpredictable* whether at a given moment the compression has been applied to any given program or whether it has been applied to all considered programs.

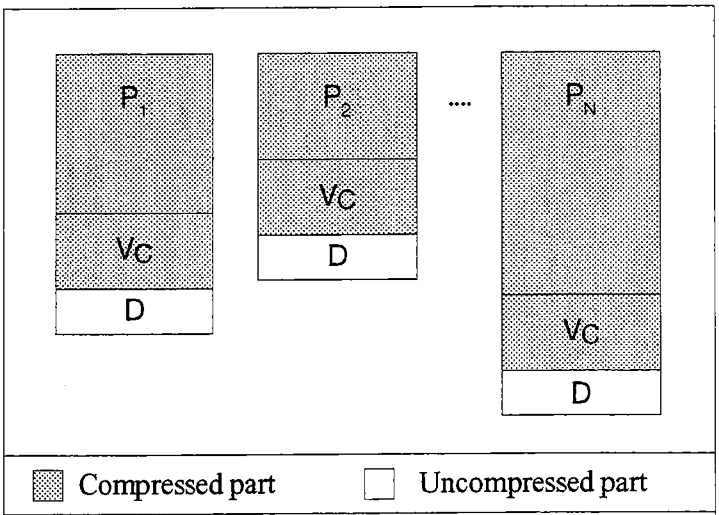


Fig.4 Programs P_i infected by compression virus V_C

The above facts disqualify the useful and harmless viruses to such a degree, that the least negative statement we can say about them, is the following:

(2) *Harmless and useful viruses can hardly exist*

In the next section we will look at whether it is necessary to regret that useful viruses have such weak prospects.

USEFUL VIRUSES VS USEFUL NON-VIRAL PROGRAMS

If a common, correctly implemented compression program is used, we avoid the problems inherent in the compression virus, V_C. First of all, we avoid problems with uncontrollability and indeterminacy, because it is possible to state on which programs to apply the compression and, after the compression is finished, it is apparent that compression has been applied only to the stated programs and not to others.

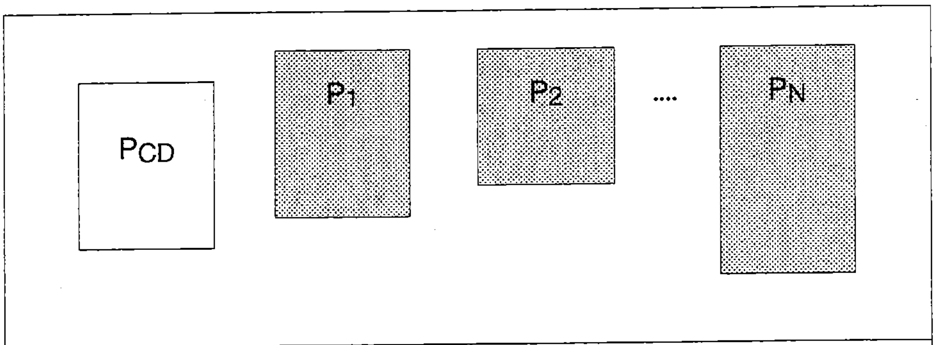


Fig. 5 Programs P_i compressed and uncompressed by program P_{CD}

The differences in the demands on memory and time are not negligible as well. The compression code together with the self-replicating one occur in each instance of the virus V_C, that is, in each infected program (Fig. 4). On the other hand, if we use a compression program P_{CD}, which may or may not be memory resident, the compression code is necessary only in one instance and the self-replicating code is unnecessary (Fig. 5). It can be seen that the programs P_1 - P_N, on which the compression program P_{CD} was applied, contain no extraneous code.

Similarly, the program P_{CD} is more time-efficient, because it works on demand only, and not like the virus V_C, which works every time it steals control. If the compression program P_{CD} is memory resident, it works autonomously, which removes the last illusory advantage of the useful viruses, for which some of their proponents argue.

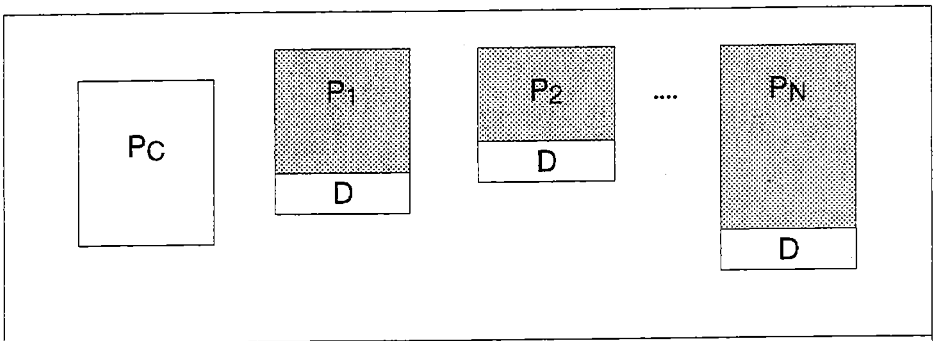


Fig. 6 Self-expanding programs P_i compressed by program P_C

In practice we also use compression program P_C, which transforms the given programs into self-expanding form (Fig. 6). A compression program P_C compresses given programs P_1 - P_N and then adds to them a decompression module, D, which automatically decompresses them at their activation. The addition of the decompression module influences the integrity of the given programs, but if the authors of the programs agree with this process, their author rights are not violated, and if they themselves apply such compression to their programs, the integrity of the programs is not affected.

The procedure, which was demonstrated in the comparison of compression viruses and compression programs, can be generalised in the following statement:

(3) For each virus which is able to execute an action, it is possible to implement a program, which is not a virus and which is able to execute the same action.

Therefore for each virus V_U (Fig. 3), which is able to execute a useful action using the module M_{UA}, it is possible to work out a useful program P_U (Fig. 7), which is able to execute the same useful action as the virus V_U. P_U needs no self-replication code, therefore it is not a virus. Instead it contains a module of selective application, M_{SA}, by which the useful action is selectively applied according the commands of the user of the program P_U. In an extreme, the program P_U can contain a copy of the module M_{SA}, which would guarantee the equivalency of an execution of the useful action. Using statement (3), the above comparison of features shown for compression viruses and compression programs is valid for every pair V_U - P_U, which executes the same action.

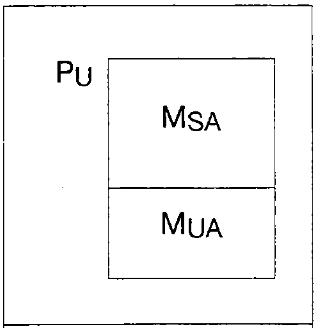


Fig. 7 Useful program

Now we are at the end of the comparison of the features of the useful viruses, V_U, and the useful programs P_U. Their comparison overview is given in Table 1. From it and from statement (3) the following statement results:

(4) Useful viruses are useless.

This is so because useful programs are unambiguously more advantageous, as useful viruses have only one from the given list of positive features, which is the ability to execute useful actions. Otherwise the usage of viruses for useful purposes is *hazardous*, because it is accompanied by several risks.

feature	**useful virus V_U**	**useful program P_U**
useful action	+ able to execute	+ able to execute
self-replication	- basic ability, without it virus is not a virus	+ does not need
parasitic ability	- basic ability, without it virus is not a virus	+ does not need
controllability	- autonomous, uncontrollable	+ user controllable
predictability	- indeterminate behaviour	+ predictable behaviour
memory usage	- unpredictably excessive	+ need not be excessive
processor usage	- unpredictably excessive	+ need not be excessive
	- *is risky and therefore negative feature*	+ *is positive feature*

Table 1. Comparison of basic features of useful viruses and useful non-viral programs

CONCLUSION

To justifiably speak about the existence of harmless computer viruses, it would be necessary to prove, or at least to show, how to implement them in such a way that it would be possible to guarantee their harmlessness. That would disprove the validity of statement (1) and at the same time open the possibility of the existence of unambiguously useful viruses.

To justifiably speak about the existence of useful computer viruses, it would be necessary to prove, or at least to show, that there exists an action which can be implemented in the framework of a virus and which cannot be implemented in the framework of any non-viral program. That would disprove the validity of statement (3) and therefore (4). Another possibility would be to prove, or at least to show, that there exists an action which is more effective to implement in the framework of a virus than in the framework of any non-viral program. That would disprove the validity of statement (4), but not (3). To justifiably speak about these viruses as being unambiguously useful, statement (1) may not be valid.

Until such proofs can be given, claims about the existence of harmless and useful viruses are but the products of *wishful thinking* of their proponents, and attempts to create and use them are hazardous. In the case of the useful viruses it is an unnecessary hazard, because useful non-viral programs do not carry the risks which viruses do. The hazardousness of the viruses results from the cumulative effect of risks connected with their usage. These are mainly risks resulting from the parasitic self-replication of the viruses, from their uncontrollable and indeterminate activity, and from their unpredictably excessive usage of memory and processor.

Software engineering looks for programming techniques whose use minimises the risks of incorrect software function. This is why we consider as unsuitable those programming techniques, which the hazardousness of the computer viruses is based on. From the viewpoint of software engineering, *viral programming techniques are dirty* at least as unstructured or non-modular programming is, since their exploitation is dangerous.

It is known that all viruses in some way violate the integrity of the infected programs, which is a given due to their parasitic nature. This interferes with the author and user rights of the respective infected programs. The authors and users of those programs have the right to protection by law, to recompensation and to the prosecution of the culprits who spread viruses actively or support their spread through negligence.

REFERENCES

[Cohen-85] Fred Cohen, 'Computer Viruses', 1985.

[Cohen-92] Frederick B. Cohen, ''Wrong' Said Fred', *Virus Bulletin*, January 1992, pp. 5-6.

[Kaspersky-93] Eugene Kaspersky, 'Cruncher - The First Beneficial Virus?', *Virus Bulletin*, June 1993, pp. 8-9.

[Kaspersky-94] Eugene Kaspersky, 'AVV - The Anti-Virus Virus', *Virus Bulletin*, January 1994, pp. 10-11.

[Timson-93] Harriet Timson, 'Cruncher - Zipping or Zapping', *Virus News International*, May 1993, p. 31.

THE EFFECT OF COMPUTER VIRUSES ON OS/2 AND WARP

John F. Morar, David M. Chess

High Integrity Computing Laboratory, IBM Thomas J. Watson Research Center, Yorktown Heights,
NY 10598, USA

Fax +1 914 784 7007 · Email morar@watson.bm.com

OVERVIEW

Although the number of OS/2 viruses can be counted on the fingers of one hand, systems running OS/2 still require protection against thousands of DOS-based viruses which can infect boot records and DOS programs on OS/2 systems.

OS/2 is a 32-bit multitasking operating system which can simultaneously run programs written for OS/2, DOS and *Microsoft* Windows™. DOS programs running under OS/2 execute in a Virtual DOS Machine (VDM) which is designed to provide an environment which appears the same as real DOS. Ironically, moving toward the goal of a perfect virtual DOS machine increases the probability that an infected DOS program will execute properly, and effectively propagate its virus to other DOS programs stored on the system. Indeed, DOS programs executing under OS/2 can frequently spread file infecting viruses to other DOS programs.

Boot sector viruses interact primarily with the Basic Input and Output System (BIOS) which is common to all IBM PC (and compatible) personal computers. Boot sector viruses typically receive control during the boot process, before the operating system is loaded; this allows them to infect boot sectors independent of the operating system in use. Boot sector viruses under OS/2 don't usually spread to diskettes because of the details of how OS/2 uses diskettes. However, they can have other detrimental effects on the system, and therefore need to be removed.

Viruses designed to infect native OS/2 executables are more complicated to write than their DOS counterparts, but they will likely be a problem at some point in the future. We are currently aware of only two OS/2 viruses.

Both of these viruses are very simple and neither of them has been detected in the wild ('in the wild' viruses are those which have been detected spreading in real life situations.)

OS/2 is far more versatile than DOS/Windows. It has the ability to run multiple DOS and Windows sessions, provides facilities for booting multiple operating systems, and allows file names up to 255 bytes long.

These additional capabilities offer new places for viruses to hide, and necessitate additional anti-virus capabilities not available in native DOS/Windows anti-virus software. Although native DOS/Windows anti-virus programs can often execute under OS/2, they do not provide an adequate level of protection. They will not have access to files with long names, nor will they find all the boot records which may be located on the machine. Under some versions of OS/2, the boot sectors are not writable by DOS/Windows programs and can therefore not be disinfected by DOS/Windows anti-virus products.

This susceptibility to DOS viruses is not unique to OS/2. Indeed, Windows NT and Windows 95 are also fertile ground for the spread of DOS viruses. Each of these operating systems requires individually tailored anti-virus protection software.

BOOT SECTOR VIRUSES UNDER OS/2

Boot sector viruses are responsible for the overwhelming majority of in-the-wild virus infections. They reside in the boot records found on each disk and diskette. The primary method for checking a particular machine for boot sector viruses is to scan all the boot records located on the personal computer. One consequence of OS/2's versatility is the possibility of additional boot records not found in DOS systems.

OS/2 provides optional facilities for installing more than one operating system on a single personal computer.

Two methods are provided for choosing which operating system is to be booted: Dual Boot and Boot Manager. Each of these methods involves manipulating which boot sectors become active. Using either of these techniques results in additional boot sectors not found on DOS and Windows-based systems. Effective OS/2 anti-virus programs will scan all the boot records located on the personal computer, even those which are not active at the time the scan is being performed.

In particular, on a Dual Boot system the system boot record of the operating system which is not currently active is stored on the hard disk, under a special name.

OS/2 anti-virus software should know to scan for boot sector viruses in files with these names. In a Boot Manager system, a special Boot Manager boot record exists which is neither a master boot record nor an operating system boot record; OS/2 anti-virus software must know how to scan and repair this special kind of boot record.

OS/2 offers a choice of two file systems; the File Allocation Table (FAT) file system and the High Performance File System (HPFS). Some boot sector viruses assume that all file systems are FAT, and write to specific disk locations in ways which can damage HPFS boot partitions. The risk of such complications for OS/2 systems in high virus risk environments can be minimized by using the FAT file system for all boot partitions.

DOS FILE INFECTING VIRUSES UNDER OS/2

DOS programs running on an OS/2 system execute inside a Virtual DOS Machine (VDM), a controlled environment in which OS/2 provides DOS programs with all the usual DOS services, and in general simulates a DOS environment.

Multiple VDMs can be used to simultaneously execute multiple DOS programs. Many infected DOS programs execute properly in OS/2 VDMs and can effectively propagate a virus to other DOS programs stored on the system.

File infecting viruses frequently install a memory resident component in the DOS operating system (or in the VDM in the OS/2 case); this component infects new programs as they are executed, or executable files as they are opened, or it may follow any of a variety of other strategies. Because the DOS simulation provided by the VDM supports this kind of memory-resident component, viruses of this kind often continue to operate in a VDM.

(Some file infecting viruses use undocumented and unsupported features of DOS to function; these will often fail in OS/2 VDMs.)

Memory-resident viruses cannot spread directly between separate VDMs; however, any program executed from within an infected VDM will likely become infected. If that program (once infected) is later executed in another VDM, that VDM can also become infected, in the sense that the virus will have installed its resident portion in that VDM as well.

The best protection for VDMs under OS/2 is to install memory-resident virus protection in each VDM as it is opened. This function can be performed automatically by anti-virus software tailored to the OS/2 environment.

Occasionally, a file-infecting virus designed for DOS will also attempt to infect OS/2 executable files. Although the structure of an OS/2 executable file is superficially similar to a DOS EXE file, it is in fact far more complex. If a DOS program attempts to infect an OS/2 executable, it will almost always fail, rendering the OS/2 executable unable to execute under OS/2, and making it impossible to repair fully the file. In some cases, trying to start an OS/2 program in an infected VDM can cause the OS/2 program's 'DOS stub' (the part of an OS/2 program which prints 'This program cannot be run in DOS mode') to become infected. An OS/2 program infected in this way can sometimes even spread the virus when started under DOS, or in an OS/2 VDM. It is therefore important to check both DOS and OS/2 executables for file-infecting viruses on OS/2 systems.

NATIVE OS/2 VIRUSES

There are currently only two OS/2 viruses known to us.

- **OS2vir1**: This virus functions by (roughly) replacing all EXE files in the current directory with a copy of itself. Since infected files no longer perform their normal functions, this is a very noticeable virus and therefore unlikely to spread. It is distributed as source code, and as distributed, prints out messages as it runs saying which files it's "infecting".
- **Jiskefet**: Replaces EXE files with a new file which contains within itself the original EXE file. When the infected file is executed, it recreates the original EXE file under another name and then executes the original file. This is a technological advance over OS2vir1 since the function of the original program is preserved. However, Jiskefet is not very effective at finding new files to infect. Similar viruses in the DOS world have never spread well, suggesting that Jiskefet will also not pose any significant threat to OS/2 systems.

In spite of the current unsophisticated attempts at OS/2 viruses, there is no insurmountable technological barrier to generating effective viruses for any of the currently shipping 32-bit operating systems; it makes sense to prepare now, by installing the best available anti-virus software designed specifically for the operating systems that you are actually using.

TAKING ADVANTAGE OF OS/2 FACILITIES

Like any other modern product, anti-virus products should take advantage of the power and flexibility which OS/2 brings to the user. An anti-virus product should, for instance, be able to run in the background at pre-selected times, to avoid interfering with the user's daily work. An anti-virus product should have a full graphical user interface, allowing any necessary user interactions to take place on the desktop, rather than through older command-line interfaces. Advanced file systems, like the one provided with OS/2, require the system to be shut down so the file system can close all files and store all data. The next action after a shutdown is to restart the system, either immediately or at some later time. The shutdown process is an excellent time to scan any diskette left in the A drive for boot sector viruses. Diskette scanning during shutdown avoids possible infection when the system is again restarted.

Even a non-bootable diskette can be infected with a boot sector virus, and can spread the virus if an attempt is made to boot from the infected diskette.

SUMMARY: REQUIREMENTS FOR PROTECTING OS/2 SYSTEMS

To be effective in protecting an OS/2 system from viruses, an anti-virus product must:

- run as a native OS/2 application, in order to check files and directories which DOS applications cannot see
- check all boot records on the system, including BootManager boot records and the files used by Dual Boot to store boot records
- provide protection for all DOS VDMs and Windows sessions running under OS/2
- check to see if there is an infected diskette in the diskette drive immediately before shutting down
- perform scheduled scans of the system, in the background, exploiting OS/2 multitasking abilities
- take advantage of the sophisticated user interface facilities in OS/2 to run cleanly on the desktop, rather than requiring command-line interaction.

Our development of IBM AntiVirus for OS/2 has been motivated by the need to satisfy all the requirements described in this article.

* IBM, OS/2 and OS/2 Warp are registered trademarks of *International Business Machines Corporation.*

* All other products are trademarks or registered trademarks of their respective companies.

HEURISTIC SCANNERS: ARTIFICIAL INTELLIGENCE?

Righard Zwienenberg

Computer Security Engineers, Postbus 85 502, NL-2508 CE Den Haag, The Netherlands
Tel +31 70 362 2269 · Fax +31 70 365 2286 · Email rizwi@csehost.knoware.nl

Though not explicitly stated, heuristic anti-virus methods have been in use for almost as long as the virus threat has existed. In the 'old days', FluShot(+) was a very popular monitor, alerting the user when it detected 'strange and dangerous' actions. This can be regarded as simple heuristic analysis, because FluShot did not know if the action was legitimate or not. It just warned the user.

During the last couple of years, several resident behaviour-blockers have been developed, used and dismissed again. In most cases, the user finds warnings irritating, aggravating and incomprehensible. The only resident protection they normally use - if any - is a resident scanner. This makes life easier for the users, because the resident scanner clearly indicates that a file or disk is infected by a certain virus when it pops up its box. The disadvantage, which the user doesn't see, is that it does not detect new viruses.

Also, the less popular (but very important) Integrity Checkers may be regarded as heuristic tools. They warn the user when the contents of files have been changed, when files have grown in size, received new time and date stamps, etc. They often display a warning such as: 'file might be infected by an unknown virus' in the case of a changed executable. Especially in a development environment, Integrity Checkers can be really irritating. The user already knows that his executable has changed, because he just changed and recompiled the source code. But how is the Integrity Checker to know that? Using a list of executables to skip is not safe, because a virus may indeed have infected an executable on the list. In that case, the change was not caused by a recompilation. However, the integrity checker can't tell the difference!

Based on these early attempts, the first generation of scanners with minor heuristic capabilities were developed. The heuristics they used were very basic and usually generated warnings about peculiar file date and file time stamps, changes to file lengths, strange headers, etc. Some examples:

```
EXAMPLE1.    COM  12345 01-01-1995  12:02:62
EXAMPLE2.    COM  12345 01-01-2095  12:01:36
EXAMPLE3.    EXE  Entry point at 0000:0001
```

The heuristics of the current, second, generation of scanners are much better. All the capabilities of the first generation scanners are obviously retained, but many new heuristic principles have been added: code analysis, code tracing, strange opcodes, etc. For example:

```
0F POP CS
```

Strange opcode – an 8086-only instruction!

```
C70600019090      MOV WORD PTR [100],9090
C606020190        MOV BYTE PTR [102],90
E9                JMP 0100
```

Tracing through the code shows that it jumps back to the entry point:

```
B9                MOV CX,....
BE....            MOV SI,....
89F7              MOV DI,SI
AC                LODSB
34A5              XOR AL,A5
AA                STOSB
E2                LOOP ....
```

This is obviously decryption code.

A (third generation?) scanner type based exclusively on heuristics exists, performing no signature, algorithmic or other checks. Maybe this is the future, but the risk of a false alarm (false positive, true negative) is quite high at this moment. In large corporations, false alarms (false positives) can cost a lot of time and thus money.

We are not going to examine this scanner type except to note that it may lead us into a new generation, or area, of system examination and protection: *Rule-based Examination Systems*.

RULE-BASED EXAMINATION SYSTEMS

Rule-based systems are as such not a novelty. They already exist, also in the security field. In this field they are often characterised by applying very few, but very broad, rules.

What we are going to look at here are Rule-based Examination Systems seen as large heuristic analysers.

Looking at this sequence of opcodes:

```
B8DCFE            MOV AX,FEDC
CD21              INT 21
3D98BA            CMP AX,BA98
75..              JNE getint21
E9....            JMP wherever
getint21:
B92135            MOV AX,3521
CD21              INT 21
```

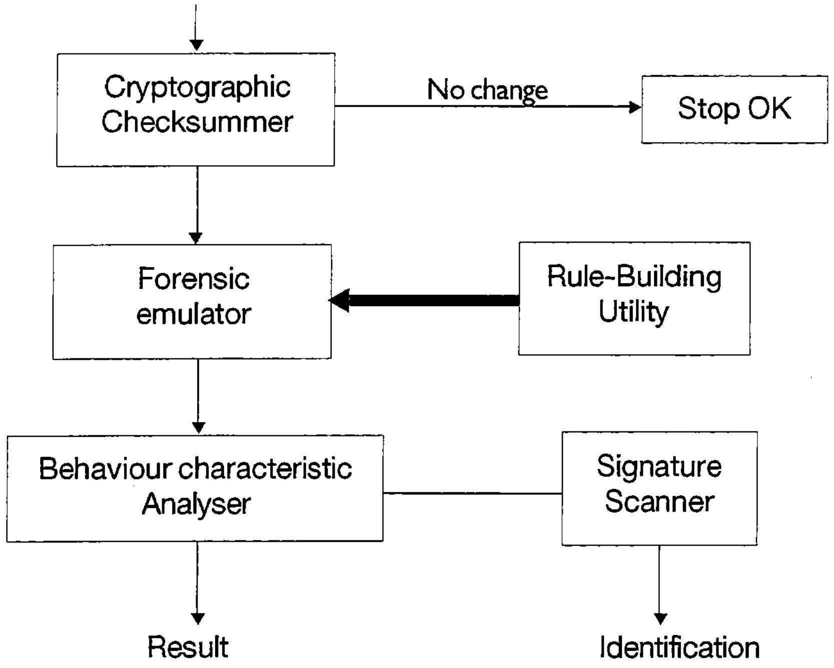
everyone in the field of computer security can see that we may have a virus here (or at least suspicious or badly programmed code). The problem is how to convert something we see in a split second into one or more specific and relevant behaviour characteristics, which we can feed into an examination system. This in turn is able to tell us whether or not we are looking at a virus.

With most of the rules used by the first generation of heuristic scanners, this was not at all difficult. Most were simple comparisons (<,>,==,!=) of the type: 'If a file date exceeds the current date, or is after the year 2000, give an alert'; 'If the seconds field of the file time shows 62 seconds, we can conclude that this is pretty strange and give an alert'. This generation of heuristics, of course, did not have the power to analyse the code in the example shown above.

The second generation of heuristic scanners has more possibilities. Bearing those in mind, defining a rule to cover the example above is not difficult, but imagine a complex decryption routine preceding the actual (virus/Trojan/suspicious) code or – most likely – legitimate code. For example:

```
re-vector int3
re-vector int1
disable keyboard
get int1 offset into di
get int3 offset into si
add counter-1 to si to point to
                 encrypted data
add counter-2 to di to point to
                 encrypted data
get word into ax
perform some calculations with ax
                 to decrypt word
store word
increase counter-1
increase counter-2
look if end of encrypted code has
                 been reached
jmp back if more code to decrypt
enable keyboard...
```

In case this is just one of the instances generated by a complex mutation engine, it will be hard to derive a heuristic rule directly to detect a virus using this engine.



One of the solutions, maybe the best one, is to include a code emulator in the analysing system as illustrated in the figure above, which shows a part of a working network security system. The file to be checked is first given to a checksummer. If the file is already known to the system, a hash code is generated across the file, and this is compared to a stored value. If these are identical, no further action is taken, and the file is declared clean. If not, the file is fed to the emulator, and the results from the code emulation are given to an analyser as described below.

Including a code emulator is possible, and as a matter of fact has already been done. It should have special knowledge of a variety of possible tricks used in malicious code; it should know when to stop emulating (e.g. at the end of a decryption routine); it should be able to realise when anti-debug tricks are used, etc. Both in order to obtain portability, and to avoid obvious pitfalls, it must adhere to one basic and important rule: *Never actually execute an instruction, only emulate it.*

In short, the task of the emulator is first to make sure that the code is decrypted (in case it was encrypted), and then to derive and combine relevant behaviour characteristics to pass on to the analyser, which analyses and organises these behaviour characteristics and compares the results of the analysis with a set of rules.

ARTIFICIAL INTELLIGENCE

From the point of view of the developer it would be nice if such a system were able to learn about behaviour characteristics and generate new rules automatically. If the system bypasses an instance of virus/Trojan/suspicious code because the current rules are no longer sufficient, special examination tools should be able to extract the necessary information from the code in question and create new rules enabling the system to detect this trojan/virus/suspicious code, and hopefully every other form derived from this one. In other words: *Artificial Intelligence.*

For security reasons, these additional tools with their special functionality should not be given to users. Evil-minded knowledgeable persons could use them to do an in-depth disassembly to research the possibilities of bypassing the rules generated by the system. Security through obscurity may not be safe, but it does help...

EMULATOR DESIGN ISSUES

When designing a code emulator for forensic purposes, a number of special requirements must be met.

One problem to tackle is the multiple opcodes and multiple instructions issue:

```
87 C3      XCHG AX,BX
93         XCHG BX,AX
87 D8      XCHG BX,AX
```

The result is the same, but different opcodes are used.

```
PUSH AX    PUSH AX
PUSH BX    MOV  AX,BX
POP  AX    POP  BX
POP  BX
```

These give the same result. More than the five different code sequences shown above exist to exchange the contents of registers AX and BX. The technique of expressing the same functionality using many different sets of opcode sequences is used by encryptors generated by polymorphic engines. Some being over 200 bytes in size, they only contain the functionality of a cleanly coded decryptor of 25 bytes. Most of the remaining code is redundant, but sometimes seemingly redundant code is used to initiate registers for further processing.

It is the job of the emulator to make sure that the rule-based analyser gets the correct information, i.e. that the behaviour characteristics passed to the analyser reflect the actual facts. No matter which series of instructions/opcodes are used to perform 3D02h/21h, the analyser only has to know that the behaviour of that piece of code is:

Open a file for (both reading and) writing.

On the one hand, this may not seem that difficult. Most viruses do perform interrupt calls, and when they do, we just have to evaluate the contents of the registers to derive the behaviour characteristic. On the other hand, this is only correct if we talk about simple, straightforward viruses. For viruses using different techniques (hooking different interrupts, using call/jmp far constructions) it may be very difficult for the emulator to keep track of the instruction flow. In any case, the emulator must be capable of reducing instruction sequences to the bare functionality in a well-defined manner. We call the result of this reduction a *behaviour characteristic*, if it can be found in a pre-compiled list of characteristics to which we attach particular importance.

Another problem is that the emulator must be capable of making important decisions, normally based on incomplete evidence (we obviously want to emulate as little code as possible before reaching a conclusion regarding the potential maliciousness of the software in question).

Let us illustrate this with a small example:

```
MOV AX,4567
INT 21
CMP AX,7654
JNE jmp-1
JMP jmp-2
```

This is an example of an 'Are you there?' call used by a virus. When tracing through the code, the emulator obviously does not know whether jmp-1 or jmp-2 leads to the code which installs the virus in case it is not already there. So, should the emulator continue with the jmp-1 flow or the jmp-2 flow? Now, a simple execution of the code will result in just one of these flows being relevant, whereas a forensic emulator must be able to follow all possible program flows simultaneously, until either a flow leads to a number of relevant behaviour characteristics being detected, at which time the information is passed to the analyser, or a flow has been followed to a point where one of the stop-criteria built into the emulator is met. The strategy used in this part of the emulator is a determining factor when it comes to obtaining an acceptable scanning speed.

Hopefully, this has illustrated some of the problems associated with designing a forensic emulator. It is a very difficult and complex part of this set-up.

Once the emulator has finished its job it passes information, a list of behaviour characteristics which it has found in the code, on to the analyser.

BEHAVIOUR RULES

Before the analyser is able to compare the behaviour characteristics found by the emulator to information in its behaviour database, this database needs to be defined. Assume that we have a COM and an EXE file infecting virus with the following behaviour:

```
!    MODIFY FILE ATTRIBUTE REMOVING READ-ONLY FLAG
!    OPEN A FILE FOR (BOTH READING AND)WRITING
!*   WRITE DATA TO END OF FILE
!*   MODIFY ENTRY POINT IN HEADER or WRITE TO BEGINNING OF
     FILE
-    MODIFY FILE DATE AND FILE TIME
-    CLOSE FILE
-    MODIFY FILE ATTRIBUTE
```

If we want to develop a behaviour rule for this virus, it will look like this:

```
1.  MODIFY_FILE_ATTRIBUTE + OPEN_FILE
    + WRITE_DATA_TO_EOF +
    MODIFY_EP_IN_HEADER

2.  MODIFY_FILE_ATTRIBUTE + OPEN_FILE
    + WRITE_DATA_TO_EOF +
    WRITE_DATA_TO_BOF
```

where rule 1 is a rule for the EXE-file, and rule 2 for the COM-file.

Since a lot of viruses and virus source codes are widely available, a number of different instruction sequences resulting in this functionality will probably show up. Normally, derived viruses contain minor changes to bypass a single scanner by just changing the order of two or more instructions, but sometimes larger code sequences can be changed without changing the functionality of the virus. It is trivial to change the code, so it will first modify the entry-point in the header or change the start-up code, and afterwards write the virus code. In order to detect these changes (variants) the next rules may be added:

```
3.  MODIFY_FILE_ATTRIBUTE + OPEN_FILE
    + MODIFY_EP_IN_HEADER +
    WRITE_DATA_TO_EOF @CODE LINE =

4.  MODIFY_FILE_ATTRIBUTE + OPEN_FILE
    + WRITE_DATA_TO_BOF +
    WRITE_DATA_TO_EOF
```

Another example (an MBR infector):

```
-   PERFORM SELF CHECK
!   HOOK INT13
!   BECOME RESIDENT
!   INTERCEPT READ/WRITE TO MBR
!   READ MBR
-*  WRITE MBR TO OTHER LOCATION
!*  WRITE NEW MBR
```

Rule:

```
HOOK_INT13 + INTERCEPT
READ/WRITE_TO_MBR + WRITE_NEW_MBR
```

The signs in front of the descriptors in the examples above hint at the weighing procedure used by the analyser to attach significance to the behaviour characteristics supplied by the emulator. A '-' means that the characteristic does not have to be present, an '!' that it must be present (but does not in itself indicate malicious code). A '*' indicates a high weighing value. Thus '-*' means that the characteristic does not have to be present in the sequence of actions, but if it is, this is a highly important fact.

If rules 1 - 4 above are examined more closely, it can be concluded that they describe behaviour found in a number of viruses from different families.

A single behaviour rule may detect an unlimited number of viruses. That is the power behind using behaviour characteristics. While at present we in most cases need a new signature or new (changed) algorithm to detect a new variant of a virus or a new virus family, the behaviour characteristics will continue to do their work. This is extremely important, because it removes the necessity for the virus researcher and the anti-virus developer to react to a new virus unless it is technologically innovative. And those are few and far between.

Of course, some viruses will be developed which will not be caught by any of the rules in the behaviour database. These must be taken care of just like we do right now with any new virus; but instead of creating a signature, we create a new rule.

With a little luck, a new virus behaves like a virus already covered by a rule. If we attach a level of importance to each part of a behaviour characteristic, we can use this in the analyser to arrive at a conclusion. Depending on the level of importance of each individual component of a behaviour characteristic detected, the system may decide to give a message to the user, such as 'may be infected by an unknown virus', or 'suspicious code'.

The reason for attaching a level of importance to each individual part of a behaviour characteristic is that it makes it easier to sort out cases where combinations of individually innocent behaviour characteristics put together constitute malicious code – or vice versa. Filedate, from *Norton's Utilities,* is able to change file date and file time; as a matter of fact, this is the purpose of the utility. The ATTRIB command is developed to change file attributes. Evidently, changing file attributes is in itself insufficient evidence of malicious behaviour. A virus needs to write to a file as well. So a file write is mandatory for code to be considered suspicious and is heavily weighted. A change of attributes is not that important, and thus given a lower weighting.

If the user so wishes, the file or part of the (decrypted) code on which the analysing system triggered can be checked by a signature scanner to see if a virus can be identified.

CREATING RULES AUTOMATICALLY

An important part of the system is a *Rule Building Utility.* Whenever a new virus or Trojan emerges, it may be processed by this utility, which is similar to the emulator, albeit with some important differences. The emulator only collects behaviour information without knowing anything about the importance of a particular type of behaviour, or if the behaviour is suspicious.

The Rule Building Utility has to learn the level of importance of behaviour characteristics, has to know which behaviour is mandatory for a virus or Trojan, which behaviour is used by a virus but may be omitted, etc. Because research and development time is very expensive, the utility must be able to remember this for similar behaviour characteristics, and only ask for additional unknown information when needed, saving the researcher valuable time.

```
Behaviour A:             Behaviour B:

SEARCH FIRST FILE        SEARCH FIRST FILE
DELETE FILE              DELETE FILE
SEARCH NEXT FILE         CREATE NEW FILE
                         WRITE CODE INTO FILE
                         SEARCH NEXT FILE
```

When rules have been defined for behaviour B and a file (behaviour A, which was reported being suspicious) is processed, the utility must be able to realise that this behaviour is not as indicative of potential maliciousness as behaviour A. As a matter of fact, if behaviour A is taken on its own, it might well be a DEL *.* command.

At first, the utility will, ask for input frequently, because it needs to build up its database. However, over a period of time this type of utility should make life easier for the researcher.

CONCLUSION

The number of viruses is increasing rapidly: this is a known fact. The time will soon arrive when scanning using signatures and dedicated algorithms will either use too much memory or just become too slow. With storage media prices dropping fast, lots of systems now come equipped with very large hard disks, which will take more and more time, and thus money, to scan using traditional techniques. A properly designed rule-based analysing system feeding suspicious code to a scanner, which can identify the suspicious code as a known virus or Trojan, or perhaps dangerous code needing further investigation, is bound to save a lot of time.

Although it is impossible to prove that code is not malicious without analysing it from one end to the other, we in *Computer Security Engineers Ltd* believe it possible to reduce significantly the time used to check files by using all the available system knowledge instead of only small bits of it, as it is done today. Using virus scanning as the primary, or in many cases the only, anti-virus defence is an absurd waste of time and money, and furthermore blatantly insecure!

ABOUT CSE

Computer Security Engineers Ltd is one of the pioneers of anti-virus system development. The anti-virus system PC Vaccine Professional was first published in 1987, and since the start of 1988 a new version has been published each and every month. From 1988, cryptographic checksumming was introduced as the primary line of defence, scanning as the second. In 1992, the emphasis shifted, and behaviour blocking was introduced as the first line of defence, followed by checksumming and – in the case of an alarm from one of these countermeasures or to examine incoming diskettes – scanning for known viruses. Most recently, the basic philosophies underlying PC Vaccine professional, or PCVP as the system is also known, were expanded into a powerful and easily-maintained network perimeter and in-depth defence based on the well-known military tenets of: (1) keep them out and (2) if you can't keep them out, find and destroy them as fast as possible.

VIRUS DETECTION – 'THE BRAINY WAY'

Glenn Coates & David Leigh

Staffordshire University, School of Computing, PO Box 334, Beaconside, Stafford, ST18 0DG, UK
Tel +44 1782 294000 · Fax +44 1782 353497

ABSTRACT

This paper explores the potential opportunities for the use of Neural Networks in the detection of computer viruses.

Neural computing aims to model the guiding principles used by the brain for problem solving, and apply them to a computing domain. It is not known how the brain solves problems at a high level; however, it is widely known that the brain uses many small highly interconnected units called 'neurons'.

Like the brain, a neural network can be trained to solve a particular problem or recognise a pattern by example. The outcome is an algorithm-driven recogniser which does not exhibit the same behaviour as a deterministic algorithm. According to the way in which it has been trained, it may make 'mistakes'. That is, it may declare a positive result for a sample which is actually negative, and vice-versa. The ratio of correct results to incorrect ones can usually be improved by more or better training.

Can such pattern recognition be harnessed to the use of virus detection? It could be argued that the characteristics of virus patterns, no matter how they are expressed, are suitable subjects for detection by Neural Networks.

INTRODUCTION

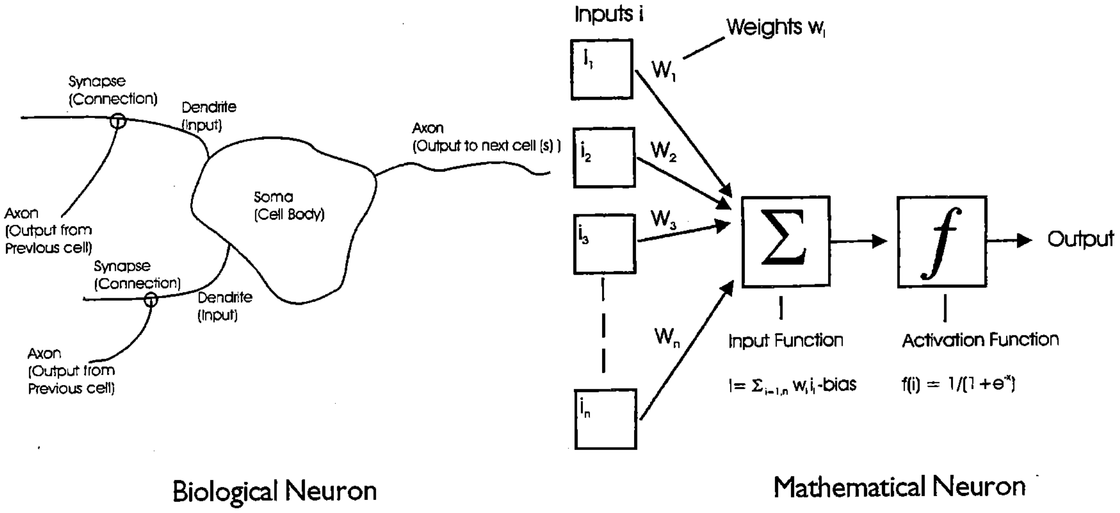
The received wisdom is that neural computing is an interesting 'academic toy' of little use, apart from modelling the animal brain. If this is true, then it is surprising that 7 out of 10 of the UK's leading blue chip companies are either investigating the potential of neural computing technology or are actually developing neural applications [Con94]. If leading edge companies are prepared to spend money on this 'academic toy', then maybe there are advantages to be gained from its use.

Without investigating new techniques (for example heuristic scanning), one must accept that the rapid rise in new viruses will exert a heavy speed penalty from existing virus scanners. As a result of this rise in virus numbers and sophistication, there will be an increasing conflict between acceptable speed and acceptable accuracy. It is easy to become complacent and rely on increasing processor power to bail us out of this problem, but processor design is increasingly becoming a mature technology.

What follows are the results of a feasibility study into the utilisation of neural networks within the field of virus detection.

WHAT IS A NEURAL NETWORK?

The workings of the brain are only known at a very basic level. It contains approximately ten thousand million processing units called neurons, each of these neurons is connected to approximately ten thousand others. This network of neurons forms a highly complex pattern recognition tool, capable of conditional learning. Figure 1 illustrates a model of the biological neuron alongside its corresponding mathematical model.



Biological Neuron

Mathematical Neuron

Figure 1

The individual neuron is stimulated by one or more inputs. In the biological neuron, some inputs will tend to excite the neuron, whilst others may be inhibitory. That is to say, some carry more 'weight' than others. This is mirrored in the mathematical model via the use of a 'weighting mechanism'. The neuron accumulates the total value of its inputs, before passing through a threshold function to determine its final output. This output is then fired as an input to another (or a number of) neurons, and so on. In the biological neuron, the axon performs the threshold function. The mathematical model would typically use a sigmoid function or a simple binary 'yes/no' threshold function. The reader is referred to [Mar93] for further discussion.

NEURAL NETWORK DEVELOPMENT

When approaching a problem using a neural network, it is not always necessary to know in detail what is to be done before planning its use. In this sense, they are quite unlike procedurally-based computer programs, which have to be written with a distinct goal in mind if they are to work properly. It is not even like a declarative program, for the same rule should apply. It is, perhaps, more like an expert system, where the outcome depends on the way in which an expert has answered a pre-defined series of questions.

In this approach, a 'standard' three-layer neural network is constructed using the 'back propagation' learning algorithm. The architecture consists of an input layer, a hidden layer, and an output layer. Training is carried out by submitting a 'training set' of data to the network's input, observing what output is given, and adjusting the variable weights accordingly. Each neuron in the network processes its inputs, with the resultant values steadily 'percolating' through the network until a result is given by the output layer. This output result is then compared to the actual result required for the given input, giving an error value. On the basis of this error value, the weights in the network are gradually adjusted, working backwards from the output layer. This process is repeated until the network has 'learnt' the correct response for the given input [DTI95]. Figure 2 illustrates this.

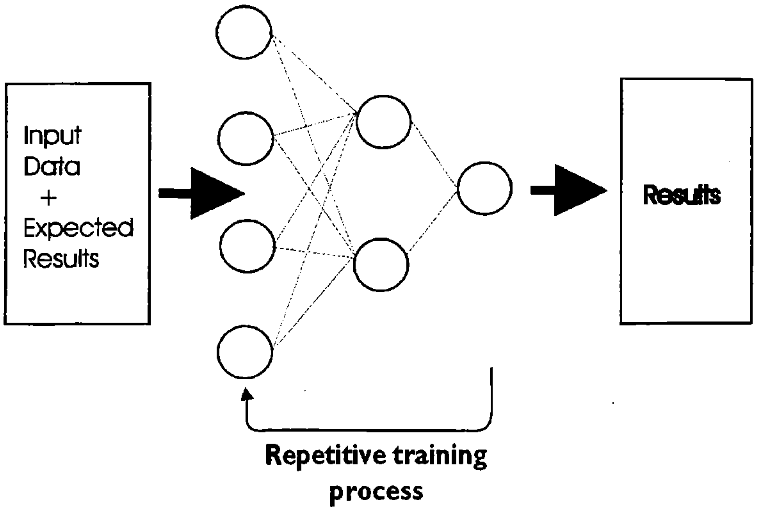


Figure 2

In this instance, the inputs represent the virus information, or other data concerning a virus-infected file. There are only two possible outputs, corresponding to 'possible virus found' and 'file appears to be OK'. The training data is divided into two classes, one containing the data for an infected file; and the other, uninfected files. When a suitable output is generated for the training data, the network is checked with a separate 'validation set'. If the output for the validation set is not acceptable, it is merged with the original training set and the entire process is repeated . This process is described schematically in figure 3.

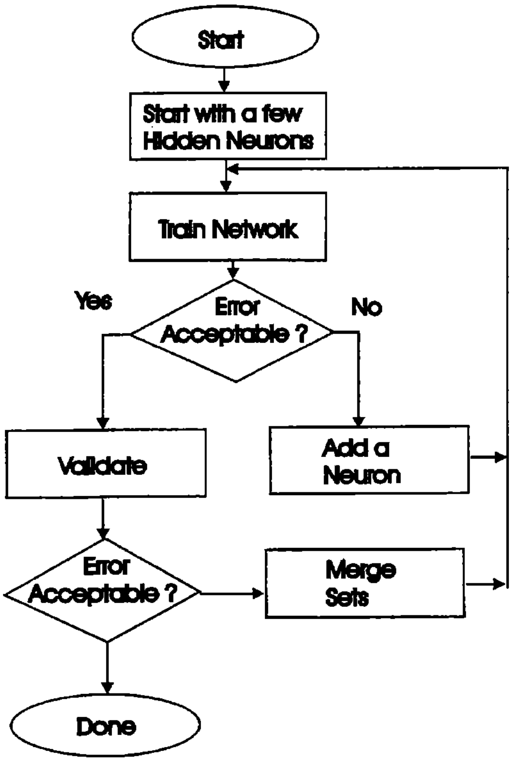


Figure 3

The result should be a very robust fuzzy recogniser capable of coping with unseen data. Because neural networks can process deeply hidden patterns, some have provided decisions superior to those made by trained humans.

EXISTING SYSTEMS

In 1990, a neural network was developed which acted as a 'communications link' between the mass of virus information available and end-user observations. By answering a set of standard questions regarding information on virus symptoms, the virus could be classified, and a set of remedies given. Due to the nature of neural networks, the system could cope with incomplete and erroneous data provided by the end user. Even when faced with a new mutation, the system still gave suitable counter-measures and information. See [Gui91] for a full discussion.

IDENTIFICATION OF VIRUS CODE PATTERNS VIA NEURAL NETWORKS

A neural network could be constructed to learn the actual machine code patterns of a specific virus. However, as most viruses are mutations of existing viruses, a network could be made to identify a virus family. This carries the advantage of being capable of identifying future variants. This would result in a set of sub-networks linked together to provide the end solution.

At the lowest level this could be done at the bit level. Figure 4 illustrates this.

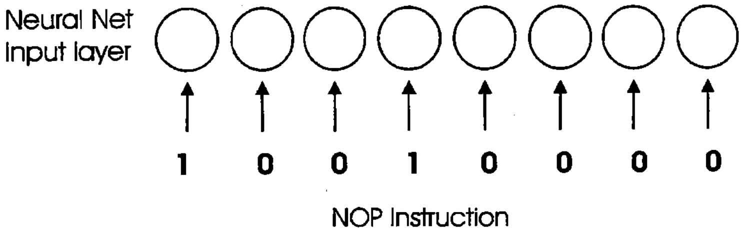


Figure 4

Although recognition at this level would be very difficult (if not impossible for a human) a neural network would be capable of it. The only limiting factors would be the volume and quality of the training data. The number of input neurons for a ½K virus code segment with a one-neuron output would be 4096. Given this, according to the 'geometric pyramid rule', the number of neurons in the hidden layer would be 64.

The number of virus samples for effective recognition would be in the region of at least 525,000. This figure should then be trebled for the number of non-infected files. Others would argue far more, due to the problems associated with false positives.

At a higher level, the input data could be presented at the byte level, where each byte would correspond to a single input neuron. In this context, the number of hidden neurons would be reduced to 22, and the number of virus samples would be at least 23,000. Again, the same applies for the number of non-infected files. This figure could be reduced further by pre-processing the code segment by extracting operand information, which could also increase accuracy and training time.

The *British Technology Group,* with the involvement of *Oxford University,* conducted research into such a solution. Although no formal documentation was produced, the results are believed to be negative.

From this, it can be seen that the use of neural networks in virus detection only seems practical at a high level. After all, a virus expert armed with a 'Virus Detection Language' and a 'Generic Decryption Engine' can provide a 100% accurate scanning result with advanced polymorphic viruses such as Pathogen in a relatively short period of time.

A NEURAL NETWORK POST-PROCESSOR

Rather than utilising a neural network to solve the virus alone, one could be used to process high level information, for example, that generated by a heuristic scanner.

Currently, most heuristic scanners use a form of emulation in order to determine the behaviour of a program file. Should that program appear to execute a suspicious activity, a 'flag' is set indicating this. However, some of these flags indicate more virus-like activity than others. In order to solve this problem, the flags are weighted via a score. Therefore, a flag indicating a 'suspicious memory reference' may be given more weighting than a flag indicating an 'inconsistent EXE header'. The total weights of the set flags are computed, and if a set threshold value is met, the heuristic scanner issues a suitable warning.

In the example of a well-known heuristic scanner, 35 of these flags are used. The weights are applied on an experimental basis. Initially, the weights are applied using a 'best-guess' approach, based on the virus experts' knowledge. The results of this are then tested on a virus collection and on a clean set of files. The results are analysed, and the weights adjusted accordingly. This cycle continues until satisfactory results are obtained. Figure 5 illustrates this cycle.

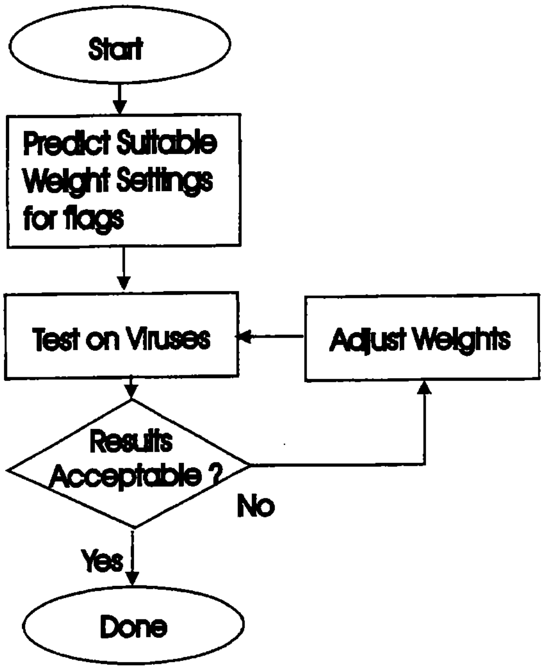


Figure 5

This process will probably increase in complexity over the next few years. In the above example, the number of flags could literally double due to the increase in knowledge, new techniques employed by the virus writers, and further development of heuristic scanners. It is imminent that the cycle of adjust, test, re-adjust will become far more complex and time-consuming. For example, why should flag-x be given a weight of 8, and not 7 or 9, and flag-y be given a weight of 1, and not 2?

Already, one can see that the illustrated cycle is very similar in nature to that used in neural network training. Indeed, a neural network could be used in place of the weighting mechanism and bias imposed by the virus expert. Based on the results of other neural network applications, the results should be very accurate, because the neural network will 'learn' the 'optimum' weights. The human element is removed, and the entire learning process is automated.

In terms of network size, the number of input neurons would be 35, with 6 hidden neurons, and 1 output neuron. In theory, the minimum number of infected file samples required for training would be at least 432. However, there would be no detrimental effects from training the network with higher samples, in order to reflect current virus numbers.

CONCLUSIONS

Neural computing is no longer seen as a pure academic subject. Indeed, many companies are now looking towards the use of neural networks as serious tools. Many systems are currently in use, with very high success rates.

It has been found that it may be feasible to use neural computing technology in the virus detection field. However, at a low level the results are unclear. There seems to be greater accuracy using deterministic techniques.

Using a neural network as a pre-/post-processing tool could offer a powerful addition to the virus expert's toolbag. Just one example is with the heuristic scanner. The authors believe other uses will also exist.

ACKNOWLEDGMENTS

As Bernard of Chartres said, echoed by Sir Isaac Newton: 'If [we] have seen further, it is by standing on the shoulders of giants'. The assistance given by the following people is gratefully acknowledged: Jan Hruska, Frans Veldman, Alan Solomon, Martin Slade, Robert Mortimer and Michael Twist. The continuing support of the staff at *Staffordshire University* and at *Visionsoft* has also been gratefully appreciated.

REFERENCES

[Con94] 'Adopting The Neural Approach', *Control Magazine*, Issue 5, March/April 1994.

[DTI95] *UK Department Of Trade And Industry*, Neural Computing Technology Programme, 1995.

[Gui91] Dr. Daniel Guinier, 'Computer 'virus' identification by neural networks', *SIGSAG*, 1991.

[Mar93] Timothy Masters, 'Practical Neural Networks in C++', *Academic Press*, 1993. ISBN 0-12-479040-2 Further Reading.

'Neural Computing – an introduction', R Beale and T Jackson, *IOP Publishing*, 1990. ISBN 0-85274-262-2.

Vesselin Bonchev, Future Trends in Virus Writing, *Proceedings of the Fourth International Virus Bulletin Conference*, 1994.

Glenn Coates and David J. Leigh, 'Virus Detection using a Generalised Virus Description Language', *Proceedings of the Fourth International Virus Bulletin Conference*, 1994.

DATA SECURITY PROBLEMS ASSOCIATED WITH HIGH CAPACITY IDE HARD DISKS

Roger Riordan

Cybec Pty Ltd, PO Box 205, Hampton, VIC 3188, Australia
Tel +61 3 521 0655 · Fax +61 3 521 0727 · Email riordan@tmxmelb.mhs.oz.au

INTRODUCTION

Every article on viruses starts with the advice 'Before running any anti-viral or integrity checking software you should always boot from a clean DOS floppy disk, to ensure there are no stealth viruses in memory where they could interfere in the test'. Just before we sent out our February update my eye was caught by a reference to problems being caused by special drivers used with large IDE drives. By chance we had just bought a 1G hard disk for a home PC, so I took a copy of VET home and installed it. Then I booted from a floppy. When I ran an uninstalled copy of VET it was unable to check drive C. This was bad enough, as it meant the traditional advice was useless. But then I ran VET from the reference disk I had just made. It announced 'Master Boot Record has been changed; would you like to replace it?' Clearly we had a problem.

This paper will discuss how the problem has arisen, the nature of the problem, what it means to users, and what they can do to avoid the dangers these drives have introduced.

BACKGROUND

Engineers traditionally regard a safety factor of 10 as conservative; if you are designing a bridge, or a building, you think of the largest conceivable load and then design the structure to handle something 10 times as big. This thinking has carried over into the computer industry; when IBM designed the PC most personal computers had 64K of memory, so they allocated a luxurious 640K. Unfortunately the unprecedented speed of development has rendered this thinking dangerously inadequate. Sizes of memory chips and disk drives have been doubling every two to three years, and so a safety factor of 10 will be used up in only three to four years. Given that it may well take a couple of years to bring a new design to market, the old joke about the computer shop with the banner 'New Model', and the salesman saying 'It's on sale; it's superseded', is all too likely to be true.

When Microsoft designed the disk handling logic for MSDOS they had to decide how to allocate all the various steps on the way from the user asking to read a particular file to the disk controller being told to read a particular sector. The first mistake they made was to make DOS do the low level sector handling, instead of leaving it to the BIOS, and the next was in allocating disk parameters to registers in the call to Int 13, which provides the interface between DOS and the BIOS [1].

The scheme they used was:

AH	Read/write command	AL	Number of sectors to read
CH	Number of first cylinder	CL	Number of first sector
DH	Starting Head	DL	Drive

This was OK for floppies, but when IBM introduced the XT (in about 1985) this allocation meant that the number of the starting cylinder could not be more than 255. This was too small, so Microsoft decided to steal the top two bits of CL, and add them to CH, increasing the maximum number of cylinders to 1024, but limiting the maximum number of sectors to 63. This was messy, but not critical. The BIOS extensions to handle the hard disk were in ROM on the disk controller card.

Then, almost immediately, the AT was introduced, and the BIOS was extended to handle the hard disk directly. The extension loaded the various parameters into registers in the disk controller, using I/O instructions to port addresses in the range 1F1 to 1F6, and finally passing a read/write command to port 1F7. The allocation used was:

1F2	Number of sectors to read	1F3	Number of first sector
1F4	Number of 1st cylinder (low)	1F5	Number of 1st cylinder (high)
1F6	Head (bits 0-3 only)	1F7	I/O command.

This arrangement was fine in itself, as it would handle any conceivable size of drive, but it was seriously incompatible with the Int 13 interface, as the maximum values which could pass unmodified through both interfaces were:

Number of sectors	63	(6 bits from CL in Int 13)
Number of cylinders	1024	(8 bits from CH, two from CL in Int 13)
Number of heads	16	(4 bits of 1F6 in drive controller)
Length of sector	512	(DOS standard)

Thus the maximum disk size had effectively been limited to 16*1024*63*512 = 528.482 Mbyte. This was not noticed at the time, as hard disks were typically 5 or 10 Mbytes, but by the end of the 80s hard disks with up to I Gbyte were becoming increasingly common. At first these used the SCSI interface, but this required a non-standard BIOS and a relatively expensive interface card. Meanwhile the far simpler IDE interface had become standard, and had improved to the point where the SCSI interface offered little, if any, improvement in performance, and there was a strong incentive to devise a way for big drives to be used with it, without having to replace the BIOS.

In 1994, several companies devised ways around this limitation, and inexpensive high capacity (ie greater than 528 Mbyte) IDE drives began to appear [2]. Most of these use variations of a scheme referred to as Extended CHS (Cylinder, Head, Sector) addressing, and replace the normal Master Boot Record with a special boot sector. This loads an extension to the BIOS from the normally unused area following the MBR. This copies itself to the top of normal memory (or to an area in high memory), hooks Int 13, and then loads a normal MBR. After this the PC boots normally, but all calls to Int 13 are caught, and the parameters translated before they are passed on to the drive controller.

When IBM designed the XT they put the MBR in sector I of head 0, cylinder 1, and left the whole of the rest of this track empty. However many ‘compatibles’ put the DOS boot sector in sector 2, with the File Allocation Table (FAT) following immediately after it. This incompatibility was not a significant problem until 1988, when the unknown authors of the Stoned virus realised this unused area made an ideal place for their virus to hide the original MBR.

This was fine on true clones, but when Stoned infected one of these non-standard PCs it overwrote the FAT, with disastrous results. Indeed I became involved in the AV industry only because the college where I worked had a lab full of Olivetti M24s, and these were crashing faster than the technicians could re-install the software on them.

Stoned, and its many descendants, have provided a strong incentive to the industry not to put anything important in track zero, and it has come to be regarded as a convenient working area. However, as we have noted, the special device drivers for the big IDE drives are loaded here, and so the old incompatibility, which rendered Stoned so serious, has been re-introduced, without any warning to the unsuspecting customers buying these drives [3].

THE DRIVERS

We believe that at least three drivers are in use, but only two seem to be common in Australia. These are:

1. Disk Manager, written by *Ontrack Computer Systems*, and sold with Western Digital drives, and
2. EZ-Drive, written by *Micro House International, Inc*, and sold with Conner drives.

Both have a non-standard MBR (in the normal location in track 0, head 0, sector 1), and store the body of the driver in track zero, head zero, but with EZ-Drive the rest of the drive is arranged normally, whereas Disk Manager moves the whole of the normal disk structure down one head (so that the normal MBR is in Cylinder 0, Head 1, Sector 1, and the DOS boot sector is in Cylinder 0, Head 2, Sector 1, and so on. For want of a better name we refer to the non-standard MBR as the Extended Boot Record (EBR). Thus in this paper, the EBR is the sector loaded first, while the MBR is the sector loaded after the driver has been installed. In normal use the EBR is invisible, and utilities (and viruses) will only see the MBR, which will appear to be in the normal location.

Disk Manager occupies sectors 2 to 30, leaving sector 7 free for Stoned (the designers had apparently not heard of the many viruses which use other sectors), and sector 63 has some data labelled ‘Disk Tables’. It is likely this location was chosen because it is relatively unlikely to be overwritten by accident.

EZ-Drive occupies sectors 2 to 14, with no provision for viruses. Sector 2 contains the ‘normal’ MBR. This has no boot program at all; everything but the actual partition record and sector marker is zeroed. There are several tables which could be drive data. All these sectors are used by known viruses.

When they are active both drivers ‘stealth’ the EBR, and show the MBR, with apparently normal partition information. Thus any virus which infects the EBR, and is compatible with the driver, will be hidden from normal anti-viral software.

Both drivers offer to ‘boot from a floppy’ after they have installed themselves (and any compatible viruses), so that you can then access the hard disk without running anything else from it. The On Track driver is already compatible with quite a few viruses, and will protect them from older AV software, and the dark side will probably try to produce more ‘compatible’ viruses to take advantage of this protection. But at least if the virus is compatible with the driver it means that the virus can be removed by the normal means after booting from a normal boot disk, without the AV software having to know about the driver.

CHANGE OF DRIVE LAYOUT

When a PC with one of these drives is booted from a floppy, not only the MBR, but also the actual physical layout of the drive, will appear to have been changed. Thus for two test drives:

Drive	Booted from floppy		Booted normally	
	Heads	Sectors/Track	Heads	Sectors/Track
Conner/Micro House 540Mb	15	17	15	63
Western Digital/Ontrack 1Gb	6	55	16	63

This discrepancy is not too serious for the Micro House driver, as all the relevant information is contained in the first 17 sectors, and these can be read using Int 13. However the On Track driver appears to store the vital drive parameters in head zero, sector 63, and the normal MBR in head 1, sector 1, as seen when the driver is resident. But if you have booted from a floppy and want to read these using Int 13, you would have to read head 1, sectors 8 & 9.

In general the new address is not immediately obvious, though it can be determined either by interrogation of the disk controller, using poorly documented commands, or by reading successive sectors until the read command fails, and then moving to the next head. Fortunately however, the function to read using direct I/O commands can read an arbitrary number of consecutive sectors, regardless of the layout. Thus if we read 64 sectors, starting with head 0, sector 1, we read the whole of track zero, including the drive parameters, and also the normal MBR.

(As an interesting aside, despite what some utilities may report, it would appear that the initial values for the Western Digital drive are almost certainly the true physical values; it is difficult enough to imagine fitting three platters in a drive of that size, let alone eight!)

DISASTER CONTROL

Disk Manager has provision for generating a boot disk which loads the driver as a device driver, so that you can access the hard disk without running any software from it. Provided the EBR (and probably also the MBR) are intact this enables you to access the hard disk even if the device driver has been severely damaged. This feature is not mentioned in the documentation, but is offered as a menu option during installation.

EZ-Drive apparently cannot generate a boot disk of this type. However the installation menu does offer ‘Antiviral Boot Sector Code’. The help menu for this says:

> This feature will defend your system against boot sector viruses. This does not include all viruses, so do not rely exclusively on this or any single virus protection package. If a virus is ever detected, a message will be posted, the virus will be removed, and the machine will reboot.

As we will see, the value of this is problematic.

No documentation is supplied with EZ-Drive. The documentation supplied with Disk Manager gives no warning about incompatibility problems with either viruses or low level utilities.

EFFECT OF VIRUSES

The hard disk boot sectors can get infected in four different situations. These are:

i. The PC is booted directly from an infected floppy

ii. An infected floppy is inserted when the driver, after being loaded from the hard disk, offers to boot from a floppy

iii. The PC is booted from an On Track special boot disk

iv. A file infected with a multi-partite virus is run.

So far as the EZ-Drive driver is concerned these all have effectively the same result. In all cases we have tried, or can think of, the driver will be over-written and the PC rendered unbootable. In our tests the 'Antiviral Boot Sector Code' was never invoked, and it would seem that the key clause in the help menu is 'If a virus is ever detected ..'.

With the *On Track* driver, in case i. the virus will infect the EBR in head 0, sector 1, and will normally overwrite one or more sectors in the same track. Many viruses will destroy the driver, but Stoned and other viruses which only overwrite sector 7 will fit in the hole provided for them, and the PC will continue to boot. In this case the, virus will be installed in memory before the driver, and the infected sector will be hidden from normal AV software.

In all the other cases the virus will infect the normal MBR in head 1, sector 1, and overwrite one or more following sectors. As these are unused, the PC will boot normally. In this case, the virus will be installed in memory after the driver.

In either case, if the PC is able to boot, the virus will be able to spread.

If the EBR is infected, normal AV software will be able to remove the virus, after booting from a clean floppy, but the PC will remain unbootable until the device driver is replaced. This can most easily be done if a copy of track zero is saved during installation. Both drivers offer some facilities for repairing damage, but these have not yet been fully evaluated.

If the MBR becomes infected normal AV software should be able to remove the virus, provided it can disable the virus in memory. However if the PC is clean booted the AV software will not be able to find the virus, unless the boot disk installs the On Track driver.

It would appear that DOS boot sector infectors should operate normally, and should not pose any special problems.

Given this range of possibilities, disinfection of these drives poses considerable problems, especially when the software is being operated by unskilled users.

THE PROBLEM

If you are responsible for any PCs fitted with one of these drives, you should be aware of the following dangers.

1. Most anti-viral and integrity checking programs recommend that you boot from a clean DOS disk before running them, to ensure that no stealth viruses are in memory. However if you do so you will be unable to access your hard disk at all. Furthermore the MBR (and even the drive parameters) will be completely different, and if the software has taken copies of the normal MBR it may offer to replace it, with disastrous results.

2. Most boot sector viruses will destroy the driver, rendering the contents of the hard disk inaccessible
3. Some viruses will allow the PC to operate normally, but will be screened from normal AV software by the drivers
4. Some utility software uses the unused area on track zero as a work space, again making the hard disk inaccessible. We have already recovered several drives damaged by 'disk optimisers'.

HOW VET HANDLES THESE DRIVES

When VET 8.3 checks a hard disk, it reads the MBR using Int 13 in the normal way, and then it reads it again, using direct port access. This bypasses the driver (and probably many security products) and returns the true contents of head zero, sector 1. VET refers to the value obtained this way as the Extended Boot Record, or EBR. It then compares the results, and if they differ, assumes that it is dealing with a 'big' drive. In this case it reads the whole track under head zero, and saves it. VET also recognises the MBR and the EBR for both the On Track and Micro House drivers.

VET will detect and warn if the PC has been booted from a floppy, and has a facility to compare track zero with the template, and if necessary replace it. If VET is asked to check the boot sectors, and detects a virus in the EBR, and a template for the clean track is available, it will automatically offer to replace the damaged track.

COMPATIBILITY PROBLEMS

Windows 95, and probably Windows NT, issue a warning when VET uses direct I/O to read the EBR. This is not fatal, but may alarm users.

Some early PCs may give erroneous results when VET reads the EBR, and could be classified erroneously as big drives, and it is possible that some security products may either complain, or confuse VET.

THE SOLUTION

If you are in charge of any PCs using these drives (and which do not have the new motherboards described in the next section) there are a number of steps you should take. These include:

1. Prepare a rescue disk for these PCs. At the minimum this should include a copy of the special driver, the same version of DOS, and any utilities required to get your system running. If your anti-viral software permits it, make a separate rescue disk for each PC, and use it to save a copy of track zero on the disk. (And make sure you label the disks clearly, so you know which PCs they belong to.)

   I would strongly advise against buying any drive using a driver which does not permit you to prepare such a disk.
2. Ensure that all anti-viral, security and utility software in your organisation is aware of these drives. Be particularly wary of users with their cherished 'private' utilities hidden away.
3. Warn the users (again!) about the dangers viruses pose to their system, and ensure that up-to-date AV software is readily accessible at all times AND USED!
4. Review your backup procedures!

THE FUTURE

An ironic feature of this affair is that the problem these drivers overcome is a short term one. As well as the Extended Cylinder, Head, Sector addressing already described, the large drives can all be addressed using a second addressing mode, known as Logical Block Addressing (LBA).

In this system, which Microsoft should have introduced years ago, all addresses are passed to the drive simply as an absolute address. Instead of fiddling around with drive parameter tables, cylinders, heads, sectors and all the rest, DOS would simply ask 'Read 73 blocks, starting at block 34,156'. Thus DOS has much less book-keeping to worry about, as all the dirty details of mapping the request to the physical arrangement of the disk can be left to the drive controller.

This frees the drive maker from the restrictions imposed by CHS addressing, and makes it far simpler to introduce more sophisticated designs. For example, in existing drives the capacity is set by the inmost tracks, where the head is moving more slowly, so that the bits are packed more closely. LBA addressing will make it easier for the drive designers to increase the capacity by putting more sectors on the outer tracks where the head is moving faster.

This advance is being utilised in two ways.

The motherboards in the latest PCs feature a modified BIOS which traps Int 13 and converts the CHS address back to an LBA address, which it then passes on to the drive. This means these PCs can use the large drives, with existing versions of DOS, without encountering the problems already described.

Windows 95, and presumably equivalent versions of competing operating systems, will use LBA addressing throughout, and will be able to access the new drives directly.

Thus everyone in the industry is being forced to worry about a problem which will have been solved almost as soon as it has been introduced. Unfortunately the large number of these drives being fitted to existing PCs means that we will probably have to go on dealing with it for another ten years.

CONCLUSION

The computer industry generally, and the PC industry in particular, is notorious for its disregard for standards, and the number and inelegance of the 'kludges' it resorts to. These drivers are one more example of this; a kludge introduced to overcome a short-term problem which will cause many users to lose data, and raise problems for other sections of the industry for years to come.

Fortunately, knowledge, as usual, is power, and if you are aware of the problem, and use the right software, these drives should not cause you any serious problems.

REFERENCES

[1] 'IDE Takes Off', John Bryan, *Byte*, March 1994

[2] 'Breaking the 528 MB DOS Barrier', Quantum Corp, *PC Update*, April 1995.

[3] 'Breaking the 528 MB DOS Barrier; the Downside', R.H.Riordan, *PC Update.* June 1995.

SCANNERS OF THE YEAR 2000: HEURISTICS

Dmitry O. Gryaznov

S&S International Plc, Alton House, Gatehouse Way, Aylesbury, Bucks, HP19 3XU, UK
Tel +44 1296 318700 · Fax +44 1296 318777 · Email grdo@sands.co.uk

INTRODUCTION

At the beginning of 1994, the number of known MS-DOS viruses was estimated at around 3,000. One year later, in January 1995, the number of viruses was estimated at about 6,000. By the time this paper was written (July 1995), the number of known viruses exceeded 7,000. Several anti-virus experts expect this number to reach 10,000 by the end of the year 1995. This large number of viruses, which keeps growing fast, is known as the glut and it does cause problems to anti-virus software – especially to scanners.

Today, scanners are the most frequently used type of anti-virus software. The fast-growing number of viruses means that scanners should be updated frequently enough to cover new viruses. Also, as the number of viruses grows, so does the size of the scanner or its database, and in some implementations the scanning speed suffers.

It was always very tempting to find a final solution to the problem; to create a generic scanner which can detect new viruses automatically without the need to update its code and/or database. Unfortunately, as proven by Fred Cohen, the problem of distinguishing a virus from a non-virus program is algorithmically unsolvable as a general rule.

Nevertheless, some generic detection is still possible, based on analysing a program for features typical or not typical of viruses. The set of features, possibly together with a set of rules, is known as heuristics. Today, more and more anti-virus software developers are looking towards heuristical analysis as at least a partial solution to the glut problem.

Working at the Virus Lab, *S&S International Plc*, the author is also carrying out a research project on heuristic analysis. The article explains what heuristics are. Positive and negative heuristics are introduced and some practical heuristics are represented. Different approaches to a heuristical program analysis are discussed and the problem of false alarms is explained and discussed. Several well-known scanners employing heuristics are compared (without naming the scanners) both virus detection and false alarms rate.

1 WHY SCANNERS?

If you are following computer virus-related publications, such as the proceedings of anti-virus conferences, magazine reviews, anti-virus software manufacturers' press releases, you read and hear mainly 'scanners, scanners, scanners'. The average user might even get the impression that there is no anti-virus software other than scanners. This is not true. There are other methods of fighting computer viruses – but they are not as popular or as well known as scanners; and anti-virus packages based on non-scanner technology do not sell well. Sometimes people who are trying to promote non-scanner based anti-virus software even come to the conclusion that there must be some kind of an international plot of popular anti-virus scanner producers. Why is this? Let us briefly discuss existing types of anti-virus software. Those interested in more detailed discussion and comparison of different types of anti-virus software can find it in [*Bontchev1*], for example.

1.1 SCANNERS

So, what is a scanner? Simply put, a scanner is a program which searches files and disk sectors for byte sequences specific to this or that known virus. Those byte sequences are often called *virus signatures*. There are many different ways to implement a scanning technique; from the so-called 'dumb' or 'grunt' scanning of the whole file, to sophisticated virus-specific methods of deciding which particular part of the file should be compared to a virus signature. Nevertheless, one thing is common to all scanners: they detect only *known* viruses. That is, viruses which were disassembled or analysed and from which virus signatures unique to a specific virus were selected. In most cases, a scanner cannot detect a brand new virus until the virus is passed to the scanner developer, who then extracts an appropriate virus signature and updates the scanner. This all takes time – and new viruses appear virtually every day. This means that scanners have to be updated frequently to provide adequate anti-virus protection. A version of a scanner which was very good six months ago might be no good today if you have been hit by just one of the several thousand new viruses which have appeared since that version was released.

So, are there any other ways to detect viruses? Are there any other anti-virus programs which do not depend so heavily on certain virus signatures and thus might be able to detect even new viruses? The answer is yes, there are: *integrity checkers* and *behaviour blockers (monitors)*. These types of anti-virus software are almost as old as scanners, and have been known to specialists for ages. Why then are they not used as widely as scanners?

1.2 BEHAVIOUR BLOCKERS

A behaviour blocker (or a monitor) is a memory-resident (TSR) program which monitors system activity and looks for virus-like behaviour. In order to replicate, a virus needs to create a copy of itself. Most often, viruses modify existing executable files to achieve this. So, in most cases, behaviour blockers try to intercept system requests which lead to modifying executable files. When such a suspicious request is intercepted, a behaviour blocker, typically, alerts a user and, based on the user's decision, can prohibit such a request from being executed. This way, a behaviour blocker does not depend on detailed analysis of a particular virus. Unlike a scanner, a behaviour blocker does not need to know what a new virus looks like to catch it.

Unfortunately, it is not that easy to block all the virus activity. Some viruses use very effective and sophisticated techniques, such as tunnelling, to bypass behaviour blockers. Even worse, some legitimate programs use virus-like methods which could trigger a behaviour blocker. For example, an install or setup utility is often modifying executable files. So, when a behaviour blocker is triggered by such a utility, it's up to the user to decide whether it is a virus or not – and this is often a tough choice: you would not assume that all users are anti-virus experts, would you?

But even an ideal behaviour blocker (there is no such thing in our real world, mind you!), which never triggers on a legitimate program and never misses a real virus, still has a major flaw. To enable a behaviour blocker to detect a virus, the virus must be run on a computer. Not to mention the fact that virtually any user would reject the very idea of running a virus on his/her computer, by the time a behaviour blocker catches the virus attempting to modify executable files, the virus could have triggered and destroyed some of your valuable data files, for example.

1.3 INTEGRITY CHECKERS

An integrity checker is a program which should be run periodically (say, once a day) to detect all the changes made to your files and disks. This means that, when an integrity checker is first installed on your system, you need to run it to create a database of all the files on your system. During subsequent runs, the integrity checker compares files on your system to the data stored in the database, and detects any changes made to the files. Since all viruses modify either files or system areas of disks in order to replicate, a good integrity checker should be able to spot such changes and alert the user. Unlike a behaviour blocker, it is much more difficult for a virus to bypass an integrity checker, provided you run your integrity checker in a virus clean environment – e.g. having booted your PC from a known virus-free system diskette.

But again, as in the case of behaviour blockers, there are many possible situations when the user's expertise is necessary to decide whether changes detected are the result of virus activity. Again, if you run an install or setup utility, this normally results in modifications to your files which can trigger an integrity checker. That is, every time you install new software on your system, you have to tell your integrity checker to register these new files in its database.

Also, there is a special type of virus, aimed specifically at integrity checkers – so-called *slow infectors*. A slow infector only infects objects which are about to be modified anyway; e.g. as a new file being created by a compiler. An integrity checker will add this new file to its database to watch its further changes. But in the case of a slow infector, the file added to the database is infected already!

Even if integrity checkers were free of the above drawbacks, there still would be a major flaw. That is, an integrity checker can alert you only **after** a virus has run and modified your files. As in the example given while discussing behaviour blockers, this might be well too late...

1.4 THAT'S WHY SCANNERS!

So, the main drawbacks of both behaviour blockers and integrity checkers, which prevent them from being widely used by an average user, are:

1. Both behaviour blockers and integrity checkers, by their very nature, can detect a virus only **after** you have run an infected program on your computer, and the virus has started its replication routine. By this time it might be too late – many viruses can trigger and switch to destructive mode **before** they make any attempts to replicate. It's somewhat like deciding to find out whether these beautiful yet unknown berries are poisonous by eating them and watching the results. Gosh! You would be lucky to get away with just dyspepsia!
2. Often enough, the burden to decide whether it is a virus or not is transferred to the user. It's as if your doctor leaves **you** to decide whether your dyspepsia is simply because the berries were not ripe enough, or it is the first sign of deadly poisoning, and you'll be dead in few hours if you don't take an antidote immediately. Tough choice!

On the contrary, a scanner can and should be used to detect viruses **before** an infected program has a chance to be executed. That is, by scanning the incoming software prior to installing it on your system, a scanner tells you whether it is safe to proceed with the installation. Continuing our berries analogy, it's like having a portable automated poisonous plants detector, which quickly checks the berries against its database of known plants, and tells you whether or not it is safe to eat the berries.

But what if the berries are not in the database of your portable detector? What if it is a brand new species? What if a software package you are about to install is infected with a new, very dangerous virus unknown to your scanner? Relying on your scanner only, you might find yourself in big trouble. This is where behaviour blockers and integrity checkers might be helpful. It's still better to detect the virus while it's trying to infect your system, or even after it has infected but before it destroys your valuable data. So, the best anti-virus strategy would include all three types of anti-virus software:

- a scanner to ensure the new software is free of at least known viruses **before** you run the software
- a behaviour blocker to catch the virus **while** it is trying to infect your system
- an integrity checker to detect infected files **after** the virus has propagated to your system but not yet triggered.

As you can see, the scanners are the first and the most simply implemented line of anti-virus defence. Moreover, most people have scanners as **the only** line of defence.

2 WHY HEURISTICS?

2.1 GLUT PROBLEM

As mentioned above, the main drawback of scanners is that they can detect only **known** computer viruses. Six or seven years ago, this was not a big deal. New viruses appeared rarely. Anti-virus researchers were literally hunting for new viruses, spending weeks and months tracking down rumours and random reports about a new virus to include its detection in their scanners. It was probably during these times that a most nasty computer virus-related myth was born that anti-virus people develop viruses themselves to force users to buy their products and profit this way. Some people believe this myth even today. Whenever I hear it, I can't help laughing hysterically. Nowadays with two to three hundred new viruses arriving monthly, it would be total waste of time and money for anti-virus manufacturers to develop viruses. Why should they bother if new viruses arrive in dozens virtually daily, completely free of charge? There were about 3,000 known DOS viruses at the beginning of 1994. A year later, in January 1995, the number of viruses was estimated at least 5,000. Another six months later, in July 1995, the number exceeded 7,000. Many anti-virus experts expect the number of known DOS viruses to reach the 10,000 mark by the end of 1995. With this tremendous and still fast-growing number of viruses to fight, traditional virus signature scanning software is pushed to its limits [*Skulason, Bontchev2*]. While several years ago a scanner was often developed, updated and supported by a single person, today a team of a dozen skilled employers is only barely sufficient. With the increasing number of viruses, R&D and Quality Control time and resource requirements grow. Even monthly scanner updates are often late, by one month at least! Many formerly successful anti-virus vendors are giving up and leaving the anti-virus battleground and market. The fast-growing number of viruses heavily affects scanners themselves. They become bigger, and sometimes slower. Just few years ago a 360Kb floppy diskette would be enough to hold half a dozen popular scanners, leaving plenty of room for system files to make the diskette bootable. Today, an average good signature-based scanner alone would occupy at least a 720Kb floppy, leaving virtually no room for anything else.

So, are we losing the war? I would say: not yet – but if we get stuck with just virus signature scanning, we will lose it sooner or later. Having realised this some time ago, anti-virus researchers started to look for more generic scanning techniques, known as *heuristics*.

2.1 WHAT ARE HEURISTICS?

In the anti-virus area, heuristics are a set of rules which should be applied to a program to decide whether the program is likely to contain a virus or not. From the very beginning of the history of computer viruses different people started looking for an ultimate generic solution to the problem. Really, how does an anti-virus expert know that a program is a virus? It usually involves some kind of reverse engineering (most often disassembly) and reconstructing and understanding the virus' algorithm: what it does and how it does it. Having analysed hundreds and hundreds of computer viruses, it takes just few seconds for an experienced anti-virus researcher to recognise a virus, even it is a new one, and never seen before. It is almost a subconscious, automated process. Automated? Wait a minute! If it is an automated process, let's make a program to do it!

Unfortunately (or rather, fortunately) the analytic capabilities of the human brain are far beyond those of a computer. As was proven by Fred Cohen [*Cohen*], it is impossible to construct an algorithm (e.g. a program) to distinguish a virus from a non-virus with 100 per cent reliability. Fortunately, this does not rule out a possibility of 90 or even 99 per cent reliability. The remaining one per cent, we hope to be able to solve using our traditional virus signatures scanning technique. Anyway, it's worth trying.

2.2 SIMPLE HEURISTICS

So, how do they do it? How does an anti-virus expert recognise a virus? Let us consider the simplest case: a parasitic non-resident appending COM file infector. Something like Vienna, but even more primitive. Such a virus appends its code to the end of an infected program, stores a few (usually just three) first bytes of the victim file in the virus body and replaces those bytes with a code to pass control to the virus code. When the infected program is executed, the virus takes control. First, it restores the original victim's bytes in its memory image. It then starts looking for other COM files. When found, the file is opened in Read_and_Write mode; then the virus reads the first few bytes of the file and writes itself to the end of the file. So, a primitive set of heuristical rules for a virus of this kind would be:

1. The program immediately passes control close to the end of itself
2. It modifies some bytes at the beginning of its copy in memory
3. Then it starts looking for executable files on a disk
4. When found, a file is opened
5. Some data is read from the file
6. Some data is written to the end of the file.

Each of the above rules has a corresponding sequence in binary machine code or assembler language. In general, if you look at such a virus under DEBUG, the favourite tool of anti-virus researchers, it is usually represented in a code similar to this:

```
START:                              ; Start of the infected program
          JMP VIRUSCODE             ; Rule 1:  the control is passed
                                    ; to the virus body
                                    ;

          <victim's code>

VIRUS:                              ; Virus body starts here

SAVED:                              ; Saved original bytes of the
                                      victim's code

MASK:     DB '*.COM',0              ; Search mask

VIRUSCODE:                          ; Start of the virus code
          MOV DI,OFFSET START       ; Rule 2: the virus restores
          MOV SI,OFFSET SAVED       ; victim's code
          MOVSW                     ; in memory
          MOVSB                     ;

        MOV DX,OFFSET MASK        ; Rule 3: the virus
MOV AH,4EH            ; looks for other
INT 21H               ; programs to infect

MOV AX,3D02H          ; Rule 4: the virus opens a file
INT 21H ;

MOV DX,OFFSET SAVED   ; Rule 5: first bytes of a file
MOV AH,3FH            ; are read to the virus
INT 21H               ; body

MOV DX,OFFSET VIRUS   ; Rule 6: the virus writes itself
MOV AH,40H            ; to the file
INT 21H               ;
```

Figure 1. A sample virus code

When an anti-virus expert sees such code, it is immediately obvious that this is a virus. So, our heuristical program should be able to disassemble a binary machine-language code in a similar manner to DEBUG, and to analyse it, looking for particular code patterns in a similar manner to an anti-virus expert. In the simplest cases, such as the one above, a set of simple wildcard signature string matching would do for the analysis. In this case, the analysis itself is simply checking whether the program in question satisfies rules 1 through 6; in other words, whether the program contains pieces of code corresponding to each of the rules.

In a more general case, there are many different ways to represent one and the same algorithm in machine code. Polymorphic viruses, for example, do this all the time. So, a heuristic scanner must use many clever methods, rather than simple pattern-matching techniques. Those methods may involve statistical code analysis, partial code interpretation, and even CPU emulation, especially to decrypt self-encrypted viruses: but you would be surprised to know how many real life viruses would be detected by the above six simple heuristics alone! Unfortunately, some non-virus programs would be 'detected' too.

2.3 FALSE ALARMS PROBLEM

Strictly speaking, heuristics do not detect viruses. As behaviour blockers, heuristics are looking for virus-like behaviour. Moreover, unlike the behaviour blockers, heuristics can detect not the behaviour itself, but just *potential ability* to perform this or that action. Indeed, the fact that a program contains a certain piece of code does not necessarily mean that this piece of code is ever executed. The problem of discovering whether this or that code in a program ever gets control is known in the theory of algorithms as the Halting Problem, and is in general unsolvable. This issue was the basis of Fred Cohen's proof of the impossibility of writing a perfect virus detector. For example, some scanners contain pieces of virus code as the signatures for which to scan. Those pieces might correspond to each and every one of the above six rules. But they are never executed – the scanner uses them just as its static data. Since, in general, there is no way for heuristics to decide whether these code pieces are ever executed or not, this can (and sometimes does) cause *false alarms*.

A false alarm is when an anti-virus product reports a virus in a program, which in fact does not contain any viruses at all. Different types of false alarms, as well as most widespread causes of false alarms, are described in [*Solomon*] for example. A false alarm might be even more costly than an actual virus infection. We all keep saying to users: 'The main thing to remember when you think you've got a virus – **do not panic**!' Unfortunately, this does not work well. The average user will panic. And the user panics even more if the anti-virus software is unsure itself whether it is a virus or not. In the case, say, where a scanner definitely detects a virus, the scanner is usually able to detect all infected programs, and to remove the virus. At this point, the panic is usually over; but if it is a false alarm, the scanner will not be able to remove the virus, and most likely will report something like: 'This file seems to have a virus', naming just a single file as infected. This is when the user really starts to panic. 'It must be a new virus!' – the user thinks. 'What do

I do?!' As a result, the user well might format his/her hard disk, causing himself a far worse disaster than a virus could. Formatting the hard disk is an unnecessary and un-justified act, by the way; even more so as there are many viruses which would survive this act, unlike legitimate software and data stored on the disk.

Another problem a false alarm can (and did) cause is negative impact on a software manufacturing company. If an anti-virus software falsely detects a virus in a new software package, the users will stop buying the package and the software developer will suffer not only profit losses, but also a loss of reputation. Even if it was later made known that it was a false alarm, too many people would think: 'There is no smoke without fire', and would treat the software with suspicion. This affects the anti-virus vendor as well. There has already been a case where an anti-virus vendor was sued by a software company whose anti-virus protection mistakenly reported a virus.

In a corporate environment, when a virus is reported by anti-virus software, whether it is a false alarm or not, the normal flow of operation is interrupted. It takes at best several hours to contact the anti-virus technical support and to ensure it was a false alarm before normal operation is resumed – and, as we all know, time is money. In the case of a big company, time is big money.

So, it is not at all surprising that, when asked what level of false alarms is acceptable (10 per cent? 1 per cent? 0.1 per cent?), corporate customers answer: 'Zero per cent! We do not want any false alarms!'

As previously explained, by its very nature heuristic analysis is more prone to false alarms than traditional scanning methods. Indeed, not only viruses but many scanners as well would satisfy the six rules we used as an example: a scanner does look for executable files, opens them, reads some data and even writes something back when removing a virus from a file. Can anything be done to avoid triggering a false positive on a scanner? Let's again turn to the experience of a human anti-virus expert. How does one know that this is a scanner, and not a virus? Well, this is more complicated than the above example of a primitive virus. Still, there are some general rules too. For example, if a program relies heavily on its parameters or involves an extensive dialogue with a user, it is highly unlikely that the program is a virus. This leads us to the idea of *negative heuristics*; that is, a set of rules which are true for a non-virus program. Then, while analysing a program, our heuristics should estimate the probability of the program to be a virus using both positive heuristics, such as the above six rules, and negative heuristics, typical for non-virus programs and rarely used by real viruses. If a program satisfies all our six positive rules, but also expects some command-line parameters and uses an extensive user dialogue as well, we would not call it a virus.

So far so good. Looks like we found a solution to the virus glut problem, right? Not really! Unfortunately, not all virus writers are stupid. Some are also well aware of heuristic analysis, and some of their viruses are written in a way which avoids the most obvious positive heuristics. On the other hand, these viruses include otherwise useless pieces of code, the only aim of which is to trigger the most obvious negative heuristics, so that such a virus does not draw the attention of a heuristical analyser.

2.4 VIRUS DETECTION VS. FALSE ALARMS TRADE-OFF

Each heuristic scanner developer sooner or later comes to the point when it is necessary to make a decision: 'Do I detect more viruses, or do I cause less false alarms?' The best way to decide would be to ask users what do they prefer. Unfortunately, the users' answer is: 'I want it all! 100 per cent detection rate and no false alarms!' As mentioned above, this cannot be achieved. So, a virus detection versus false alarms trade-off problem must be decided by the developer. It is very tempting to build the heuristic analyser to detect almost all viruses, despite false alarms. After all, reviewers and evaluators who publish their tests results in magazines read by thousands of users world-wide, are testing just the detection rate. It is much more difficult to run a good false alarms test: there are gigabytes and gigabytes of non-virus software in the world, far more than there are viruses; and it is more difficult to get hold of all this software and to keep it for your tests. 'Not enough disk space' is only one of the problems. So, let's forget false alarms and negative heuristics and call a virus each and every program which happens to satisfy just some of our positive heuristics. This way we shall score top most points in the reviews. But what about the users? They normally run scanners not on a virus collection but on a clean disks. Thus, they won't notice our almost perfect detection rate, but are very likely to notice our not-that-perfect false alarms rate. Tough choice. That's why some developers have at least two modes of operation for their heuristical scanners .The default is the so-called 'normal' or 'low sensitivity' mode, when both positive and negative heuristics are used and a program needs to trigger enough positive heuristics to be reported as a virus. In this mode, a scanner is less prone to false alarms, but its detection rate might be far below what is claimed in its documentation or advertisement. The often-used (in advertising) figures of 'more than 90 per cent' virus detection rate by heuristic analyser refer to the second mode of operation, which is often called 'high sensitivity' or 'paranoid' mode. It is really a paranoid mode: in this mode, negative heuristics are usually discarded, and the scanner reports as a possible virus any program which happens to trigger just one or two positive heuristics. In this mode, a scanner can indeed detect 90 per cent of viruses, but it also produces hundreds and hundreds of false alarms, making the 'paranoid' mode useless and even harmful for real-life everyday use, but still very helpful when it comes to a comparative virus detection test. Some scanners have a special command-line option to switch the paranoid mode on; some others switch to it automatically whenever they detect a virus in the normal low sensitivity mode. Although the latter approach seems to be a smart one, it takes just a single false alarm out of many thousands of programs on a network file server to produce an avalanche of false virus reports.

2.5 HOW IT ALL WORKS IN PRACTICE: DIFFERENT SCANNERS COMPARED

Being myself an anti-virus researcher and working for a leading anti-virus manufacturer, I have developed a heuristic analyser of my own. And of course, I could not resist comparing it to other existing heuristic scanners. We believe the results will be interesting to other people. They underscore what was said about both virus detection and false alarms rates. As the products tested are our competitors, we decided not to publish their names in the test results. So, only FindVirus of *Dr Solomon's AntiVirus Toolkit* is called by its real name. All the other scanners are referred to with letters: Scanner_A, Scanner_B, Scanner_C and Scanner_D. The latest versions of the scanners available at the time of the test were used. For FindVirus, it was version 7.50 – the first version to employ a heuristic analyser.

Each scanner tested was run in heuristics-only mode, with normal virus signature scanning disabled. This was achieved by either using a special command-line option, where available, or using a special empty virus signature database in other cases.

The test consisted of two parts: virus detection rate and false alarms rate. For the virus detection rate *S&S International Plc* ONE OF EACH virus collection was used, containing more than 7,000 samples of about 6,500 different known DOS viruses. For the false alarms test the shareware and freeware software collection of SIMTEL20 CD-ROM (fully unpacked), all utilities from different versions of MS-DOS, IBM DOS, PC-DOS and other known files were used (current basic *S&S* false alarms test set).

When measuring false alarms and virus detection rate, all files reported were counted; reported either as 'Infected' or 'Suspicious'. Separate figures for the two categories are given where applicable.

In both parts of the test, the products were run in two heuristic sensitivity modes, where applicable: normal or low sensitivity mode, and paranoid or high sensitivity mode. The automatic heuristic sensitivity adjustment was prohibited, where applicable.

The results of the tests are as follows:

Virus Detection Test

	Files	Files triggered (infected + suspicious)				
	scanned	Normal			Paranoid	
FindVirus	7375	5902	(N/A)	80.02%	N/A	
Scanner_D	7375	5743	(0+5743)	77.87%	6182 (0+6182)	83.54%
Scanner_C	7375	5692	(0+5692)	77.18%	N/A	
Scanner_A	7375	4250	(N/A)	57.63%	6491 (N/A)	87.74%
Scanner_B	7392(*)	3863	(2995+868)	52.38%	6124 (2992+3132)	82.68%

(*) Scanner_B was tested couple of days later, when 17 more infected files were added to the collection.

Table 1. Virus detection test results.

False alarms test

	Files	Files triggered (infected + suspicious)				
	scanned(*)	Normal			Paranoid	
FindVirus	13603	0	(N/A)	0.000%	N/A	
Scanner_A	13428	11	(N/A)	0.082%	371 (N/A)	2.746%
Scanner_B	13471	17	(0+17)	0.126%	382 (0+382)	2.836%
Scanner_D	13840	24	(0+24)	0.173%	254 (0+254)	1.824%
Scanner_C	13603	28	(0+28)	0.206%	N/A	

(*) Different number of files reported as scanned is due to the fact different products treat somewhat different sets of file extensions as executables.

Table 2. False alarms test results

3 WHY 'OF THE YEAR 2000'?

Well, first of all simply because I could not resist the temptation of splitting the name of the paper into three questions and using them as the titles of the main sections of his presentation. I thought it was funny. Maybe I have a weird sense of humour. Who knows...

On the other hand, the year 2000 is very attractive by itself. Most people consider it a distinctive milestone in all aspects of human civilisation. This usually happens to the years ending with double zero; still more to the end of a millennium, with its triple zero at the end. The anti-virus arena is not an exclusion. For example, during the EICAR'94 conference there were two panel sessions discussing 'Viruses of the year 2000' and 'Scanners of the year 2000' respectively. The general conclusion made by a panel of well-known anti-virus researchers was that, at the current pace of new virus creation by the year 2000, we well might face dozens (if not hundreds of thousands) of known DOS viruses. As I tried to explain in the second section of this paper (and other authors explained elsewhere [*Skulason, Bontchev2*]), this might be far too much for a current standard scanners' technique, based on known virus signature scanning. More generic anti-virus tools, such as behaviour blockers and integrity checkers, while being less vulnerable to the growing number of viruses and the rate at which the new viruses appear, can detect a virus only **when** it is already running on a computer or even only **after** the virus has run and infected other programs. In many cases, the risk of allowing a virus to run on your computer is just not affordable. Using a heuristic scanner, on the other hand, allows detection of most of new viruses with a regular scanner safe manner: **before** an infected program is copied to your system and executed. And very much like behaviour blockers and integrity checkers, a heuristic scanner is much more generic than a signature scanner, requires much rare updates, and provides an instant response to a new virus. Those 15-20 per cent of viruses which a heuristic scanner cannot detect could be dealt with using current well-developed signature scanning techniques. This will effectively decrease the virus glut problem five fold, at least.

Yet another reason for choosing the year 2000 and not, say, 2005 is that I have strong doubts whether the current computer virus situation will survive the year 2000 by more than a couple of years. With new operating systems and environments appearing (Windows NT, Windows'95, etc.) I believe DOS is doomed. So are DOS viruses. So is the modern anti-virus industry. This does not mean viruses are not possible for new operating systems and platforms – they are possible in virtually any operating environment. We are aware of viruses for Windows, OS/2, Apple DOS and even UNIX. But to create viruses for these operating systems, as well as for Windows NT and Windows'95, it requires much more skill, knowledge, effort and time than for the virus-friendly DOS. Moreover, it will be much more difficult for a virus to replicate under these operating systems. They are far more secure than DOS, if it is possible to talk about DOS security at all. Thus, there will be far fewer virus writers and they will be capable of writing far fewer viruses. The viruses will not propagate fast and far enough to represent a major problem. Subsequently, there will be no virus glut problem. Regrettably, there will be a much smaller anti-virus market, and most of today's anti-virus experts will have to find another occupation...

But until then, DOS lives, and anti-virus developers still have a lot of work to do!

REFERENCES

[*Bontchev1*] Vesselin Bontchev, 'Possible Virus Attacks Against Integrity Programs And How To Prevent Them', *Proc. 2nd Int. Virus Bulletin Conf.*, September 1992, pp. 131-141.

[*Skulason*] Fridrik Skulason, 'The Virus Glut. The Impact of the Virus Flood', *Proc. 4th EICAR Conf.*, November 1994, pp. 143-147.

[*Bontchev2*] Vesselin Bontchev, 'Future Trends in Virus Writing', *Proc. 4th Int. Virus Bulletin Conf.*, September 1994, pp. 65-81.

[*Cohen*] Fred Cohen, 'Computer Viruses – Theory and Experiments', Computer Security: A Global Challenge, Elsevier Science Publishers B. V. (North Holland), 1984, pp. 143-158.

[*Solomon*] Alan Solomon, 'False Alarms', *Virus News International*, February 1993, pp. 50-52.

COMPUTER VIRUSES AND ARTIFICIAL INTELLIGENCE

David J Stang

Norman Data Defense Systems Inc, 3028 Javier Road, Suite 201, Fairfax, VA 22031, USA
Tel +1 703 573 8802 · Fax +1 703 573 3919

INTRODUCTION

The world of computing has talked much about 'artificial intelligence', but unfortunately the last decade has not seen much intelligence in software. The task of defending systems against computer viruses is one in which artificial intelligence could certainly be applied, with potentially valuable results.

The purpose of this paper is to show how traditional anti-virus practices (namely, scanning) cannot keep up in today's more sophisticated computer virus era. Instead, one must look towards advanced techniques for generic prevention, detection, and removal of viruses.

BACKGROUND: THE PROBLEMS OF DETECTION

Computer viruses have been around since 1986. Since then, four distinct phases have emerged, each with a different impact on anti-virus scanners:

PHASE 1: SIMPLE, STATIC VIRUSES

In 1986, there were 8 viruses, all written in what we will term 'the traditional approach'. That is to say, each virus was written with static code, resulting in every copy of the virus looking the same. The traditional approach to detecting these viruses was to search for static code with a scanner and alarm when a match was found. The key to scanning is to have a 'scan string' which identifies each virus. These scan strings can only be extracted if the anti-virus vendor has a sample of the virus; and the goal of scanning is to detect a virus which has already infected a file or a boot sector.

PHASE 2: ENCRYPTION

In 1987, virus authors began writing encrypted viruses, such as Cascade, in an attempt to defeat scanners. In Cascade, all but the first 35 bytes of the virus are encrypted, with each copy being different. This difference was accomplished by an encryption algorithm which used the original file size as the key. The actual number of different copies (except for the first few bytes) is the number of different file sizes available. Scanner developers solved such problems by scanning for those 35 static, stable bytes. This feat was manageable, and soon all scanners detected all copies of Cascade.

PHASE 3: ENCRYPTION WITH VARIABLE JUNK IN DECRYPTION ALGORITHM

In the late 1980's, virus authors came up with another method of defeating scanners: by inserting variable bytes in various places, replacing the static code reminiscent of 1986. The best of such viruses placed these variable bytes in the decryption algorithm itself. Scanner developers solved this new problem this way: define scan strings which contain markers for variable bytes, develop scanning algorithms to find some bytes, skip a variable number of bytes, and search for a match on the next bytes. For instance, to detect the fully encrypted virus Sverdlov (also known as 'Hymn of the USSR', 'USSR.1962', etc, and written in September, 1991), a scanner could use a scan string such as 'FE EB 02 ?? ?? 83 EE 08'. Using this method, scanning was slowed, but the scanners could still 'win'.

PHASE 4: POLYMORPHICS ROAM THE EARTH

In 1990, the beginning of the fourth and current phase, virus authors began writing polymorphic viruses, in which the stable bytes – the decryption algorithm – became shorter and shorter, and in which a number of different encryption algorithms were nested. Some viruses, such as Whale (August, 1990), were also able to be encrypted in memory, meaning that a traditional scan of memory might not identify them any better than a traditional scan of files. Some scanner vendors responded with large additional chunks of code in their scanner engines – code which was able to decrypt a specific polymorphic virus. Other vendors responded initially with more brute force in the scan string approach. For example, to detect the 32 morphs of Whale, Norton Anti-Virus version 1.5 provided 32 scan strings.

This approach seemed to work for a time, but then polymorphic engines came on the scene, including the Cybertech Mutation Technology, Dark Avenger's Mutation Engine (MtE), Simulated Metamorphic Encryption Generator (SMEG), Trident Polymorphic Engine (TPE), Virus Creation Laboratory (VCL), and more. Such engines take a non-polymorphic virus as input and then output the virus with polymorphic qualities. The availability of such engines has obviously made the task of writing a polymorphic virus straightforward. With these engines, virus authors no longer need to write their own polymorphic code. This cuts down on their production cycle and, as a result, the number of polymorphics has doubled about every 8 months, rather than the 8.5 month doubling time for viruses in general. Today, over 200 viruses are known to use a polymorphic engine, and another 50 polymorphic viruses exist which do not use any 'off-the-shelf' engine. Quite understandably, scanners have a difficult time detecting such polymorphic viruses.

It is difficult to estimate how many different polymorphic viruses exist:

1. Scanners have trouble detecting all copies of a single polymorphic virus, much less discriminating between two viruses which use the same engine. Thus, all viruses written with MtE are likely to be identified by a scanner as 'MtE'. If car buyers could only distinguish cars based on engine manufacturer, they would not think they had much to choose from.

2. No one seems to be able to agree on exactly what a polymorphic virus is. We could say that it was a virus which never had more than a string of *n* constant, consecutive bytes in common across copies of itself. But then we would need to define *n*. If *n* is less than a certain size, scanners will sometimes raise a false alarm. If the position of these bytes is variable, the false alarm rate goes up. If the bytes are bytes in common with many other uninfected programs (e.g., '(c) Microsoft') then false alarms are inevitable.

Some of these engines make the viruses highly polymorphic and difficult to detect using a traditional scanner. For instance, Girafe, which uses the Trident Polymorphic Engine (TPE), is able to create 16^{16} different copies of itself – about 18,446,774,000, 000,000,000 different morphs! Anti-virus vendor response was predictable: try to reverse-engineer a polymorphic virus, then try to detect its engine. However, vendor success has been limited. Some products, such as Central Point Anti-Virus seem unable to detect any polymorphic viruses whatsoever (see *Virus Bulletin*, July 1994). Few products are able to detect all copies of such polymorphic viruses as NATAS or SatanBug, and no product is able to detect all copies of Commander Bomber.

Another characteristic of the fourth phase – the large numbers of new viruses being produced – augments the strain on 'scanners' due to the sheer number of scan strings which must be produced. In the past 5 years, the total number of viruses has doubled about every 8.5 months. As of January 1, 1995, there may have been 7,198 or so different viruses[1]. If this number is correct, and the doubling rate is 8.5 months, then at the time of writing this, there would be about 16,000 different viruses. Surely this number exceeds the discrimination power of scanners.

There are many reasons for this glut of new viruses.

There is ample help for doing so in published books, journals (such as *Computer Virus Developments Quarterly*, which published source code for a 'Windows 95' virus two months before Windows 95 shipped), virus authoring software, and heavily commented source code. Virus Exchange BBSs (VXBBSs) are electronic bulletin boards, to which hackers connect in order to communicate electronically and exchange files and viruses. VXBBSs typically stock the source code for 1,000 or more viruses, along with samples of viruses which a budding author can disassemble and study. Virus authors believe in sharing their virus-writing skills, and therefore provide their source code for others to revise and/or adapt.

In contrast, not everyone can write an anti-virus product. The effort required to write a product which stops all viruses is far greater than the effort required to write a single virus which gets past some anti-virus products. There is no published help for doing so, either. Vendors strive to be profitable, so don't share source code for their products.

With perhaps 500 active virus authors in the world, and only about 50 active anti-virus vendors, the stage is set for overtaking products with problems. Over the past 4 years, viruses have emerged at a faster rate than scanner detections have improved. Where once a scanner could detect 100% of the world's viruses, today few scanners detect more than 80%, and some, such as MSAV/MWAV with DOS 6.22, may only detect about 25%. (MSAV/MWAV in DOS 6.22 is exactly the same program which shipped with DOS 6.00, and only detects 1,404 viruses – about 25% of those which some other products can detect). In relative terms, scanners have gotten worse.

BACKGROUND: THE 'PROBLEMS OF REMOVAL'

Users and organizations live by a single cardinal rule: any virus found must be removed. Although some believe that deleting the infected file and restoring the file from backups is sufficient and satisfactory, often no backups exist, and sometimes the backups are infected. Removing the virus from an infected file or boot sector is a worthy and reachable goal.

As with scanners, new virus technologies have been problematic for virus removal.

The traditional approach for dealing with removal was to 'hard code' the instructions for removal into the product: move the file pointer, copy bytes, then truncate the file. With hard-coded removal instructions, precision in both the identification and removal algorithms is critically important, for an error in either means that the anti-virus product simply damages the file or sector. Many users who have experienced such damage have stoically considered this to be one of the hazards of war; an unfortunate event.

The removal problem for vendors is complicated by the large number of viruses that are out there. Today, it is no longer adequate for a scanner to casually identify a virus based on a dozen bytes matched, then proceed to brutalize the file in which they were found, because the actual virus might be a derivative of the original which the scanner has identified – one which requires different removal steps. For instance, one single scan string can detect most members of the Jerusalem family, but various variants of Jerusalem add different numbers of bytes to the infected file.

[1] See Norman Technical Report #9, 'How Many Viruses Are There?'

Here are just a few examples:

Variant of Jerusalem	Bytes Added
Suriv 1	897
Slow	1701-1716
Jerusalem 2187	2187

Any removal instructions which treat each of these different samples in the same manner are likely to leave the user in an unfortunate condition.

Removal is further complicated when a virus is derived from two or more other viruses. Some scanners use short scan strings, with which they might detect the file or boot sector to be infected with one of the progenitors, not the derivative. Furthermore, if the scanner proceeds to attempt to remove the virus, the scanner would surely fail to remove the derivative virus and/or succeed in damaging the file.

Encryption makes for difficult removal as well. If a file virus encrypts just 1 byte of a file, then traditional removal requires that the scanner know how to decrypt that byte; for, without decryption, removal is not possible.

Anti-virus vendors have responded to the issue of removal with a variety of tactics. They have argued that only 100 or so viruses have ever been found in the wild, thus the ability to remove thousands of viruses is not important. Unfortunately, no one has done any real research on which viruses are found in the wild, and it is likely that there are thousands, rather than hundreds, which have surfaced in one or more offices in the past year.

Anti-virus vendors have also coped by suggesting that the virus at hand is a 'new variant'; that the user somehow has the misfortune of being one of the first to ever have this virus. Users are in no position to judge the truth of this statement, but the fact is that because viruses take a great deal of time to become widely distributed and common, most of these 'new variants' are actually 2 or 3 years old.

OTHER PROBLEMS WITH THE CURRENT ANTI-VIRUS PARADIGMS

We have noted that the current anti-virus paradigms have problems with both virus detection and removal, and that these problems are growing over time. There are other problems as well, leading us to the conclusion that a new paradigm is necessary.

1. **Scanning takes time.** It takes a finite amount of time to scan a machine for viruses – perhaps 5 minutes or more. If the country's 70 million employees who use PCs spend 5 minutes a day scanning, and earn $15 an hour, the annual cost of scanning (260 days a year) is $22,659,000,000. The 'costs of scanning' exceed the purchase price of anti-virus software after just a few weeks of scanning.
2. **Scanning is mis-timed.** If a machine is scanned every day at 9:00 AM, and infected one day at 9:05 AM, the virus has an entire day to spread throughout the machine and the organization. A virus can spread surprisingly far in this length of time. Even if the scanner is able to detect this virus, scanning presumes that the virus has already infected some disks or files. Traditional scanning is thus a means of detecting a problem which has already occurred, rather than a way of preventing a problem.
3. **Scanners need constant upgrading.** With 10 new viruses each day, full detection via a scanner requires daily upgrades to the product. But most organizations find upgrades a nightmare, and will not be able to upgrade this frequently. Organizations need something which does not require frequent upgrades.

4. **Scanners are slowing.** The more viruses a scanner must search for, the more places within a file it must search, and the more files it must search across, the slower the search must be. Since strings must be stored in memory, and memory is limited, we will soon see two-pass products, which load one set of strings, scan, then load a second set and scan. While vendors have used indexed searching techniques to speed their task, and computers are faster than they once were, their drives are also getting larger. Tomorrow's scanners will inevitably be slower than today's and yesterday's.

5. **Scanner maintenance is beyond the capability of vendors.** If there are only 10,000 viruses today, and the average virus requires only 1 hour to analyze, write detection instructions, and test those instructions, then creating just the detection instructions is a 10,000 hour job – about 4 person years. To compound the problem, polymorphic viruses can each require weeks of analysis by the world's most skilled programmers to determine how to detect it. It is no wonder that during the past year, many vendors have either failed or been absorbed by bigger companies. Today, the U.S. has only a few vendors with American-made products: *IBM, McAfee, Symantec/Norton, Norman Data Defense Systems* and *RG Software*. The programming strain of the traditional scanning paradigm has been one of the causes of this shrinking in the anti-virus vendor industry.

IN SEARCH OF A NEW PARADIGM

If the old scanning paradigm is falling farther behind in dealing with the virus problem, what must we look toward in a new paradigm? Here are some minimal requirements:

- ➨ The new paradigm must be able to prevent viruses from infecting boot areas and files, and prevent them from gaining control of the machine. The adage about an ounce of prevention is still true.
- ➨ The new paradigm must not require any user intervention. Background, transparent operation is critical for users who do not have the time to do a daily (or hourly!) scan.
- ➨ The new paradigm must not slow the user or the machine. Today's computers are used in business, where time is money.
- ➨ The new paradigm must be able to remove all viruses without necessarily resorting to prior knowledge of the specific nature of the virus. This is a requirement because it will be increasingly likely, in the future, that the virus in your machine is not a virus which your vendor has ever seen.
- ➨ The new paradigm must provide automatic recording of events and automatic notification of the organization's virus response team.

These requirements point to a proactive, preventive solution which includes artificial intelligence: behavior blocking (dynamic code analysis) and static code analysis.

BEHAVIOR BLOCKING (DYNAMIC CODE ANALYSIS)

If we cannot realistically expect to detect viruses with scan strings – because of their proliferation, because of the increase in polymorphics, etc – then we must find some other way of detecting them. Ideally, users should strive to buy a product which prevents viruses from infecting rather than merely detecting them after they have infected. This kind of protection is available with behavior blocking.

DEFINITION

Behavior blocking is defined as the process of dynamic code analysis. The sequence of actions of a program is monitored to determine if the actions are consistent with the behavior of viruses. Because a blocker cannot merely monitor action, but must prevent certain actions, the smart behavior blocker must employ some technique to prevent the actual results of a sequence of steps, (for instance, the behavior blocker could permit a program to execute in a 'virtual machine' until it had determined that the sequence of actions was legitimate or was virus-like; if legitimate, the actions could be echoed to the real machine. The virtual machine could include virtual CPUs, virtual drives, etc). The techniques used by one behavior blocker may differ from those used by another, but the underlying principle will be the same: a sequence of code execution will be monitored until it is determined that the sequence is safe or is harmful; if harmful, the code will not be permitted to execute and the user will be notified.

HISTORY OF THE IDEA

The idea of behavior blocking is not entirely new. Andy Hopkins was one of the first to offer a behavior blocker named Bombsqad, a TSR which would alert the user whenever a boot sector was written to, a file was written to, and so on. It simply watched the usage of DOS interrupts, and when a designated event took place, stopped the event and alerted the user, awaiting permission to continue. The problem with his approach was that it could not distinguish between a virus and a user. So the user would attempt to write to a file, and the warning would be triggered. After a number of false alarms, the user would eventually abandon the behavior blocker.

Bombsqad is no longer used. Many detractors of the technique assumed that behavior blocking had to cause false alarms, and dismissed the approach. Today, behavior blocking is not widely accepted as an approach, in part because of the inadequacy of the early demonstrations of the technique.

A number of products come with components which could be considered to be behavior blockers. For example, products from *ESaSS* and *Prescription Software* include resident **file attribute monitors**. If a virus is to infect an executable file (in a 'parasitic' way[2]), it can do so only if the file attributes permit it. If a file is marked read-only, then the virus must change this attribute to read-write before infecting. Of course, viruses have been clearing file attributes, infecting, then restoring attributes since the Jerusalem virus (1987). A resident attribute monitor can intercept the process of clearing attributes, or the process of changing the read-only attribute to read-write, or the process of changing any attribute. It can be selective or clumsy, be programmed to permit exceptions (such as changes caused by ATTRIB) or not, and can even be self-learning (let the user come up with an attribute changer of his own, a self-learning resident attribute monitor can be told to ignore these kinds of changes, and won't bother the user again when this same program is used to change other attributes).

Another approach to behavior blocking can be found in *Norton's* resident scanner, which intercepts Ctrl-Alt-Del, and checks the floppy drives for boot viruses. If it finds the floppy infected, it does not permit the warm reboot, but rather warns the user that the disk is infected. This effectively prohibits a reboot from infecting a machine with a boot virus, 'blocking' the infection.

Smart behavior blocking has been in use worldwide for several years. Developed by RE Solutions in Malaysia, and marketed in Asia by *Extol* and elsewhere by *Norman Data Defense Systems*, a device driver is able to 'see' the difference between an uninfected TSR (such as IPX or NETX) loading into memory and an infected TSR loading into memory. It is able to see the difference between the behavior of a database program which writes bytes to a database, and a virus which writes bytes to an executable. This behavior blocking device driver, called NVC.SYS, is part of 'Armour', *Norman's* virus protection package.

HOW SMART BEHAVIOR BLOCKING WORKS

A *smart* behavior blocker is able to disentangle the complex behavior of a virus from the complex behavior of a user running complex software. The basic design of such an instrument requires that viruses be very well understood by the designer, and that detailed sequences of behavior, not simple coarse behaviors, be examined and analyzed. The designer of a smart behavior blocker must use statistical analysis to determine the probabilities that particular behavior sequences are those of a virus or of a user. A coarse behavior

[2] A virus that 'infects' by creating separate code that is called before the 'infected file (as with DIR-II or a companion virus) will not be thwarted by file attributes. Such viruses are rare.

blocker might simply stop writing to a COM file, which may be an entirely valid action (e.g. some MS-DOS patches write directly to executables). A smart behavior blocker might reason as follows:

Action	**Comment**
A process opens a file of type 'COM'	Nothing wrong so far.
The process reads to the end of the file and then adds to the end of it, increasing its size.	Still nothing wrong, but suspicious.
The process returns to the beginning of the file and patches the code to point to the segment which was appended to the file.	Definitely something wrong. Like virus activity, and must be stopped, reversed, and reported.

ADVANTAGES OF BEHAVIOR BLOCKING

Behavior blocking has its advantages:

- Because the behavior blocker prevents the virus from infecting, it eliminates the need for removal from all but the original infected file.
- A behavior blocker can have a long shelf-life. Advanced algorithms can be used so that upgrades need not be done with the frequency of scanner updates. Because the behavior blocker does not scan for specific viruses but instead looks at the program's behavior, it need not be upgraded each time a new virus is discovered.

BEHAVIOR BLOCKING VS. RESIDENT SCANNING

Behavior blockers are sometimes confused with resident scanners. In fact, they are completely different technologies. Resident scanners are scanners that stay in memory, and scan a file or boot area when triggered by some event. Therefore, the problems with resident scanners are the same as those we have already enumerated for scanners. In addition, resident scanners necessarily occupy large amounts of memory, something most users cannot spare these days[3]. To cut down on memory consumption, most resident scanners do not contain any code to detect polymorphic viruses. This is not surprising – the code required would exceed the amount of memory a user is willing to allocate to such a program, and the resident scanner would slow the machine far too much. But a smart behavior blocker can detect and stop all polymorphic viruses at the time they try to go resident or infect a file. After all, a resident polymorphic virus goes into memory exactly the way a non-polymorphic virus does; a polymorphic's infection process is exactly the same as a non-polymorphic's process.

Smart behavior blocking's effectiveness is shown by *Norman* tests, in which only 12 viruses of 1,000 new 1994 viruses (less than 1.2%) got past the **1993** version of the *Norman* device driver. In comparison, the most recent versions of traditional non-resident scanners missed more than 20% of these viruses.

FUNCTIONS OF A BEHAVIOR BLOCKER

What might a behavior blocker be asked to do? Thinking of 'behavior' alone, here is a short list:

- Prevent a virus from going resident by loading low and allocating memory.
- Prevent a virus from going resident by loading to the top of conventional memory and not allocating memory.

[3] It is not necessary for a resident scanner to occupy a lot of conventional or upper memory, however. Thunderbyte's TBSCANX and its companion TBDRIVER can be loaded so as to use 0Kb of conventional memory, and only 4Kb of upper memory. The remainder of memory consumption is effectively hidden from the user, in the form of extended or expanded memory.

- ➨ Prevent a virus (or any other code) from writing to the Master Boot Sector or boot sector.
- ➨ Prevent a virus from adding its code to programs, whether resident or direct action.
- ➨ Prevent a virus from tunnelling around anti-virus or other software in an effort to gain control of the machine without detection.

If we did not split hairs, and permitted a resident behavior blocker to do other forms of anti-virus work, and wished it to be generic (working with all viruses), then we could add to this list of functions:

- ➨ Determine if a boot virus is resident at the time the behavior blocker is loaded, and optionally disable the resident code.
- ➨ Determine if a floppy disk is infected with some boot virus, and if so, clean it.

We might also expect that the behavior blocker would signal users audibly and visually (whether in DOS or Windows), would recommend the proper course of action, would record the event in a log, and would transmit relevant information to a server if the user is on a network. It happens that all of these capabilities are provided by *Norman's* NVC.SYS.

DRAWBACKS OF BEHAVIOR BLOCKING

Even smart behavior blocking may never be always smart enough. Some programs, for instance, write to themselves in a virus-like way. Installation and upgrade routines may patch existing files in a virus-like way. WINCIM and other communication programs may create temporary marker files when a batch download is requested, then dribble bytes into these existing files during a download, in a virus-like way. Programmers compiling to a file which already exists may overwrite the existing file in a virus-like way. Access control products, and products which use stealth boot virus technology to redirect the view of the Master Boot Sector (such as *MicroHouse*' EZDrive and *OnTrack*'s Disk Manager), can look like they are stealth boot viruses in memory. Odd boot sectors on a floppy can be detected by generic examiners as boot viruses, and get an unneeded cleaning.

In the end, whether a behavior blocker works well for a user, without false alarms, depends on exactly what other software they are using. In the future, the very, very smart behavior blocker will probably be able to deal with exceptions, so that certain pre-defined or user-defined events do not trigger it.

STATIC CODE ANALYSIS FOR DETECTION

Behavior blocking requires behavior, or live action. A program which loads and executes is behaving. But a behavioral analysis is not the only 'generic' way to detect a virus.

CHECKSUMMING

One common method of detecting a virus in a file is to checksum the file and store the information. Later, when the file is checksummed again, the checksummer can compare the current result with the previously stored value and warn the user of any difference.

The checksum approach has so many problems with its common implementations that it has fallen out of favor with knowledgeable users.

Problems include:

- ➨ Many checksummers ignore boot sectors and Master Boot Sectors, yet perhaps 90% of office infections are of boot sectors, rather than files.

- Checksummers which do not scan memory might be run at the same time a virus which infects on open, close, or directory scan is resident in memory. In such a case, the checksummer itself will become infected, as well as the files which it checks.
- Checksummers cannot report on why a file has changed, and therefore triggers false alarms on self-modifying programs.
- Checksummers cannot name, remove, or tell us anything about viruses.

There is a pleasant variation of checksumming which we have found useful. Since the master boot record and the boot record are small, and are rarely changed 'legally', why not back them up, then compare the backup copy with the current version; if they have changed, tell the user, and offer to repair by replacement? *Norman* has used this approach in its BootGuard (BG.EXE) and rescue disk program (RC.EXE) with considerable success. RC.EXE can also determine that CMOS has changed in some way, and can restore it.

STATIC CODE ANALYSIS

A superior alternative to checksumming is that of static code analysis, both for detection and removal. Static code analysis is defined as the process during which the code is not loaded and executed but rather studied while it is 'dormant'. Examples of static code analysis include 'heuristic scanning' and code disassembly.

Static code analysis for detection of viruses generally involves looking at the 'sum of the parts'. If the tool finds a few very good reasons to think something contains a virus, or finds a number of fairly good reasons, it can conclude that the file is infected.

Products able to do heuristic analysis of static code (i.e. a file or sector which was stored on a drive) and conclude whether or not the code contained a virus have been around for years. Some implementations have been slow, some produce false alarms, but generally the approach has merit. Products of note which offer heuristic scanning include TBScan (from the *ESaSS*), F-Prot (from *Frisk Software*), NSCAN and ViewBoot (both from *Norman*).

In an appendix, we provide samples of static code analysis. For a file virus, we provide copies of the output from *ESaSS's* TBScan and Norman's NScan. (F-Prot will, if run with the switches /analyze /only, produce the line '[filename] seems to be infected with a virus'. We did not think this enough information to provide a page in the appendix. If F-Prot is told to do heuristics, but it recognizes the virus by name, it does not provide anything but the name of the virus found.) For a boot virus, we provide copies of the output from *Norman's* ViewBoot. (Neither F-Prot nor TBScan provide static code analysis of boot sectors.) This output includes both the ViewBoot report and the ViewBoot disassembly.

Static code analysis offers these potential benefits:

- Potentially it can provide accurate information on a virus sample. If a product identifies a virus as 'Jerusalem', static code analysis can, potentially, provide more useful information about that particular sample than V-Base or other pre-published reports, for the name 'Jerusalem' is not adequately precise to determine which strain it is, and thus, exactly what it does.
- It can, potentially, determine that a suspect sample is infected, or is not infected with a virus. This can be useful if the virus is new, and not detected and named by a standard scanner.

However, static code analysis can suffer from these drawbacks:

- Static code analyzers must be carefully tuned, or they will show very high false alarm rates. Such alarms ultimately lead to distrust of the static code analyzer, and cause all manner of mischief and mayhem until the alarm is determined to be a false alarm.

- Static code analyzers typically do not name the virus, when they find one. While the static code analyzer cannot be faulted for this, it leaves us with an incomplete understanding of what we have found.
- Static code analysis can be slower than traditional scanning. Since traditional scanning is already slower than users might like, static code analysis is likely to not be used quite as often as might be appropriate.
- Static code analysis, in itself, constitutes detection, but not removal. Unless the code analyzer contains algorithms for removal, it leaves the user in the precarious position of knowing they have a problem, but not knowing what to do about it. There are, however, generic approaches to removal – approaches which work. We discuss these approaches in the next section.
- A static code analyzer must know how to decrypt an encrypted virus, or it will (falsely) conclude that the file is not infected.

STATIC CODE ANALYSIS FOR REMOVAL

In the same way we use static code analysis for detection, we can use static code analysis for removal. The goal is generic removal – removing a virus without identifying it by name and without having prior knowledge about the particular virus. Many sceptics might say that generic removal is not possible, but some companies, such as *ESaSS* and *Norman*, have been doing this with great success for years.

NON-SPECIFIC CLEANING OF FILES

The basic concept is simple. Consider the lowly COM file. On the first pass, the first 3 bytes of the COM file can be captured and stored along with some appended code which indicates the file's original size, date, and time. When a virus infects such a file, it replaces the first 3 bytes, stores them, and typically appends to the bottom of this file. On the second pass with the generic remover, the remover can see that the code it has added is no longer at the bottom. If the virus has left the code intact, then removal involves simply replacing the first 3 bytes (now a jump to the virus) with the original 3 bytes and then truncating the file to the correct size, date, and time. With an EXE file, the process involves first safely storing the header of the file, along with the size, date, time, and program entry point in a separate file. Repair can be done by replacing the header; removing prepended code, as needed, returning the entry point to its original state; then truncating to the correct length.

What if a file becomes infected with a virus which encrypts 1 or more bytes of the file? The generic remover can discern that the result of its effort is not successful, because the stored checksum mismatches the checksum which would be produced on its straightforward repair. In such a case, the remover can build a 'box' in memory, and run the virus in the box, allowing it to single-step against a 'virtual CPU'. The virus begins by decrypting itself (and the program it has infected). With each step, the generic remover studies the results to determine if the checksum is now correct. If so, it copies the decrypted file from memory to the drive, and closes the box on the virus, shutting it off long before it does any damage.

In the case of Windows files, overwriting becomes a serious problem for traditional removers. Many DOS viruses which infect Windows files assume that the file is a DOS file, and overwrite the first few thousand bytes of Windows program code, adding themselves to what they see as a DOS program with a DOS EXE header. Because of this destruction, Windows EXE files do not seem to survive infection by DOS EXE infectors. However, with generic removal, the Windows program code can be stored safely away and restored when needed. This restoration is enough to remove the virus and return the Windows program to its original condition.

NON-SPECIFIC CLEANING OF THE BOOT AREA

There are other approaches to generic removal. Boot viruses provide the simplest example. Consider a non-stealth, non-encrypting boot virus (which is to say, most virus infections). The virus typically copies the original Master Boot Record or boot record to some other sector, places its code in the sector of the code it has just copied, and ends with a jump to the displaced code. Then, when the machine boots, it reads the infected code, then the displaced unmodified original code.

Generic removal is straightforward: simply determine that the sector is infected (for instance, count the number of occurrences of calls to Interrupt 13h. Healthy Master Boot Records and healthy boot records each contain 2 occurrences; infected sectors contain more or less). Now look in all the obvious places for such a virus to have moved the original code: the slack space (side 0, cylinder 0, sectors 2 through n), the bottom of the root directory, the final cylinder, and any clusters marked bad in the FAT. Once the healthy record is located, simply copy it back to where it belongs. This is the approach taken by *Norman's* NVCLEAN, and it works with almost all boot viruses. It fails in those rare cases of viruses that overwrite the sector, such as Da'Boys.

If the virus is a stealth, encrypting virus (such as Monkey), the process can be different: use a pair of independent mechanisms to determine whether Interrupt 13h is owned by hardware or software. If owned by software, take a 'photograph' of the master boot record and boot record as the resident virus would permit. Such a picture is perfect, showing the sector as it should look. Now disable the virus by borrowing Interrupt 13 from it, and take another look. Either the Master Boot Sector or boot sector will look different, and we are in a position to repair by reversing the virus' original displacement of the sector. Thus in the case of Monkey, we can now copy the code from side 0, cylinder 0, sector 3 (the original Master Boot Record, which we saw in unencrypted form when Monkey was active) to side 0, cylinder 0, sector 1, overwriting Monkey and 'cleaning' the machine. This is the approach taken by *Norman's* NOSTELTH.

Another approach to dealing with the stealth, encrypting virus is easier on the programmer; a bit harder on the user: boot dirty (from the infected hard disk) and take a picture of the Master Boot Sector and boot sector. Copy the picture to a floppy. Now reboot clean. When drive C yields an 'invalid drive specification', simply write the code photographed to the appropriate sectors. This can be done with *Norman's* BootGuard.

THE FUTURE OF ARTIFICIAL INTELLIGENCE IN THE WAR AGAINST VIRUSES

It is a mistake to assume that nothing has been done with artificial intelligence concepts in fighting viruses. In truth, the most popular anti-virus products seem to use none of these techniques in their own design. But good anti-virus products do use artificial intelligence. Each of the techniques described in this paper has successful implementation in commercial products. For instance, *Norman's* products, which incorporate all of the new paradigm techniques described here, boast about 2 million users around the world.

But there is much to be done. For instance, users still shop for traditional scanners, having learned this paradigm back in 1989, when the paradigm had merit and the implementations of today's new paradigm left much to be desired. User acceptance is required before the vendors of traditional anti-virus products begin to look into the new intelligent paradigm.

User acceptance will always be affected by the false alarm rate. If a product triggers false alarms too often, users will reject it. That is not to say that reviewers will find fault. Typically, a reviewer aims a scanner at a directory full of viruses and garbage. A traditional scanner, which is well-written, will separate the viruses from the garbage by following the flow of program execution before reaching a decision. If virus code is in an unreachable area, the program is damaged, not infected. This means that the very best scanners can achieve detection rates of only 60% in some tests, where the number of garbage files is high. But any product which triggers false alarms under such conditions is likely to win praise from the reviewer.

APPENDIX: ESASS'S TBSCAN ANALYSIS OF A JERUSALEM STRAIN

```
Thunderbyte virus detector v6.24 - (C) Copyright 1989-1994, ESaSS B.V.

TbScan report, 02-21-1995 12:55:00

Parameters: c:\oops\*.com heuristic lo



C         \OOPS\JERUSTD.COM infected by Jerusalem related virus
c         No checksum / recovery information (Anti-Vir.Dat) available.
F         Suspicious file access. Might be able to infect a file.
M         Memory resident code. The program might stay resident in memory.
U         Undocumented interrupt/DOS call. The program might be just tricky
          but can also be a virus using a non-standard way to detect itself.

Found 1 files in 1 directories, 1 files seem to be executable.
```

1 file is infected by one or more viruses

APPENDIX: NORMAN'S NSCAN ANALYSIS OF A JERUSALEM STRAIN

```
Scanning Results.

Report prepared by NSCAN on 02-21-1995 at 12:58:12.

Contact your help desk or Norman Data Defense Systems Inc. at 703-573-8802
with questions.


There is a 100% chance that C:\OOPS\JERUSTD.COM contains a virus.

   * appears to infect when files are loaded and executed.

   * opens files, moves file pointer, reads files, writes to files, closes
     files.

   * gets, sets attributes.        * gets, resets file date.

   * adds the text ->I2V21<- to the end of files it infects.

Use of Memory:

   * allocates memory.      * resizes memory.          * goes resident.
Interrupt Usage:

   * disables, enables interrupts.

   * uses interrupts: 21h, 08h, 24h.

Symptoms:

   * deletes files.

   * displays a message or graphic.

   * effects may be date-activated.

   * checks for date of 13, Friday, 1987.

Stealth Index: 6 (above average: resets attributes, sets file date, hides in

     memory, disables error handler.)

4E00B8003DCD21725A2EA370008BD8B80242B9FFFFBAFBFFCD2172EB0505002EA31100B90500BA6B



--------------------------------- SUMMARY ---------------------------------

Switches             : C:\OOPS ANALYZE

Total files scanned  :       3

Total bytes scanned  :          3,325

Total infected files :       1

---------------------------------------------------------------------------
```

APPENDIX: NORMAN'S VIEWBOOT ANALYSIS OF A BOOT VIRUS

```
Analysis of Boot Record of A:

Prepared by ViewBoot, a product of Norman Data Defense Systems Inc.


Date: 02-21-1995

Time: 13:01:48


1.   Positive Identification

— >  This sector contains the Form.A.DS5 virus or variant! <—

— >  Norman checksum: 00081694

(The .DS means this virus is one identified by David Stang, but not yet
accurately named by other products. The name is provisional.)

2.   Analysis of code in the sector

   û Number of FATs is normal: 2

   – Contains no code to wait for keyboard input if disk is not bootable. May
     be encrypted.
   - This boot sector does not contain bootstrap code. This is abnormal. The
     virus may occupy two sectors, or may be encrypted.

   - Contains no code to display error messages in event of trouble. May be
     encrypted.

   - Contains code to write sectors. This is very virus-like! This code is
     found 3 times!

   - Contains 8 occurrences of calls to Int 13h. Healthy boot and master boot
     sectors contain 2 and only 2 occurrences of this interrupt. Boot viruses
     usually contain more, sometimes less.

   - Contains code to get the date from the real-time clock. This is like some
     viruses, and unlike healthy sectors.

   û Normal boot signature (Uª) found.

   —> Conclusion: This sector is infected with a virus. <—
```

APPENDIX: DISASSEMBLY BY NORMAN'S VIEWBOOT

```
Disassembly of Boot Record of A:
Performed on 02-21-1995 at 13:02:01
By ViewBoot, a program from Norman Data Defense Systems 703-573-8802
(c) 1994 Norman Data Defense Systems Inc.

Start Hex                 Meaning                Comment
0    EB5390               jump                   Boot Sector Jump
                          This jumps to the bootstrap routine.
3    4D53444F53352E30
                          db                     OEM name (MSDOS5.0)
                          This is the manufacturer's version of MS-DOS
                          — Start of BIOS parameter block —
11   0002                 dw                     bytes per sector
13   01                   db                     sectors per cluster
14   0100                 dw                     number of reserved sectors
                          Usually 1 (0100), unless the manufacturer has
                          reserved additional sectors.
16   02                   db                     number of FATs
                          The number of File Allocation Tables following the
                          reserved sectors. If two or more, the spares can be
                          used for data recovery.
17   E000                 dw                     number of root directory entries
                          the maximum number of entries in the root directory
                          0002 means 200 (common for large hard drives)
19   400B                 dw                     total number of sectors
                          on the drive. If 0000, then this value is provided
                          just below, with huge sectors.
21   F0                   db                     media descriptor byte
                          F0 means 1.44Mb, 2.88Mb, 1.2Mb, or other media.
22   0900                 dw                     number of sectors per FAT
24   1200                 dw                     sectors per track
26   0200                 dw                     number of heads
                          a number like 0700 means 8 heads - reverse the
                          bytes to read 0007, then remember that the first
                          head is 0, so 0007 = 8 heads
28   00000000             dd                     number of hidden sectors
                          a number like 11000000 means 11 hidden sectors
                          - the least significant byte goes first.
```

```
32   00000000           dd                    huge sectors - numer of
                        sectors if total number of sectors (above) is 0
                        - End of BIOS parameter block -
                        Otherwise, the value here is 00
38   29                 db                    extended boot signature (29h)
39   6828DF15           dd                    volume ID number
43   4E4F524E45545554202020
                        db 11 dup(?)          volume label (NORNNETUT )
                        A volume label of NO NAME is created if the drive
                        or disk is formatted without specifying a volume
                        label (with /V)
54   4641543132202020   db 8 dup (?)          (FAT12 ) file system type
                        FAT12 means a 12-bit FAT
62   FA                 cli                   Disable interrupts
63   01                 ??
64   FE                 ??
65   D6                 ??
66   8A                 ??
68   F0                 ??
69   87                 ??
70   E900F0             jmp loc
73   053B00             add ax,3B00
76   01                 ??
77   04                 ??
78   3B                 ??
80   01                 ??
81   0100               add [bx+si],ax
83   80                 ??
84   01                 ??
85   FA                 cli                   Disable interrupts
86   33C0               xor ax,ax             Zero AX register
                        AX has been set to 0.
88   8ED0               mov ss,ax
90   BCFE7B             mov sp,FE7Bh
93   FB                 sti                   Enable interrupts
94   1E                 push ds
95   56                 push si
96   52                 push dx
97   50                 push ax
```

```
98   07            pop es
99   B8C007        mov ax,C007h       ax now has the value 0
102  8ED8          mov ds,ax
104  33F6          xor si,si          Zero SI register
                   SI has been set to 0
106  26            ??
107  83            ??
108  2E            ??
109  13            ??
110  04            ??
111  02            ??
112  26A11304      mov ax,es:data
116  B106          mov cl,06h
118  D3E0          shl ax,cl          Shift w/zeros fill
120  8EC0          mov es,ax
122  33FF          xor di,di          Zero DI register
                   DI has been set to 0.
124  B9FF00        mov cx,00FFh
127  FC            cld                Clear direction
128  F3A5          rep movsw          Rep when cx >0 Mov [si] to
                                      es:[di]
130  06            push es
131  B89A00        mov ax,9A00h       ax now has the value 9
134  50            push ax
135  BBFE01        mov bx,FE01h
138  B80102        mov ax,0102h       ax now has the value 102
141  8B            ??
142  0E            push cs
143  4D            dec bp
145  8B            ??
146  16            push ss
147  4F            dec di
149  CD13          int 13h            interrupt 13h, function 02h
                   This function reads one or more sectors into memory.
                   al = #sectors to read, ch = cylinders to read,
                   dl = drive, dh = side to read.
151  72FE          jc loc             Jump if carry Set
153  CB            retf
154  0E            push cs
```

```
155  1F           pop ds
156  E82F00       call sub
159  E84100       call sub
162  BB4C00       mov bx,4C00h
165  BE4100       mov si,data
168  BF4603       mov di,data
171  E8C700       call sub
174  B404         mov ah,04h          AH has now been set to 04h
176  CD1A         int 1Ah             real time clock
                  AH has been set to 04h. When this interrupt is called,
                  function 04h gets the date, with cx returning the
                  year, and dx returning the month/day.
178  80FA18       cmp dl,18h
181  750C         jne 0C
183  BB2400       mov bx,2400h
186  BE4500       mov si,data
189  BF5D03       mov di,data
192  E8B200       call sub
195  5A           pop dx
196  5E           pop si
197  1F           pop ds
198  33C0         xor ax,ax           Zero AX register
                  AX has been set to 0.
200  50           push ax
201  B8007C       mov ax,007Ch        ax now has the value 7
204  50           push ax
205  CB           retf
206  33C0         xor ax,ax           Zero AX register
                  AX has been set to 0.
208  8EC0         mov es,ax
210  BB007C       mov bx,007Ch
213  B80102       mov ax,0102h        ax now has the value 102
216  8B ??
217  0E           push cs
218  49           dec cx
220  8B           ??
221  16           push ss
222  4B           dec bx
224  CD13         int 13h             interrupt 13h, function 02h
```

```
                    This function reads one or more sectors into memory.
                    al = #sectors to read, ch = cylinders to read,
                    dl = drive, dh = side to read.
226  C3             retn
227  0E             push cs
228  07             pop es
229  B280           mov dl,80h
231  B408           mov ah,08h           AH has now been set to 08h
233  BBF903         mov bx,F903h
236  CD13           int 13h              interrupt 13h, function 02h
                    This function reads one or more sectors into memory.
                    al = #sectors to read, ch = cylinders to read,
                    dl = drive, dh = side to read.
238  7214           jc loc               Jump if carry Set
240  B280           mov dl,80h
242  890E4900       mov cx,ds:data
246  89164B00       mov data,dx
250  B80102         mov ax,0102h ax now has the value 102
253  B90100         mov cx,0001h
256  32F6           xor dh,dh            Zero DH register
                    DH has been set to 0.
258  CD13           int 13h              interrupt 13h, function 02h
                    This function reads one or more sectors into memory.
                    al = #sectors to read, ch = cylinders to read,
                    dl = drive, dh = side to read.
260  726E           jc loc               Jump if carry Set
262  81C3           BE01 cmp word ptr [di],BE01h
266  B104           mov cl,04h 268 80 ??
269  3F             ??
270  80             ??
271  7407           je loc07
273  83             ??
274  C3             retn
275  10             ??
276  E2F6           loop local loop
278  EB5C           jmp short loc
280  8A             ??
281  77             ??
282  01             ??
```

```
283  8B                ??
284  4F                dec di
285  02                ??
286  890E5100          mov cx,ds:data
290  89165300          mov data,dx
294  B80102            mov ax,0102h          ax now has the value 102
297  BBF903            mov bx,F903h
300  CD13              int 13h               interrupt 13h, function 02h
                       This function reads one or more sectors into memory.
                       al = #sectors to read, ch = cylinders to read,
                       dl = drive, dh = side to read.
302  7244              jc loc                Jump if carry Set
304  81BF3F00          cmp word ptr [di],3F00h 308 01 ??
309  FE                ??
310  743C              je loc3C
312  817F0B00          cmp word ptr [di],0B00h
316  02                ??
317  7535              jne 35
319  B80103            mov ax,0103h          ax now has the value 103
322  8B                ??
323  0E                push cs
324  49 dec cx
326  8B                ??
327  16                push ss
328  4B                dec bx
330  CD13              int 13h               interrupt 13h, function 02h
                       This function reads one or more sectors into memory.
                       al = #sectors to read, ch = cylinders to read,
                       dl = drive, dh = side to read.
332  7226              jc loc                Jump if carry Set
334  BBFE01            mov bx,FE01h
337  49                dec cx
338  890E4D00          mov cx,ds:data
342  89164F00          mov data,dx
346  B80103            mov ax,0103h          ax now has the value 103
349  CD13              int 13h               interrupt 13h, function 02h
                       This function reads one or more sectors into memory.
                       al = #sectors to read, ch = cylinders to read,
                       dl = drive, dh = side to read.
```

```
351  7213          jc loc             Jump if carry Set
353  E82B00        call sub
356  BBF903        mov bx,F903h
359  B80103        mov ax,0103h       ax now has the value 103
362  8B            ??
363  16            push ss
364  53            push bx
366  8B            ??
367  0E            push cs
368  51            push cx
370  CD13          int 13h            interrupt 13, function 02h
                   This function reads one or more sectors into memory.
                   al = #sectors to read, ch = cylinders to read,
                   dl = drive, dh = side to read.
372  C3 retn
373  33C0          xor ax,ax          Zero AX register
                   AX has been set to 0.
375  8EC0          mov es,ax
377  26            ??
378  8B            ??
379  07            pop es
380  89            ??
381  04            ??
382  26            ??
383  8B4702        mov ax,[bi+02h]
386  89            ??
387  44            inc sp
388  02            ??
389  FA            cli                Disable interrupts
390  26            ??
391  89            ??
392  3F            ??
393  26            ??
394  8C            ??
395  4F            dec di
396  02            ??
397  FB            sti                Enable interrupts
398  C3            retn
399  BEF903        mov si,data
```

```
402  BF0300            mov di,data
405  03                ??
406  F7                ??
407  B93C00            mov cx,003Ch
410  FC                cld                     Clear direction
411  F3A4              rep movsb
413  33F6              xor si,si               Zero SI register
                       SI has been set to 0
415  BFF903            mov di,data 418 B9FF00 mov cx,00FFh
421  F3A5              rep movsw               Rep when cx >0 Mov [si] to
                                               es:[di]
423  C7                ??
424  0555AA            add ax,55AA
427  C3                retn
428  8B                ??
429  1E                push ds
430  1100              adc [bx+si],ax
432  B104              mov cl,04h 434 D3 ??
435  E3                mul bx                  dx:ax = reg * ax
436  A1                ??
437  16                push ss
439  F6                ??
440  26                ??
441  10                ??
442  0002              add [bp+si],al
444  C7                ??
445  40                inc ax
446  A3                ??
447  03                ??
449  8B                ??
450  1E                push ds
451  13                ??
453  2B                ??
454  D8                ??
455  8A                ??
456  0E                push cs
457  0D                ??
459  49                dec cx
460  D3                ??
```

```
461  EB89           jmp short loc
463  1E             push ds
464  0500C3         add ax,00C3
467  B90200         mov cx,0002h
470  BF0800         mov di,data
473  BE0F00         mov si,data
476  BBF903         mov bx,F903h
479  32F6           xor dh,dh           Zero DH register
                    DH has been set to 0.
481  B80202         mov ax,0202h        ax now has the value 202
484  9C             pushf               push flags
485  FF1E4100       call data
489  7273           jc loc              Jump if carry Set
491  F7             ??
493  FF             ??
494  0F             pop cs              Dangerous 8088 only
495  7506           jne 06
497  8108F70F       cmp word ptr [di],F70Fh
501  EB2B           jmp short loc
503  46             ??
504  47             ??
505  3B             ??
506  3E             ??
507  050073         add ax,0073
510  55AA           Normal marker for end of boot record
                    Number of unexplained bytes: 77
                    Note: this disassembly is imperfect.
                    Please do not attempt to reassemble.
```

LATE SUBMISSION

The following papers are a late addition to the proceedings and, therefore, appear out of sequence.

THE EVOLUTION OF POLYMORPHIC VIRUSES

Fridrik Skulason

Frisk Software International, PO Box 7180, 127 Reykjavik, Iceland
Tel +354 5 617273 · Fax +354 5 617274 · Email frisk@complex.is

The most interesting recent development in the area of polymorphic viruses is how limited their development actually is. This does not mean that there are no new polymorphic viruses, far from it - new ones are appearing constantly, but there is nothing 'new' about them - they are just variations on old and well-known themes.

However, looking at the evolution of polymorphic viruses alone only shows one half of the picture - it is necessary to consider the development of polymorphic virus detection as well. More complex polymorphic viruses have driven the development of more advanced detection methods, which in turn have resulted in the development of new polymorphic techniques.

Before looking at those developments that can be seen, it is perhaps proper to consider some basic issues regarding polymorphic viruses, starting with the question of why they are written.

That question is easy to answer - they are written primarily for the purpose of defeating one particular class of anti-virus product - the scanners. Considering virus scanners are the most popular type of anti-virus program, it is not surprising that they are the subject of attacks.

At this point it is worth noting that polymorphic viruses pose no special problems to a different class of anti-virus product, namely integrity checkers. This does not mean that integrity checkers should be considered superior to scanners - after all there is another class of viruses, the 'slow' viruses, which are easily detected by scanners, but which are a real problem for integrity checkers.

Fortunately, polymorphic slow viruses are not common at the moment. As a side note 'slow polymorphic' viruses also exist, and should not be confused with 'polymorphic slow' viruses. This category will be described at the end of this paper, together with some other 'nasty' tricks.

Considering how virus scanners work, a virus author can in principle attack them in two different ways - either by infecting an object the scanner does not scan, or by making the detection of the virus so difficult that the scanner, or rather the producers of the scanner may not be able to cope with it.

Polymorphic viruses attempt to make detection difficult - either too time consuming to be feasible, or beyond the technical capabilities of the anti-virus authors.

The success of virus authors depends not only on their programming skills, but also on the detection techniques used. Before describing the current techniques, however, a brief classification of polymorphic viruses is in order.

Polymorphic viruses are currently divided into three groups:

1) **Encrypted, with variable decryptors**. This is the largest and currently the most important group. Several methods to implement the variability are discussed below, but most of them should be familiar to readers of this paper.
2) **'Block-swapping' viruses**. Only a handful of viruses currently belong to this group, but they demonstrate that a polymorphic virus does not have to be encrypted. These viruses are composed of multiple blocks of code, theoretically as small as two instructions, that can be swapped around in any order, making the use of normal search strings nearly impossible.
3) **Self-modifying viruses using instruction replacement techniques**. This is where the virus may modify itself by replacing one or more instructions in itself with one or more functionally equivalent instruction when it replicates. So far this category is only a theoretical possibility, as no viruses have yet been written that use this technique. It is possible that some such viruses will appear in the future, perhaps only to written to demonstrate that it can indeed be done.

Considering that the viruses that currently fall into the second group are easy to detect using ordinary search strings, and that the third group is non-existent, the only polymorphic viruses currently of interest are encrypted ones.

For that reason the term 'polymorphic viruses' should, in the rest of this paper, really be understood to mean only viruses of the first group, that is, encrypted with variable decryptors.

So, how are those viruses detected?

Basically the detection methods fall into two classes - those that detect and identify only the decryptor and those that look 'below' the decryptor, detecting the actual virus. This is not a strict 'either-or' classification - a scanner may analyse the decryption loop to determine that it might have been generated by a particular virus, before spending time decrypting the code.

DECRYPTION-LOOP DETECTORS

There are several different methods that have been used to detect and identify decryption loops - which used to be the standard way of detecting polymorphic viruses - but there are several significant problems with these methods. The most common methods are described later, but if they are only used as the first step, and the virus then properly decrypted some of the following problems disappear:

- **Virus-specific**. Basically, the detection of one polymorphic virus does not make it any easier to detect another.
- **More likely to cause false positives**. As we get more and more polymorphic viruses, capable of producing an ever-increasing variety of decryptors, the chances of generating a false positive increase, as some innocent code may happen to look just like a possible decryptor.
- **Identification is difficult**. Many polymorphic viruses will generate similar decryptors, and it is entirely possible that a scanner will mis-identify a decryptor generated by one polymorphic virus as having been produced by another, unrelated virus. Also, in the case of variants of the same polymorphic virus, it may be possible to determine the family, but not the variant.
- **No disinfection**. Virus disinfection requires the retrieval of a few critical bytes from the original host file that are stored usually within the encrypted part of polymorphic viruses. This means that virus-specific disinfection is generally not possible, at it would require decrypting the virus.

On the positive side, detection of a particular decryptor may be quite easy to add, although that depends on the design of the scanner and the complexity of the virus. The decryption techniques are old, and several anti-virus producers have abandoned them, in favour of more advanced methods.

The most common detection methods in this group are:

- Search strings containing simple wildcards
- Search strings containing variable-length wildcards
- Multiple search strings
- Instruction usage recognition
- Statistical analysis
- Various algorithmic detection methods

SEARCH STRINGS CONTAINING SIMPLE WILDCARDS

The limitations of this method are obvious, as it can only handle a few 'not very polymorphic' viruses, which are sometimes called 'oligomorphic'. They may for example make use of a simple decryption loop, with a single variable instruction. The least variable polymorphic virus uses two different instructions, NEG and NOT, which differ by only one bit. Defeating this detection method is easy: just insert a random number of 'junk' instructions at variable places in the code. 'Junk' does not mean 'invalid', but rather any instruction that can be inserted in the decryption loop without having any effect. Typical examples include NOP, JMP $+2, MOV AX, AX and other similar 'do nothing' instructions.

SEARCH STRINGS CONTAINING VARIABLE-LENGTH WILDCARDS

This method takes care of decryptors that contain those junk instructions. However, there are two problems with this approach. Some scanners cannot use this method as their design does not allow variable-length wildcards, but that really does not matter, as the technique is very easy to defeat: just make the decryptor slightly more variable so that no single search string, even using a variable-length wildcard will match all instances of the decryptor. This can be done in several ways.

- ➠ Changing register usage: For example the DI register might be used for indexing, instead of SI, or the decryption key might be stored in BX instead of AX.
- ➠ Changing the order of instructions: If the order of instructions does not matter, they can be freely swapped around.
- ➠ Changing the encryption methods: Instead of using XOR, the virus author could just as well use ADD or SUB.

MULTIPLE SEARCH STRINGS

This is generally considered an obsolete technique, but many anti-virus producers used it back an 1990 when the Whale virus appeared. This virus could be reliably detected with a fairly large set of simple search strings. Today, however, most of them would probably use a different method. This detection method can easily be defeated by increasing the variability of the decryptor past the point where the number of search strings required becomes unreasonably large. There are other cases where the multiple search string technique has been used. One anti-virus company had access to the actual samples of a particular polymorphic virus that were to be used in a comparative product review. Rather than admitting that they were not able to detect the virus, they seem to have added a few search strings to detect those particular samples - and they did indeed score 100% in that test, although later examination revealed that they only detected 5% of the instances of the virus in question.

INSTRUCTION USAGE RECOGNITION

This method was developed to deal with Dark Avenger's Mutation engine. It basically involves assuming initially that all files are infected, then tracing through the decryptor, one instruction at a time. If an instruction is found that could not have been generated by a particular virus as a part of the decryptor, then the virus is not infected by that virus. If one reaches the end of the decryptor, still assuming that the file is infected, it is reported as such. There are two major ways to attack this technique, but the more obvious is to increase the number of possible instructions used in the decryptor. If a virus used every possible instruction in a decryptor, it simply could not be detected with this method without modifying it. The second method is more subtle, but it involves making it more difficult to determine when the end of the decryption loop has been reached.

STATISTICAL ANALYSIS

This method is generally not used, due to the unacceptably large risk of false positives. It basically involves statistical analysis of the number of certain bytes in the decryptor. It works best with viruses that generate large decryptors, that use few and uncommon 'do-nothing' instructions.

Other algorithmic detection methods are possible, and are frequently used. Sometimes they are only used to quickly eliminate the possibility of a particular file being infected with a particular virus, for example:

```
IF      The file is an EXE-structure file
AND     The initial CS:IP value equals 0000:0000
THEN    The file is not infected by Virus-X
```

In other cases the algorithm provides detection, instead of negative detection:

```
IF      The file is a COM-structure file
AND     It is at least 5623 bytes long
AND     It starts with a JMP FAR to a location at least 1623 bytes from the end of the file
AND     The first 10 instructions contain at least five instructions from the following set
        {AAD,NOP,CLI,CLD,STC}
AND     Within the first 100 bytes from the entry point there is an XOR [SI/DI/BX],AX instruction
AND     Within the first 200 bytes from the entry point there is a branch instruction that transfers
        control back to the XOR instruction described above
THEN    The file is infected with Virus-Y
```

It should be obvious from this example that the rules can get complex, perhaps unreasonably complex, and obviously require significant work to implement. Also, in some instances it is just not possible to get a sufficient number of rules like this to ensure accurate detection, not even considering the rules the virus itself may use to determine if a file has already been infected as the number of false positives would be too high.

At this point it is very important to bear in mind that, while false positives are a very serious problem for the anti-virus author, they do not matter at all to the virus author. A false positive just means that the virus will not infect one particular file it might otherwise have infected...so what - after all, it has plenty of other files to infect.

Having looked at the detectors that only detect the encryption loop, we must look at the more advanced detectors, which detect the actual virus, instead of just the encryption loop.

Compared to the decryptor-detecting methods, the following differences are obvious:

- **More generic**. These methods require significantly more initial work, but the extra effort required to add detection of a new polymorphic virus is far less than with some of the other methods described above.
- **Less chances of false positives**. Having decrypted the virus, it should be possible to reduce the chances of false positives almost down to zero, as the entire virus body should be available.
- **Identification is easy**. When the virus has been decrypted, identification is no more difficult than in the case of non-encrypted viruses.
- **Easy disinfection**. The same applies to disinfection - it should not be any more difficult than if the virus had not been encrypted to begin with.

There are two such techniques which have been used to detect polymorphic viruses.

'X-ray'

Generic decryption

The X-raying technique was probably only used in two products, both of which have mostly abandoned it by now. It basically involved assuming that a particular block of code had been encrypted with an unknown algorithm, and then deriving the encryption method and the encryption keys from a comparison between the original and encrypted code.

As this sounds somewhat complicated, an example is in order:

Assume that manual decryption of one virus sample reveals that a particular block of code should contain the following byte sequence:

```
B8 63 25 B9 88 01 CD 21
```

The corresponding encrypted block of code in a different sample looks like this:

```
18 C4 8B 0C 34 C2 07 F0
```

Is there any way this sequence could have been obtained from the first one by applying one or two primitive, reversible operations like for example:

XOR with a constant
ADD/SUB a constant
ROL/ROR a fixed number of bytes

Yes, because XORing the two sequences together generates the sequence:

```
A0 A7 AE B5 BC C3 CA D1
```

Calculating the differences between the bytes in that sequence gives the following result:

```
07 07 07 07 07 07 07
```

which shows that the original sequence, and (presumably) the entire virus body can be obtained by XORing each byte with a key, and then adding the constant value 7 to that key, before applying it to the next byte.

Using this method, it may be possible to deduce the operation of the decryptor, without looking at it at all. There is a variant of the X-ray method which has been developed by Eugene Kaspersky, which works in a different way, but produces the same result.

The reason 'X-raying' has mostly been abandoned is that it can easily be defeated, for example by using an operation the X-ray procedure may not be able to handle, by using three or more operations on each decrypted byte or by using multiple layers of encryption.

The last method to be developed does not suffer from that limitation, and can handle decryptors of almost any complexity. It basically involves using the decryptor of the virus to decrypt the virus body, either by emulating it, or by single-stepping through it in a controlled way so the virus does not gain control of the execution.

Unfortunately, there are several problems with this method:

- Which processor should be emulated? It is perfectly possible to write a virus that only works properly on one particular processor, such as a Cyrix 486 SLC, but the decryptor will just generate garbage if executed on any other processor. An intelligent emulator may be able to deal with this, but not the 'single-stepping' method.
- Single-stepping is dangerous - what if the virus author is able to exploit some obscure loophole, which allows the virus to gain control. In this case, just scanning an infected file would result in the virus activating, spreading and possibly causing damage, which is totally unacceptable. It should be noted that a very similar situation has actually happened once - however the details will not be discussed here.
- Emulation is slow - if the user has to wait a long time while the scanner emulates harmless programs, the scanner will probably be disabled, and obviously a scanner that is not used will not find any viruses.
- If the virus decryptor goes into an infinite loop and hangs when run, the generic decryptor might do so too. This should not happen, but one product has (or used to have) this problem.
- How does the generic decryptor determine when to stop decrypting code, and not waste unacceptable amount of time attempting to decrypt normal, innocent programs?
- What if the decryptor includes code intended to determine if it is being emulated or run normally, such as a polymorphic timing loop, and only encrypts itself if it is able to determine that it is running normally?
- What if the decryptor is damaged, so that the virus does not execute normally? A scanner that only attempted to detect the decryptor might be able to do so, but a more advanced scanner that attempts to exploit the decryptor will not find anything. This is for example the case with one of the SMEG viruses - it will occasionally generate corrupted samples. They will not spread further, but should a scanner be expected to find them or not?

Finally, it should be noted that there are other ways to make polymorphic viruses difficult than just attacking the various detection techniques as described above.

'Slow polymorphic' viruses are one such method. They are polymorphic, but all samples generated on the same machine at the same time will seem to have the same decryptor. This may mislead an anti-virus producer into attempting to detect the virus with a single search string, as if it was just a simple encrypted but not polymorphic virus.

However, virus samples generated on a different machine, or on a different day of the week, or even under a different phase of the moon will have different decryptors, revealing that the virus is indeed polymorphic.

Another recent phenomena has been the development of more 'normal-looking' polymorphic code. Placing a large number of 'do-nothing' instructions in the decryptor may be the easiest way to make the code look random, but it also makes it look really suspicious to an 'intelligent' scanner, and worthy of detailed study. If the code looks 'normal', for example by using harmless-looking 'get dos-version number' function calls, it becomes more difficult to find.

So, where does this leave us? Currently anti-virus producers are able to keep up with the virus developers, but unfortunately the best methods available have certain problems - the one most obvious to users is that scanners are becoming slower. There is no indication that this will get any better, but on the other hand there are no signs that virus authors will be able to come up with new polymorphic techniques which require the development of a new generation of detectors.

MACRO VIRUSES - THE SUM OF ALL PH33RS?

Richard Ford

National Computer Security Association, 10 South Courthouse Ave, Carlisle, PA 17013, USA

Tel +1 717 258 1816 · Fax +1 717 243 8642 · Email rford@ncsa.com

The discovery of the first macro virus in the wild has precipitated a flurry of articles over the last few weeks. The coverage has ranged in depth and topic, from claims that 'Revolutionary Multi-Platform Virus Attacks Word Users World-Wide' [1] to the rather more muted coverage given by *Microsoft*, which claimed that 'a prank macro is a harmless but annoying macro.' [2]

Which of these viewpoints (if either) is correct, and just what are the threats posed by WordMacro.Concept, and macro viruses in general?

A BRIEF HISTORY OF MACRO VIRUSES

The possibility of a macro virus is nothing new [3]. Many virus researchers have discussed the subject, and several papers addressing the issue have been published - an overview is presented in [4]. However, we now have not one, but three, macro viruses, all of which have appeared in reasonably quick succession. The idea (and, unfortunately, the actual code) is now in the public domain, and it seems likely that we will see a spate of variants in the next few months.

For the sake of brevity, we will concentrate on the *Word for Windows* environment for the remainder of this paper, but readers should note that the techniques described are applicable to many different applications. Before taking a look into the crystal ball and trying to predict what developments we may expect, let us examine three macro viruses: WordMacro.Concept, WordMacro.DMV and WordMacro.Nuclear.

WORDMACRO.CONCEPT

The operation of a macro virus is, in essence, identical to that of the more familiar file infecting virus: when an infected macro is run, the virus replicates, creating macros which contain copies of the virus code within them. Whereas a file infector requires that the infected file be run, a macro infector requires that an infected macro be run. Unfortunately, the functionality provided by many applications allows macros to be executed automatically, frequently leaving the user completely unaware that anything untoward has occurred. In the case of WordMacro.Concept, the virus utilises a 'feature' of *Microsoft Word*: if a file is opened which contains a macro named 'AutoOpen', the contents of this macro are automatically executed. Users have come to expect features like this, and often demand this type of functionality in their mail readers, PGP front ends, etc. However, it is precisely this feature which allows the virus to operate.

In the case of WordMacro.Concept, this AutoOpen macro performs a brief check to make certain that the system is not already infected. If the virus is not already installed, it creates several new macros on the system, named AAAZAO, AAAZFS, FileSaveAs and PayLoad. Finally, the virus displays a message box, containing what appears to have been intended to be an infection counter.

The PayLoad macro is never called and contains only the text

```
Sub MAIN
        REM That's enough to prove my point
End Sub
```

making it obvious that the virus author fully recognised the potential for future viruses. The macros AAAZAO and AAAZFS are simply used for storing copies of the AutoOpen and FileSaveAs macros which are added to infected files.

The bulk of the work done by the virus is carried out by the FileSaveAs macro. This macro uses a very powerful feature of the *Microsoft WordBASIC* programming language: the ability to 'enhance' standard features of the Word environment (in this case, the File 'Save As' menu option) by linking them to a new macro routine. This happens transparently, and the user is not aware that any skulduggery has taken place.

When any file is saved using the File 'Save As' menu option, control is passed to the virus' own FileSaveAs macro. This displays the standard File Save As dialogue box, with one exception: the user is only given the opportunity to select a destination drive and directory using the point and click GUI if the file is not already infected.

Once a filename and location has been selected, the virus creates four new macros in the target document: AutoOpen (which is simply a copy of the AAAZAO macro), AAAZAO, AAAZFS and PayLoad. Infection is now complete and the new document is saved. It should be noted that in fact although the file is saved with the extension .DOC, it is now more correctly titled a 'Document Template', although this distinction is lost on the average user.

This virus does deliberately reveal its presence when it infects a system, but this routine is trivial to remove. Note that we have only taken a very brief look at the properties of this virus; a more complete examination may be found in [5].

WORDMACRO.DMV

The document in which the WordMacro.DMV virus was distributed states that it was written by Joel McNamara. The paper claims to address the inherent risks of applications which allowed for the automatic loading and execution of embedded macros. The virus code contains the following justification and description of McNamara's actions:

```
REM    This demonstrates an application-specific document virus
REM    generated by and automatic macro in Microsoft Word 6.0.
REM    Code is executed each time a document is closed. This
REM    macro is only a demonstration, and does not perform any
REM    destructive actions.

REM    The purpose of this code is to reveal a significant security
REM    risk in software that supports macro languages with
REM    auto-loading capabilities. Current virus detection tools are
REM    currently not capable of detecting this type of virus, and
REM    most users are blissfully unaware that threats can come from
REM    documents.
```

It should be noted, however, that the claim that current tools are incapable of detecting this type of virus is untrue.

Unlike WordMacro.Concept, WordMacro.DMV utilises another 'auto-executing' macro within the *Word* environment: AutoClose, which is executed whenever a document is closed. In order to understand exactly how DMV functions, let us consider the actions which take place when a DMV-infected file is opened.

Whenever a file is opened within *Word*, *Word* checks the file format. If it is a document template file, it searches for macros contained within it and adds them to the local environment. In the case of DMV, an infected document contains a macro which is automatically executed whenever a document is shut.

The AutoClose macro first checks all the global macros, to make certain that it is not already installed. If this condition is not met, the AutoClose macro is copied to the global document template. The macro then checks to see if the document which is currently being closed is infected. This is carried out by searching for the existence of a macro within that particular document. If one is found, the document is assumed to be infected, and the routine aborts. If no macros are present, the virus copies the AutoClose macro from the global document template into the current document, and closes it. Thus, the new document (once again, now more correctly labelled a document template) is infected, and is capable of spreading the virus to other machines.

WORDMACRO.NUCLEAR

The most complex *Word* macro virus to date, of which I am aware, is named WordMacro.Nuclear. This virus comes complete with ten different macros: AutoExec, AutoOpen, ToolsMacro, FileSaveAs, FilePrint, FilePrintDefault, InsertPayload, PayLoad, DropSuriv and FileExit.

Rather than present a complete analysis of the virus, I will simply give an overview of its main features.

The overall operation of the virus is very similar to that of both WordMacro.Concept and WordMacro.DMV. However, the author of Nuclear has taken the development significantly further. The most immediate difference is that the individual macros which make up the virus code cannot be viewed, as they have been saved as 'Execute only'. Although this restriction can be easily circumvented, it represents at least some attempt by the author to hide his handywork.

Secondly, the virus contains a number of different side effects.

- If the date is 5th April, the virus attempts to overwrite the files IO.SYS, MSDOS.SYS and COMMAND.COM, rendering the DOS partition unbootable. Fortunately, due to a bug in the code, this routine fails.
- If a user attempts to print a file and the seconds value of the time is greater than 55, the lines

  ```
  And finally I would like to say:
  STOP ALL FRENCH NUCLEAR TESTING IN THE PACIFIC!
  ```

  are added to the end of the document.
- If the time is 5pm, and an uninfected file is opened, the virus attempts to launch DEBUG, and run a binary infected with the Ph33r virus. Once again, this routine is buggy and fails.

Although the techniques used within Nuclear are not entirely successful, the ideas are there. The possibility of dropping a 'traditional' file virus or rendering the fixed disk unbootable just by double-clicking on a 'data' file is very close to being realised.

PROBLEMS PROBLEMS

The main reason why WordMacro.Concept caused such a big stir was that it is capable of infecting files which were not considered executable in the normal sense of the word. However, as applications have become more complex, the concept of executable must also be extended to objects which we would normally consider to be 'data', such as spreadsheets, and word processor documents. This is a separate threat from that posed by object linking and embedding; this increasingly blurred distinction between data and executable is explored in [6].

The existence of macro viruses in the wild causes a number of problems for both users and anti-virus product developers. Some of the more obvious risks are outlined below:

- A macro virus will not necessarily be in a filename with a known extension. Whereas a typical file infector will usually only infect files with the extension COM of EXE, a macro virus can infect any file, provided it is in the correct format for the environment in which it is running. For example, MS-WORD does not care about the file extensions used by files; if one wishes to save weekly reports with the extension .RPT, this is perfectly acceptable. Thus, macro viruses may well infect files according to their use, not their extension. Moreover, these will tend to be files which are typically thought of as data, not executables.
- Macro viruses are very easy to write. In the case of the two *Word* viruses currently known, the entire source code of the virus is visible within all infected files. This means that very little programming knowledge is required to modify the virus. The author is already aware of WWW sites which are offering samples of the virus for download.
- As applications become better integrated, there is the possibility of macro viruses which can spread between different applications. Even if this is not realised, readers should note that the virus theoretically may be able to spread within specific applications, regardless of the *underlying* operating system. In the case of WordMacro.Concept, any operating system which is capable of running *Word 6.0* is susceptible to the virus.
- Virus non-specific anti-virus products are currently unable to detect macro viruses. The infected object is one which by its nature we expect to change, and so a simple behaviour blocking or checksumming technique is not viable. It is possible to alter such programs to handle these viruses, but such efforts may not be practical. In the case of a checksummer, it is relatively easy to only checksum those parts of a file which are concerned with macros. However, the checksum would alter every time any macro within the file was altered. Additionally, new documents created would have to be assumed to be clean. Those developing entirely hardware-based products are likely to have an even more difficult time, as the actions of the virus are identical to certain functions which a legitimate program might undertake.
- Although it is possible to scan all areas of all files, this would greatly increase scan times. By making the scanner aware of the *Word* document template file structure, the necessity to scan an entire file would be decreased. Furthermore, a scanner manufacturer may opt to scan only those files with .DOC or .DOT extensions, and ask the user to only scan all files if a virus is discovered. One concern with a scanner-based solution is the ease with which the virus can be altered - whether this becomes a genuine problem or not remains to be seen.
- It is increasingly common to use an Email package which is capable of sending binary files over the Internet without any special actions on the part of the user. Such a system provides the perfect environment for a macro virus like WordMacro.Concept. While a user may think twice before running an 'executable' file sent to him as a file attachment, the expansion and loading of an infected document file will often pass completely unnoticed. Additionally, what could be more natural than to send a document via Email? It is likely that until the user community gains a solid understanding of the actual risks posed by macro viruses, few will take the necessary precautions.

- Scanning for macro viruses in several different 'data' file formats requires time and research on the part of the anti-virus software vendor. While scanner size and complexity will hopefully not suffer too badly, the effects on TSR virus protection are more difficult to predict. In this case, memory overhead and speed of execution are paramount. There are likely to be additional problems for TSR behaviour blockers.
- It is possible to launch other executables simply by opening a data file. If DEBUG is present on the system, it is trivial to assemble a program and execute it. Thus, Trojans, logic bombs and viruses which are not confined to a particular application's environment may be dropped onto a system.
- Macro viruses are intimately involved with the 'data' file to which they are attached. By using macro commands it is extremely easy to include damaging routines which change the data stored on the system in subtle ways (e.g. WordMacro.Nuclear's plea against nuclear testing).

SOLUTIONS

In the preceding section, we have outlined some of the more immediate problems posed by macro viruses, while trying to avoid seeding the virus writing community with new ideas. Suffice it to say that most macro languages are extremely powerful, and simply by adding a suitable amount of imagination, it is possible to carry out many varied attacks using this technique. It should also be noted that these attacks are in no way limited to *Microsoft Word*: many applications are vulnerable to attack in the same way.

There is some debate as to the long-term solution to the major virus threat. While some vendors seem to feel that a scanner-based solution is adequate, it seems likely that improvements can be made within the environment of the effected application. Although it will never be possible to completely eliminate the possibility of writing a macro virus, it is possible to make the task a little more difficult.

Perhaps the largest contribution to the virulence of WordMacro.Concept is the fact that the AutoOpen macro is run by default, transparently to the user. Similarly, the feature of WordMacro.DMV which makes it a potential threat is that the file can add a new macro to the global environment simply by being loaded and closed. Although it is possible to prevent the execution of both AutoOpen and AutoClose, if the document is loaded/closed with the Shift key pressed, this is not a practical solution. While it is conceivable that a user may remember when opening a document, few would be likely to be suspicious when exiting one.

One possible reduction in the threat is to prevent any macro from autoexecuting, without first warning the user. However, although this puts control firmly into the hands of the user, it is still not an entirely acceptable fix as it requires continual user interaction whenever a document which contains macros is encountered; interaction which the average user is not technically competent to provide.

Another possibility would be for *Word* to warn a user that a file is a document template, and not a document. Once again, however, this puts the onus for protection on the user, who has to make the decision whether to open the document or not.

Although several fixes rapidly appeared for the WordMacro. Concept virus, we are, as yet, devoid of a long term solution. Adding macros to the *Word* environment which prevent installation of this particular macro virus is all well and good, but will provide protection from just this one variant. Even during the development of this paper, another *Word* macro virus has been discovered; a custom-built fix for each is not an attractive solution.

There may be some silver lining to the cloud. According to *Microsoft*, it is currently finishing work on a macro-based solution which will alert the user of possible macro virus attacks. This shield is based on currently available *WordBASIC* functionality, and it attempts to detect suspicious files before they can infect the system. The user will then have the choice of opening the file with or without its embedded macros.

Microsoft plans to post this updated solution on many on-line services, including its WWW site, http://www.microsoft.com.

CONCLUSIONS

There seems to be little doubt that macro viruses are here to stay - whether or not we like it. Although this paper has concentrated on the *MS Word WordBASIC* environment, it is possible to write similar macro viruses for many other applications. Unless developers keep a careful eye on the functionality they build into products, this situation is likely to get worse not better.

There is currently no obvious long-term solution to the problem. In the interim, a scanner-based technique will suffice, but the ease with which macro viruses can be edited is a definite cause of concern. An integrity checking approach is likely to be fraught with difficulties, though proponents of such products should not feel too gloomy: devices which rely on a purely hardware based approach may have encountered a threat type which is impractical (although not necessarily impossible) to defend against.

REFERENCES

[1] *S&S International*, Press Release (1995)

[2] Microsoft Product Support Services Applicaiton Note WD1215 (1995)

[3] Harold Highland, 'A Macro Virus', *Computers & Security 8*, pp. 178-188 (1989)

[4] Vesselin Bontchev, 'Future Trends in Virus Writing', *Proc. 4th Int. Virus Bulletin Conference*, pp. 65-82 (1994)

[5] Sarah Gordon, 'What a (Winword.) Concept', *Virus Bulletin*, September 1995, pp. 8-9

[6] James Beckett, 'When is Not a Program a Program?', *Virus Bulletin*, July 1993, pp. 9-11

UK GOVERNMENT CERTIFICATION OF ANTI-VIRUS SOFTWARE

Chris Baxter

CCTA, Room 2024, Empress State Building, Lillie Road, London, SW1, UK
Tel +44 171 824 4101

INTRODUCTION

There are said to be difficulties with evaluations of anti-virus products.

It is widely felt amongst industry figures that in many cases they are not being done sufficiently well to inform users in an effective manner. And the problems are not minor, or due to a surfeit of 'cowboy' evaluations done on the cheap, although this has often been the case in the past. When we see experts of the calibre of Vesselin Bontchev complaining that the best evaluations he can manage are not good enough we know that the problems are not superficial, but fundamental.

If the experts are talking like this, what hope have the end users of these products? Can we do to anything to improve the situation, and if so, what?

I have been working with the UK Government on aspects of evaluation applied to anti-virus products, and I am going to talk about some of the history and principles of evaluation in this rather esoteric area, what government work in both the US and Europe has brought to the field, and what could be done to improve the situation. Though this is intended to be a general talk, and I will not be describing detailed test protocols which I would recommend, I will be trying to indicate general areas where I believe current evaluations are having difficulty, and how we are hoping to alleviate this.

I will also be talking at a general level because I believe that the world of a/v evaluations has something to teach those who evaluate other security products, and that the particular difficulties which are faced in our community are also faced, to a lesser extent, by those in other computer security fields.

EARLY EVALUATION HISTORY

The problem of malicious software started in 1986, with the BRAIN and VIENNA viruses, plunging the world into a new dark age, or giving security professionals something extra to threaten us with, depending on your point of view. From the start the world faced this new threat with an almost complete lack of information. You may recall Peter Norton's famous rejection of the problem as an 'urban myth, like alligators in the sewers'. This field started as an arcane one, filled with forbidden secret knowledge, and the rapid rate of change we have experienced since then has ensured that it remained largely a dark continent to most people.

In the earliest days of a/v work many researchers kept their collections and the knowledge they gained from studying them closely guarded secrets. It is understandable in hindsight. The press were warning that viruses would bring down the Western world with its reliance on technology, and early academic studies indicated the potential for rapid world-wide spread. It was surely right to pass as little information as possible out to the world at large, since any data could be used to improve viruses. And if an a/v product was being sold it made no sense to release data to competitors, did it?.

The first 'product evaluations' under these conditions tended to be discussions on bulletin boards, typically inside the academic community. Looking back on such early discussions, it seems that the 'best' product was usually designated as the one which could detect the 'latest' virus. Product vendors would sometimes respond with complaints (particularly if they also detected that virus but under a different name), and the stage was set for the numbers game, which so many people played between 1989 and 1993.

You will all remember that period. The best product was the one which detected the most viruses. It was so easy - anyone could weigh up the relative merits of two products simply by noting the numbers on the front of the packaging. It was just as easy for the a/v producers, especially when it was realised that what was being measured was not what you could do, but what you claimed you could do. And we are only talking about scanners here. Salesmen for products which were based on checksummers or behaviour analysis would regularly claim that not only could they detect all known viruses, but all those as yet unwritten, which put them in a commanding lead, on the grounds that infinity is a greater number than any which can conveniently be written on the front of a 3 1/2" disk.

The non-specialist PC magazine evaluations of that period tended to be functional descriptions, with the simple results of a run against a collection noted, sometimes prominently, and sometimes even almost ignored, as if the colour of the installation screen was a more important feature than the number of viruses found. With some of the evaluations that could almost be true, since the collections which were used for testing tended to come from one product manufacturer, who not surprisingly detected all of them. When such magazines attempted to run independent tests without expertise from the product manufacturers, as on one famous occasion in the UK when a magazine used a collection of 'viruses' altered so that they would not replicate, the findings degenerated into farce.

MORE CONSIDERED OPINIONS

A number of academic institutions began to show an interest in studying this field, offering the prospect of truly independent expertise, and possible doctorates. After all, Frederick Cohen can claim the rather dubious title of 'father of the virus problem' as a result of his seminal paper in this field. Unfortunately the early results of academic research were rather mixed - the application of medical epidemiology techniques to the problem led some early researchers to conclude that all computers in the world would soon be infected by viruses - data which the press were eager to publish, but which brought no fame to the researchers. The best work, by universities such as Karlsruhr, has been invaluable in keeping an independent voice of comment on a/v products alive but the work required has been very heavy, and dependent on individual skilled researchers remaining with the university. These have tended to move out or be 'poached' by the a/v industry as it expanded during this period, and since there remains a real shortage of skills in this field it will obviously be increasingly difficult for a good researcher to retain an independent voice against increasing financial temptations from the industry .

Through all this a few specialist a/v magazines have maintained an island of comparative reliability. They ran tests to their own protocols, against their own collections, which were generally reliable. They commissioned individual expert appreciations of particular products, and more recently have begun to offer specialist tests on individual types of viruses or aspects of protection such as repair or heuristic scanning. They have remained the best source for end users to obtain useful information about a/v products.

Yet there are still problems with these evaluations, admittedly both independent and expert, for the end-user. A typical test report is best characterised as a race with the tester holding the prize. We all believe that these race meetings are held fairly, and we applaud the principle of publishing the protocols so that those who wish to can investigate them, but what do the findings mean? Is a small percentage difference in a table significant, or is a large one not particularly worrying? It is hard to know what is acceptable, as opposed to which is best and whether it matters that another, on the day, is (just) second best.

And of course the other problem is that the race meetings keep happening. A winner on one track comes third or fourth a month later in a different race, even one which you have reason to believe is run as fairly as possible by reputable officials. Perhaps that is what is meant when some of the most respected names claim that, no matter how hard they try, even the best run competitions are just not good enough. Are there any other approaches to evaluation which would give us a warmer feeling that we can rely on the findings staying reasonably constant, and maybe even letting us understand how they apply to real life? During this period of time one of my responsibilities was the provision of advice to government departments on the purchase of a/v products, and I certainly wished that this were so.

GOVERNMENT COMPUTER SECURITY EVALUATION TECHNIQUES

THE ORANGE BOOK

While industry has been struggling with the issue of how to recognise a good anti-virus product since 1986, government bodies around the world responsible for purchasing secure computer system, usually for the military, have been trying to understand what constituted a good secure system for a somewhat longer period. It might be thought, and I think that it is the case, that some of this work could be applied to the problem of commercial evaluations of a/v products. Both the scholia of knowledge and experience are aimed at a similar problem, and both complement each other. For while the a/v community have developed their evaluation approaches from experience, much government work in this field has been designed from theory.

The first example of government evaluation work is, of course, the US Army and MITRE studies which culminated in the 'Orange Book' the Trusted Computer Security Evaluation Criteria. This document was the first attempt in the world to lay down guidelines on what constituted 'security' on a computer system, and so, as the first metric, set the language and thinking for all the work that came later.

Of course, the Orange Book did not address virus problems directly. It was really quite restrictive, confining itself to the security problems of the military in the 1970s, which were almost exclusively those of confidentiality on operating systems. Viruses may cause confidentiality problems, but are essentially a system integrity issue, confused by the fact that they are an active threat. Most system integrity threats are passive - a line goes down, a component breaks. But viruses are active, like the majority of confidentiality threats.

The major contribution of the Orange Book was to propose that security on a computer system could be seen as comprising two primary components, Functionality and Assurance. Any system had these components, and an evaluation consisted of defining a required functionality and then testing it with a specified level of rigour to give the required assurance that the functionality existed, and was properly implemented.

The Orange Book comprised, indeed comprises, because it is still the basic US Government Computer Security Evaluation manual, a set of ascending levels of functionality for an operating system. As the levels rise extra functionality is added on, and more stringent testing is required. Continuing the image of an athletics competition, if a/v industry evaluations are like races, this is closer to a high jump. The levels and rules are well specified, and any product can be entered at a particular height.

As it stands, the Orange Book is not much use for a/v product evaluation, since it contains no specific a/v functionality. Because of this, no a/v product has been evaluated using Orange Book techniques.

ITSEC

In response to the Orange Book work, several European countries began developing their own criteria for security evaluations, and within a few years these were brought together as a single method for security evaluation which could be recognised throughout Europe. This was the IT Security Evaluation Criteria (ITSEC).

With the benefit of hindsight it could be seen that restricting the testing regime to operating systems alone was too constraining, and ITSEC allows any functionality to be specified and tested, to increasing assurance levels similar to the Orange Book.

This increased flexibility had the disadvantage of making the evaluation approach more complex. ITSEC does not consist of a series of levels against which a restricted set of products can be matched, but rather a process, a series of stages starting with the definition of a functionality, and progressing through a series of tests to achieve assurance.

The major contribution of ITSEC was to recognise that, to gain the flexibility of being able to specify any function and test for it, two new concepts had to be introduced. We understand that a product may have Functionality and Assurance, but if we want to allow any functionality, we need to be sure that it is the right functionality. So the ITSEC testing regime specifies that, when the Assurance testing is done, specific stages of testing must identify the Correctness of the functions (whether they work as specified), and the Effectiveness of the functions (whether what they do is useful).

Testing for the Effectiveness of a product only makes sense if we know what it is that it has to do effectively. So the ITSEC process requires a statement to be made at the beginning of the evaluation describing the threat which the product is intended to deal with, and the manner in which it does so. This statement, together with other related items such as the environment in which the product is meant to work, is known as the Security Target, and an Orange Book evaluation is similar to an ITSEC evaluation with a government specified Security Target.

If the Orange Book process can be thought of as describing a high jump competition, the ITSEC process is a little more general. Think of it as describing the rules for a jumping competition - high, long, triple, standing, indeed any kind of jump which can be imagined.

With the ITSEC process we have the first indication that a formal government-sponsored evaluation method could be of use in the field of a/v product examination. An a/v product could be offered for evaluation with an appropriate Security Target, and be given a grading according to the level of assurance that it could achieve. What advantages might this approach offer over the more accepted techniques I have described above?

ITSEC ADVANTAGES

Government evaluation techniques differ from typical Industry techniques in that considerable emphasis is placed on the theory of what is being done, and on adherence to a set of rules. In the case of ITSEC evaluations, an approach is taken which relies less on testing that a given function (say, detecting FORM) exists, and more on proving that it must exist by study of the design and implementation of the code. This is certainly a slower and more cumbersome approach than simply testing whether the detection capability works, but it has the advantage of being a more fundamental approach -there is less room for errors such as testing products against files which do not contain viruses. Such testing is not banned, it is just that assurance is best seen as coming from an examination of the product design against a specification of the threat. Of course, it is crucially important that the original requirement - the threat - is right.

Another difference which will be noted is this emphasis on tracing the function and code of the product from the original requirement. The concept is less obvious in both a typical Industry evaluation, where the evaluating expert decides what will be tested for, and the Orange Book type of evaluation where a set of functions is predetermined, and was originally derived from theoretical work. This is an area in which the ITSEC approach might offer advantages to a typical magazine evaluation; instead of setting up a single test which has to be fair to two different products with two different approaches each product could be required to justify its claims and approach in the light of the threat. In practice this would be too costly and labour-intensive to consider for tight magazine budgets and deadlines, but the principle offers one of the few available methods of being fair to each product and still testing them in a justifiable way.

So have we produced the answer? Is the a/v industry beating a path to our door? Sadly, the answer is no, or at least, not yet. In fact, we have had some experience of a/v evaluations using the earlier UK methodology that pre-dated ITSEC, but was very similar to it in concept and were sufficiently unhappy with the results to suspend evaluating this type of product until we felt that we could do a good job.

ITSEC DISADVANTAGES

Our experiences are worth examining, since I believe that they can shed some light on my earlier question - why it is that even the best industry a/v evaluators are unhappy with their results.

The first is that an ITSEC evaluation costs money, a fair amount of money. Has this made ITSEC ineffective as a general purpose security evaluation process? Though cost must always be a factor I think the answer is no. Although an ITSEC certificate is not yet considered an essential accessory for all security products, there is a continual and growing stream of applicants requesting evaluations.

Is it that an ITSEC evaluation takes a considerable time? This is certainly a problem for products in the a/v arena, where new releases happen every month. Any reasonable examination of a product must take some time, but the approach taken by the ITSEC methodology of examining the fundamental design of the product takes longer. However, we know that major test studies with large collections take a lot of time to set up too, and this is accepted by the industry. It is true that any evaluation system intended for a/v products needs the ability to react quickly to new releases of products, and ITSEC could not do this at the time. But then, neither could any other approach.

If the ITSEC evaluation approach can offer a more reliable method of checking functionality than testing alone, and can offer a more fair evaluation of products using different approaches to solving the same problem, why were we as unhappy with our evaluation technique as some of the less-skilled magazines were in the early days of industry evaluations? In a word, it has to do with the threat.

An ITSEC evaluation depends crucially on a statement of the threat at the outset. From this a justification of what the product does is developed, and this is then shown to be effective and correct in the later stages of the process. And the statement of the threat that the product is expected to defend against is obtained from the product producer.

From one point of view this makes sense. A security product manufacturer may decide to defend against any kind of threat he wishes, and it is assumed that if his clients wish to defend against this threat also, they will buy his wares. If they do not, the product will not sell. But this analysis does not cater for the situation where few or no users have any idea of what the threat really consists of, and even the product producers are not too sure.

In retrospect what is likely to happen is obvious. The product manufacturers simply claimed that the threat they had to match was the one which their product defended against. And not surprisingly, they passed the evaluation. The problem we had was similar to that of the simplistic early magazine evaluations, in that each manufacturer would simply present their collection of viruses and claim that that was the threat, which, of course, for them it was.

IMPROVING ITSEC AND OTHER EVALUATION PROCESSES.

How could we solve this problem? Well, we had to have an unambiguous and clear definition of the threat. So we tried to find out more about it, and instead, I believe, stumbled on the problem which lies at the heart of all current a/v product evaluations, done by the industry, independent magazines, academic institutions or governments. There is no clear, widely agreed and accepted definition of the threat from malicious software.

There are certainly people who know all that is available to be known about the problem. There are institutions which house the largest collections of virus-related data, and even viruses, in the world. But there are few people, and I have not found any, who are able to quantify the real-world threat from different viruses, explain how effective each type is at passing from machine to machine in reality rather than theory, and generally provide all the data which the ITSEC methodology requires if it is to successfully evaluate a/v products against the actual threat.

I believe that this is the problem which is causing similar difficulty to the best independent researchers in the world today. You can certainly maintain a clean (ish) and up-to-date (ish) collection, though this is a major achievement which few institutions in the world have managed in practice. You can establish a test protocol which is fair to all products (though it is probably an impossibility to establish a protocol which is admitted by all the product manufacturers to be fair!) But when you have done these things, which are very hard to manage, and run a test, how do you distinguish between two products scoring 97% and 98%? How do you distinguish between two products scoring similar marks but detecting different viruses? If you decide to run a special test, such as detection of a particular sub-type of virus, how do you relate this to the performance of the product as a whole?

These questions are currently difficult to answer, and the subject of debate between researchers. An evaluator can certainly present findings as a set of figures and offer an opinion, but an end-user reading them cannot be sure how they relate to the virus problem in the world in general, let alone his computer in particular.

This problem has hit the formal evaluation processes beloved by governments harder that the independent evaluations undertaken by industry experts. After all, no problem is ever understood fully, and by definition the best industry experts are doing the best evaluations that can be done in the world. They can make estimations and offer opinions which are worth something even in the absence of proof. Those of us who are working with more formal evaluation techniques, however, fall at the first hurdle if the base data we need to work from is absent. I think, however, that if the problems we have found can be addressed, then all those concerned with the a/v world will benefit - the manufacturers, independent experts and users - all, in fact, except the virus writers.

You will not be surprised, therefore, to learn that we are trying to tackle the problem of finding enough information about the threat from viruses to recover our capability to evaluate a/v products using the ITSEC method. We have been working in this area for the last two years, and have been collecting some of the data we feel that we will need for over a year now, so we are now close to being able to 'open for business' again.

It may be of interest to know what sort of data we are collecting. We have several processes operating to gather data and convert it into a suitable format for use in an evaluation. The earliest one we have put into action - it has been running for a year and a half now - is the collection of data on virus incidents. We need to know which viruses are common, which are rare but still out there, what damage they cause, and several other items of information associated with an infection.

This information is collected in the UK from a combination of sources. Government departments, the police and the virus industry all contribute, merging their data into a common format, and we have a database of some 3000 incidents which is now big enough to examine for trend information. We operate this in association with the German computer security authorities, who also contribute, and I hope shortly to be able to increase our collection area to take in the USA, Canada and other countries.

As well as virus incident data we will need access to large virus collections. We have found that the most reputable collections are, in general, either happy to make their collections available to a body with our aims, or will agree to run tests to our protocols once we have agreed them. However, as I indicated before, the emphasis on ITSEC evaluations is less on practical tests and more on the description and justification of processes in the product which are intended to defend against the threat. This does not preclude the running of tests in a similar manner to other evaluation protocols, but does require us to examine a product and understand why it is performing the actions it is.

This requirement has led us to develop another less usual aid to evaluations, a Virus Encyclopedia. For the use of ITSEC evaluators, this document aims at describing all known virus techniques, with particular reference to how the technique attempts to circumvent detection, and the implications for testing products which aim to deal with it. Armed with this data, an ITSEC evaluator can examine the design of a/v products for known flaws which viruses may take advantage of, and check that all known virus attack techniques are able to be defended against.

Finally, we have a coverall subject heading, that of 'Best Practice'. Items are considered for inclusion here by a UK Industry-Government liaison group, who make a ruling on whether a particular technique - selling a/v products only on write-protected disks, for instance - should be a requirement for any product to pass evaluation.

All these items, put together, constitute the Threat Statement, a description of the entire body of malicious code which we would expect an a/v product to deal with, including descriptions of what would constitute 'dealing with', and indications of how this could be measured.

It will immediately be understood that the Threat Statement is dynamic. We have not commissioned single items of work, but instead have set up processes which are intended to continue to function indefinitely, and which can pass out the data we require on the virus threat and associated subjects as we require. This dynamic nature of the threat brings a second problem in its wake - if we can manage to evaluate a product by 'freezing' the threat at one moment in time and requiring the manufacturer to justify his product design against it, what do we do when the next issue of the product comes out?

A second evaluation would be out of the question. The ITSEC process is never going to be as fast as a conventional evaluation, and it is not free. The organisation who want the product evaluated, usually the manufacturer but sometimes a large customer, have to pay an Evaluation Facility to undertake the actual evaluation work. While we may hope that one evaluation per year may be commissioned, we are unlikely to get one per month!

We have approached this problem in a novel manner, and one which would, perhaps, be less available to an independent researcher or magazine evaluating several products in tandem. Having checked that the product is acceptable, we will proceed to examine the manufacturer to determine whether the company is capable of keeping the product up to scratch.

This operation will resemble a quality inspection in some ways; it will examine the processes inside the company which are needed to maintain the product, and confirm that they are in place and working. Then, if everything is acceptable, the company can be issued with its ITSEC certificate valid for a year. It will retain this depending on a satisfactory outcome from the periodic audit visits which will occur during this time.

These processes for formally measuring and documenting the threat are unlike anything we have worked with before, but they seem to be called for by the unusually dynamic nature of the virus threat. We believe that they will work - our data already seems to be following the trends in the virus world closely, and we have hopes that it will prove useful in a wider variety of circumstances than for virus product evaluations alone.

CONCLUSION

I have indicated the crucial importance that formal evaluations, and ITSEC in particular, attach to overtly describing the threat. Of course, all security evaluations must be done with some appreciation of the threat in mind, but in fields where this threat moves slowly there is usually a general consensus amongst most workers in the field.

I believe that many of the current concerns of evaluators in this field, of all types, springs from the fact that there is too little agreement on what constitutes an adequate anti-virus product, because there is too little known and quantified about the threat. This knowledge is lacking on both sides of the counter - the product manufacturers may be more informed that the users of the character of the threat, but they would still find it hard to quantify.

We have begun the processes needed to pin this data down, to enable all those involved in this subject to talk about the threat in the same language, and provide a platform for differing conclusions to be discussed, and agreement reached. This process seems to be operating well over the last year in the UK, but viruses are, of course, a world problem, and world solutions are needed to cope with them.

So we hope over the next few months to encourage a similar process of monitoring the virus threat using a standard format to begin in other countries round the world. If this can be achieved we will have a repository of data which may be used in different ways by different evaluation techniques, but which should still produce similar findings, since it will be derived from a single source, mutually agreed by the virus researchers around the world. If we can manage this, then the work we are doing will have a useful purpose over and above that for which it was originally designed, and possibly one which will enable users of all computer security products in the future to have a greater belief in the effectiveness of their chosen products.