## 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.

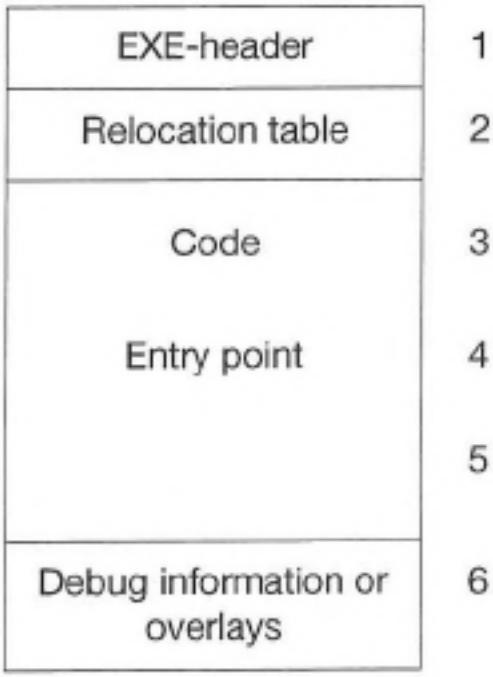



*Fig. 1*

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 affect performance. 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 [Ser]:

**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.

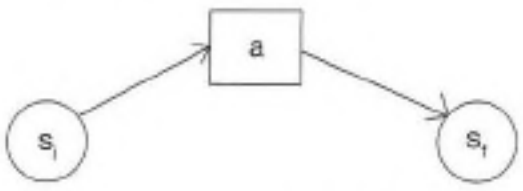



*Figure 1: State transition diagram*

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:

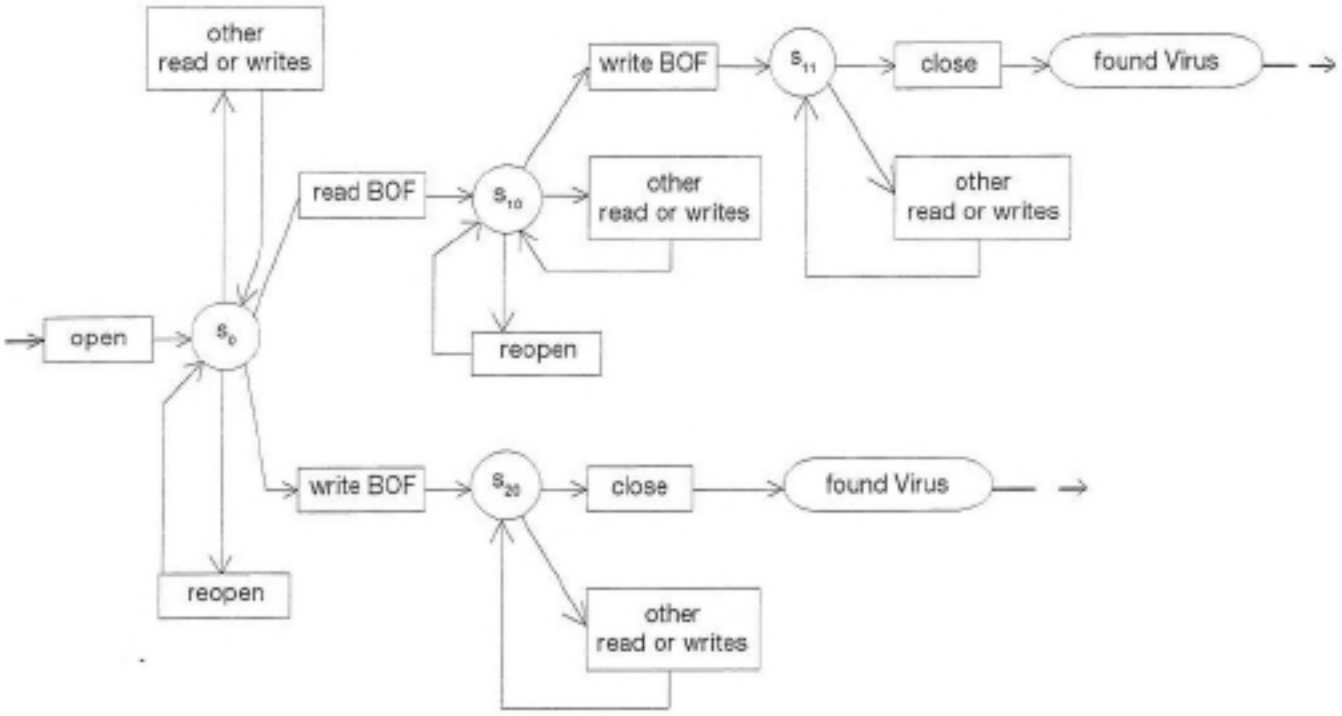



*Figure 2: Generic rule for identifying COM file infectors*

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.

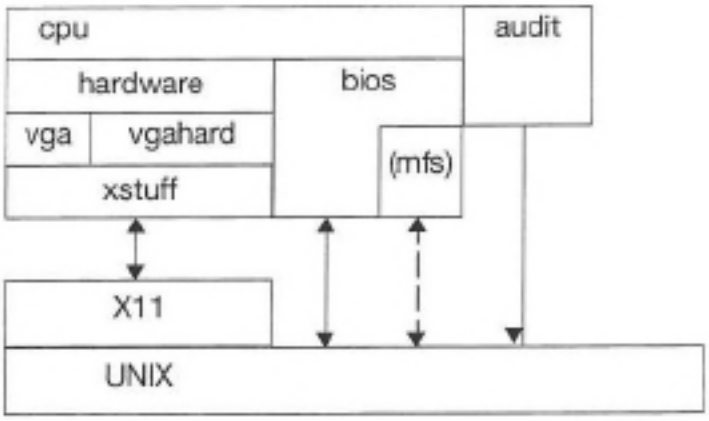



*Figure 4: Modules in Pandora*

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.

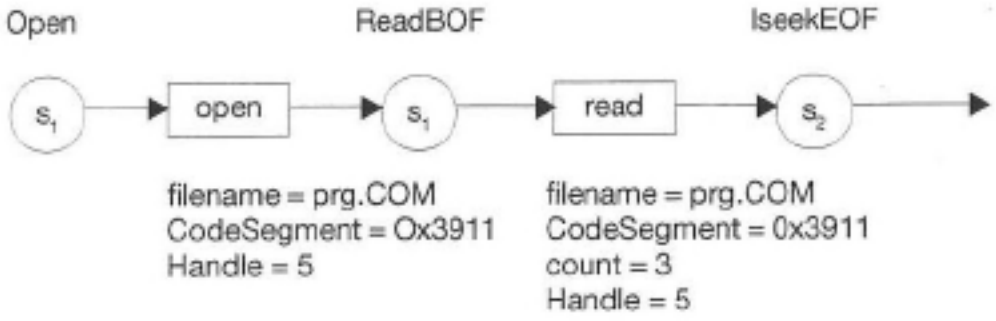



*Figure 5: Example of a state transition diagram*

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.

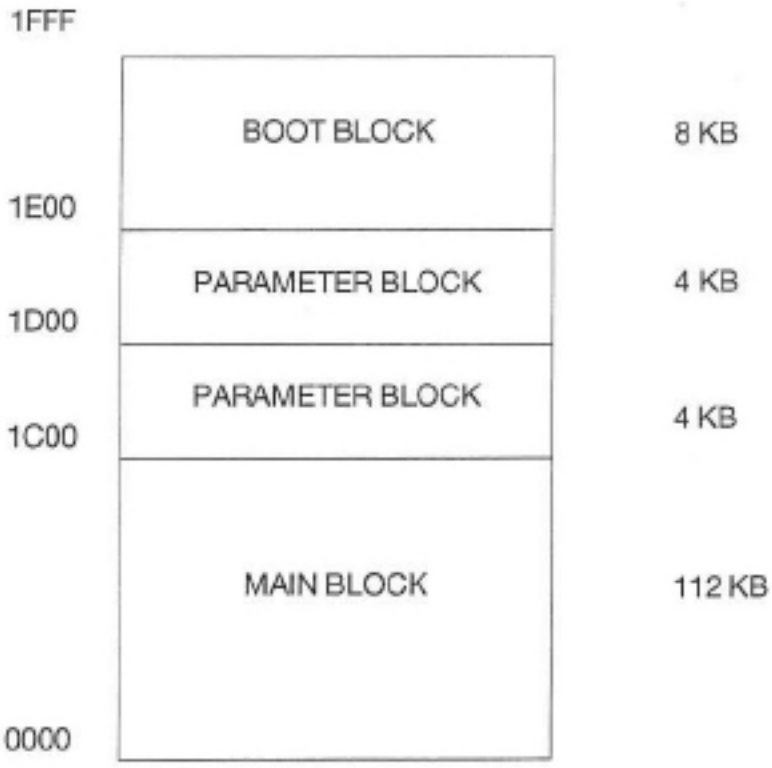



*Figure 1. Memory Map of Intel 28F001BX*

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).

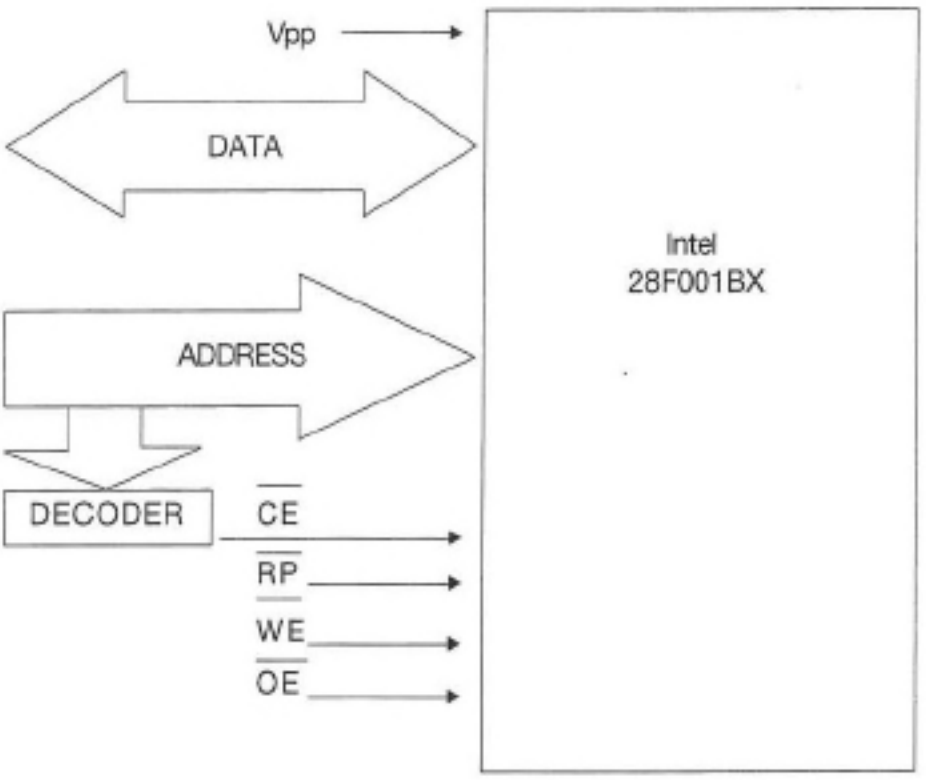



*Figure 2. System - Flash Memory Interface*

### 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

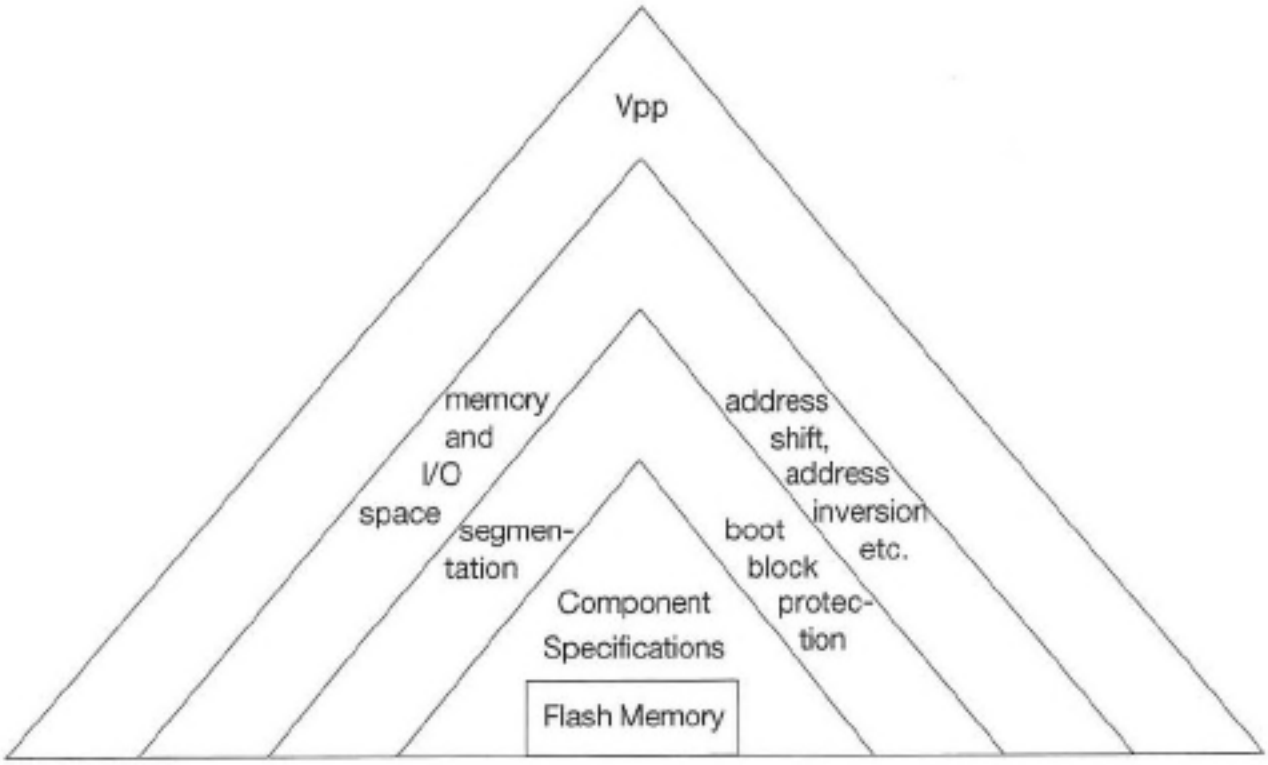



*Figure 3. Erase/Program Protection Scheme*

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).

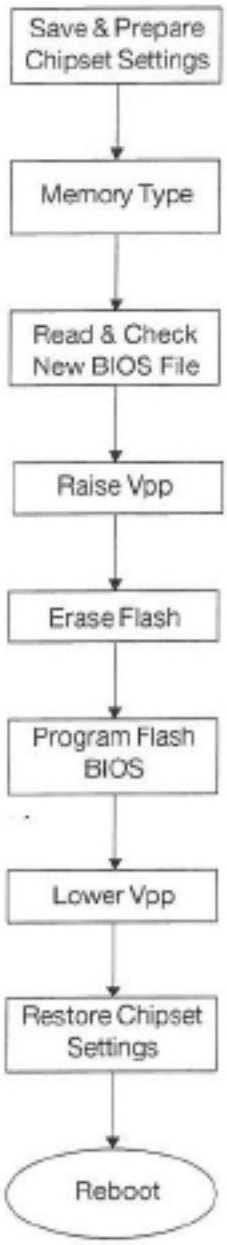



*Figure 4.*
*Flash BIOS Update*

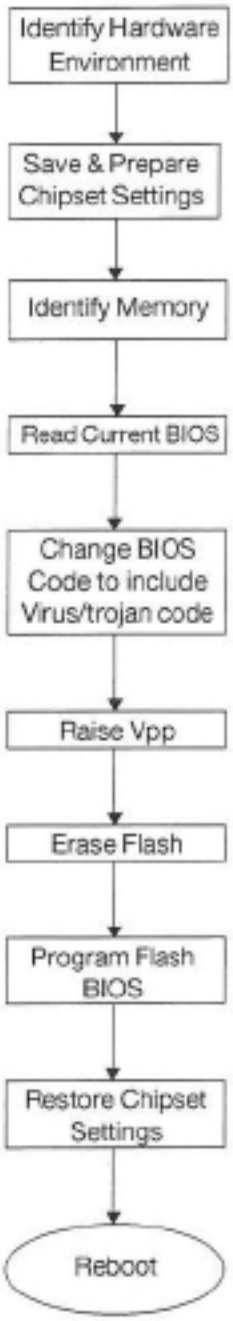



*Figure 5.*
*Flash BIOS Infection*

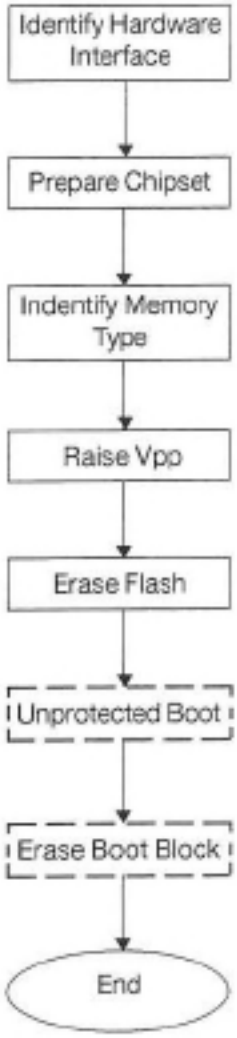



*Figure 6.*
*Flash BIOS Destruction*

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.

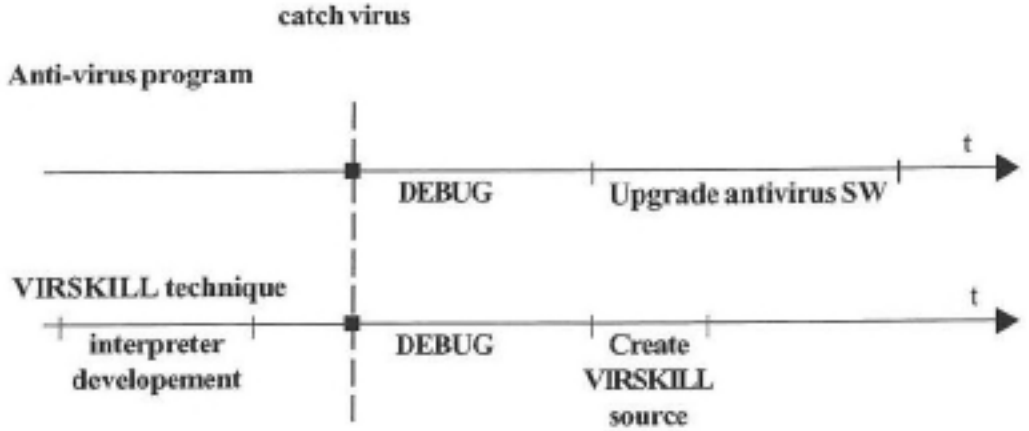



*Fig. 1: Upgrading periods I.*

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.

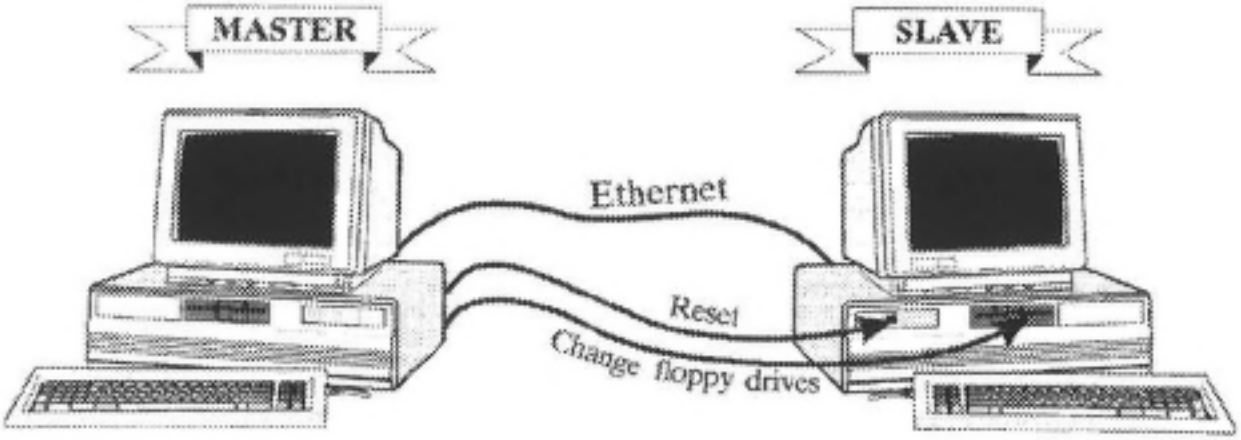



*Fig.2: The structure of AVAS*

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.

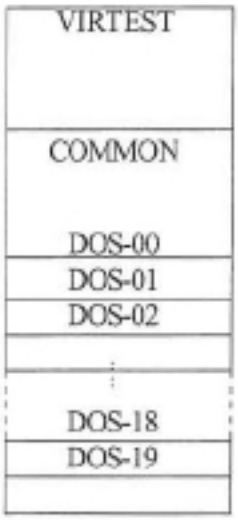
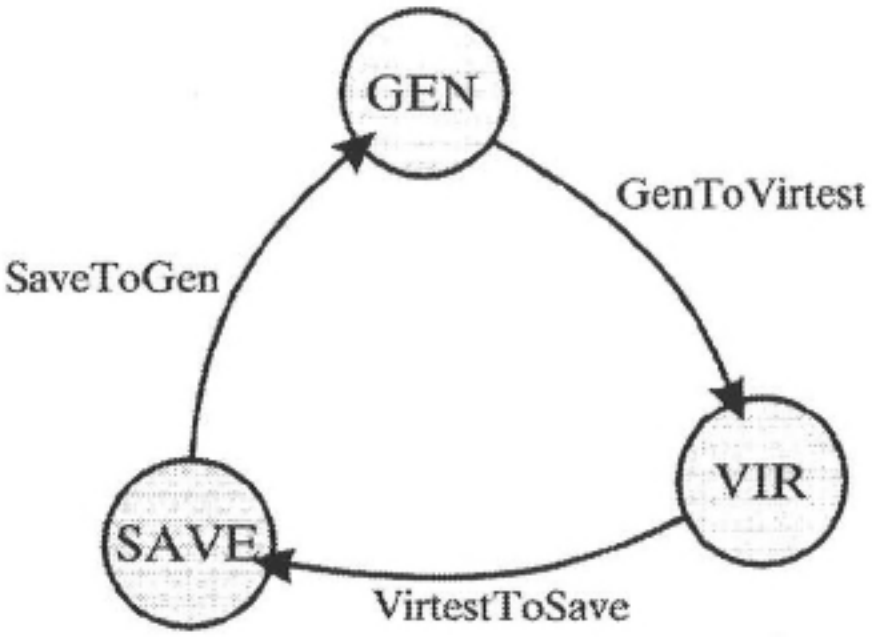



*Fig. 3: The structure of the disk in SLAVE*

During the operation of the SLAVE, three principal states are possible as follows (Fig. 4):

- virus reproduction state (**VIR**)
- data saving state (**SAVE**)
- regeneration 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 | | | | |
|---|---|---|---|---|---|---|---|
| VIR | SAVE | GEN | Name | size | Start Cyl | End 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 |
| - | - | | 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 |
| - | - | C: | 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.

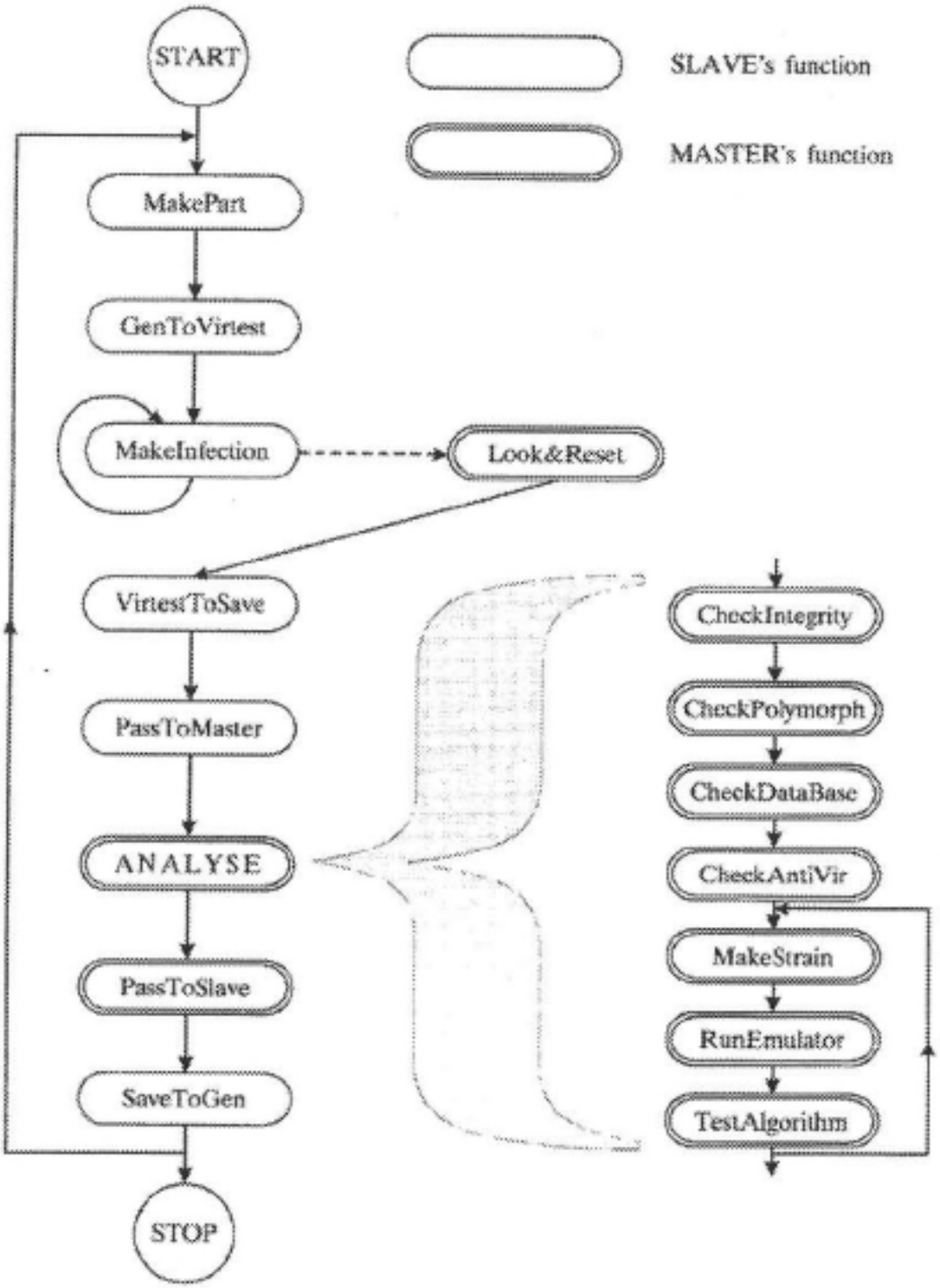



*Fig.6: Working Cycles*

## 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).

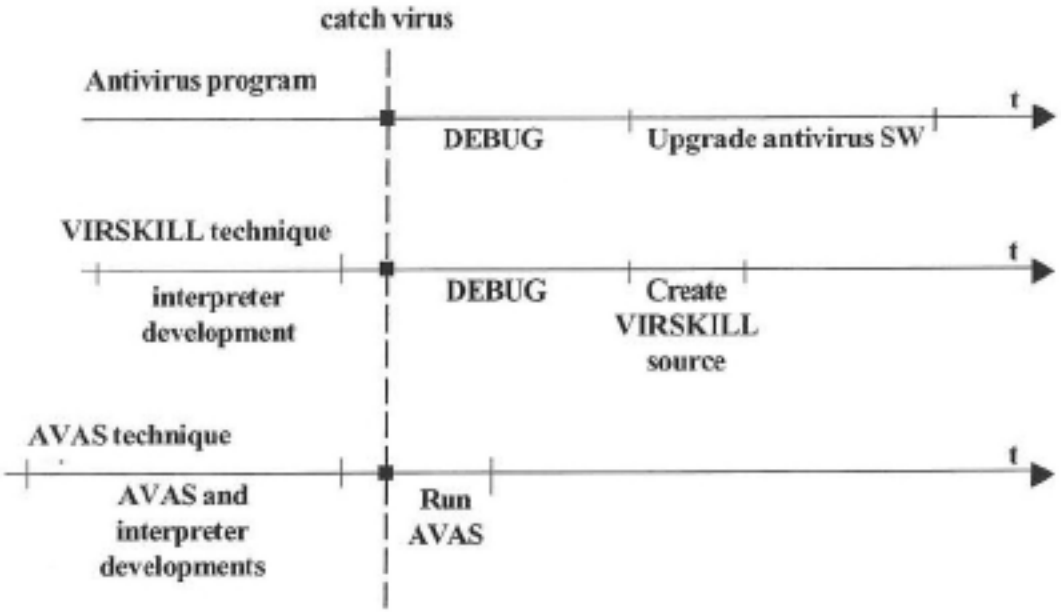



*Fig. 7: Upgrading periods II*

## 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 / BUFFERS | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|
| 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)<br>(recognized only by FLOCK.COM, ignored by GOAT and GOATS) |
| /H, /? | Help screen |
| /Inn | use fill byte 'nn' instead of standard zero-fill<br>(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<br>(stack segment is filled with 'STACK' by default) |
| /Mnnnn | place code in the middle of the file exactly at nnnn<br>position ('nnnn' is 32-bit value, but see limitations below) |
| /N[nnnn] | fill goat file with pseudorandom bytes. The parameter<br>(if given) is a random number generator seed.<br>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:

$$2 < \text{Size\_of\_COM} < 65280$$

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.

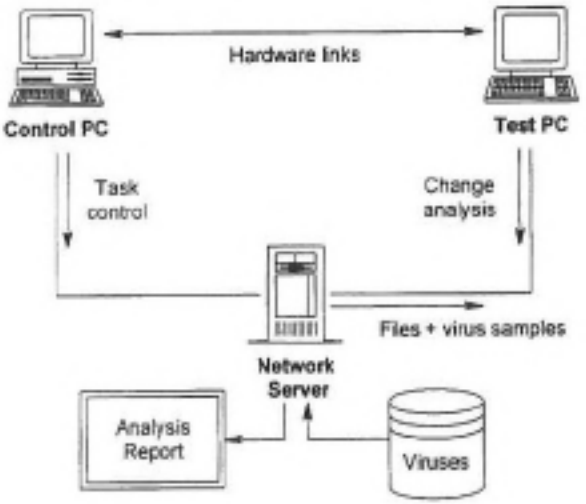



*Fig. 1 ATOMS component links*

## 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 paper is a late addition to the proceedings and, therefore, appears 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?

DAY 2

000188

# 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 I 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.