November 1991 ISSN 0956-9979

VIRUS BULLETIN

THE AUTHORITATIVE INTERNATIONAL PUBLICATION
ON COMPUTER VIRUS PREVENTION,
RECOGNITION AND REMOVAL

CONTENTS

EDITORIAL

A Little Knowledge is a Dangerous Thing

'*Give a child a hammer and all of a sudden things around him will start looking like nails.*' (Anon.)

No MIS or DP manager in his right mind would hand out *The Norton Utilities* to all PC users regardless of their need or ability to use such powerful and potentially destructive software. The misuse of *Norton* (or *PC Tools* or any other powerful disk editor) could cause untold damage if its distribution were not limited to technically competent staff.

Neither would it be wise to instruct general users as to the existence or nature of certain of the more dangerous DOS commands - most PC users in business run a limited set of applications (text editors, spreadsheets, databases and DTP being prevalent) and can remain happily oblivious to such two-edged swords as *FORMAT* and *FDISK*. Even the humble delete command dons a perplexing mantle when combined with those cheeky wildcard characters '*.*'! Obviously, all of the DOS commands are thoroughly outlined in the user manual - be it from *Microsoft* or *IBM* or *Digital Research*. It is fortuitous in this instance, therefore, that people rarely, if ever, refer to manuals while using software - this is one of the major reasons why software developers employ 'walk-thru' menus. The information and resources to cause untold accidental damage are readily available within DOS itself but are not clearly signposted as such.

The point here is that handing out powerful security and audit tools to the masses, and providing superficial education about the operating system to people who don't need that information, is a recipe for disaster. Instructions to use security tools and techniques should be on a 'need to know' basis - PC Support should know the exact locations of all such software and its distribution should be extremely limited.

In the case of anti-virus software, many packages aim to provide comprehensive virus detection and removal facilities. 'Toolkit' utilities of this sort provide programs to replace boot sectors, edit specific areas of disk and even write protect drives. Is it wise to place such power in the hands of users? Even a simple scanner becomes a complex beast when you examine the number of menu driven or command line options that many developers have provided. Can PC Support be sure that users will comply with their *ex-cathedra* statement to boot from a write-protected system floppy disk? Do users know *what* such a disk is, or *why* they should use it, or *how* to prepare it? Do they even know the difference between drive A: and drive C:?

It is sometimes difficult for technicians to appreciate the relative ignorance of non-technicians - many developers of security software still believe that mass populations can be trained to use their products correctly and effectively. More seasoned and forward-thinking observers knew a long time ago that this belief was nothing more than a pipe dream. Given that it takes a massive and concerted effort to educate users in the most *basic* aspects of corporate security, training a mass community in the use of relatively complex software security packages is a wholly untenable objective.

Security managers in many organisations rightly conclude that they are simply not prepared to trust end-users with *any* degree of technical responsibility when it comes to combating computer viruses. As a result, the tools and techniques for this job are restricted to those capable of using them. This is an *elitist* rather than *populist* approach - considering the wealth of accumulated ignorance in any society it is entirely understandable and well conceived. This is not to belittle education; if a problem can be explained in simple, straightforward and readily understood terms then that is all to the good. However, there is an enormous divide between understanding the basic tenets of a problem and proficiency in dealing with it!

The strategy which is most effective and which has been widely adopted combines three elements: *central reporting*; *specialist response teams* (variously known as PC SWAT or CERT, or simply PC Support) and *risk analysis*. *Central reporting* effectively channels all enquiries and problems to the SWAT team - qualified technicians who have studied the virus problem and who are supplied with the requisite information and tools to deal with any outbreak both swiftly and correctly. *Risk analysis* is the process of identifying the 'mission critical' machines within the organisation. These are the machines from which any loss of availability or integrity would have a serious impact on the overall performance of the organisation. A risk analysis of any organisation usually shows that 'mission critical' PCs comprise a relatively small percentage of overall IT resources. It is vital, however, that these PCs are adequately protected and this *may* necessitate the purchase of suitable defensive software.

One questionable strategy is that of equipping *every* single PC with defensive software. This may be necessary in high security environments but imposes burdens in terms of financial outlay, support costs and inconvenience. Memory-resident software may well cause memory clashes (with *Windows 3.0* or SHARE.COM for example), false alarms and is critically dependent on *user-compliance*. Checksumming software needs careful implementation and management - particularly in an a shifting software environment. Memory-resident software is prone to subversion by stealth viruses as is checksumming software if it is run in an infected DOS environment. Monitors and checksummers can be administered - but not on 5,000 PCs in 60 or 70 locations!

Ultimately, put the diagnostic tools in *capable hands* and purchase defensive software on the basis of considered *risk analysis*. Finally, make friends with *Symantec* and the *Federation Against Software Theft*...locate all unauthorised copies of *The Norton Utilities* and delete them!

TECHNICAL NOTES

Norton and The Cascade Message

In recent months *VB* and various anti-virus software manufacturers have received a number of calls concerning reports of 'Cascade' by the *System Information* (SI) program supplied with *Norton Utilities* version 5.0. These spurious reports have nothing to do with the Cascade virus and do not imply infection by any virus.

The confusing message is displayed by SI in its list of hardware interrupts, as a description of IRQ2 and refers to the internal hardware of the PC.

PC-ATs contain two 8259A hardware interrupt controller chips, each of which support eight system interrupts. The 8086 family of microprocessors has a single line to communicate with the interrupt controllers, so IRQ2 on the first interrupt controller accepts interrupt requests from the second controller. Thus the two controller chips are connected in a cascading manner - hence the 'Cascade' message.

The actual Cascade virus has been the cause of various misinterpretations; the virus is also known as 1701 which also happens to be a standard disk read error message! Perhaps Hailstorm, Fall, Autumn Leaves or one of the many other aliases for this virus should be adopted and standardised.

FDISK /MBR

Kevin Powis of *Visionsoft* has brought a little known and very useful feature of *FDISK* supplied with MS-DOS 5 to our attention. This feature will effectively remove many current viruses which infect the Master Boot Sector (Track 0, Head 0, Sector 1).

If a PC running MS-DOS 5 is infected by such a virus, the user can boot from a clean write-protected system diskette upon which is stored a copy of *FDISK*.

The *FDISK* program should then be run from the diskette drive. By typing FDISK /MBR at the A: prompt, clean Master Boot Sector code is written to the first physical sector on the hard disk. Moreover, when this *FDISK* option is invoked, all initialisation data in the 64 byte Partition Table stored in physical sector 1 is left completely intact.

Note: this option should only be used if the Partition Table in the first physical sector on the hard disk is present and correct after the Master Boot Sector has become infected. This may not necessarily be the case with future computer viruses. [*Such viruses may already exist. Tech Ed.*]

Fortunately, most viruses which currently infect the Master Boot Sector do not tamper with the Partition Table.

This feature, combined with the ability of *SYS* to remove DOS Boot Sector viruses (those which infect the boot sector of the active DOS partition), provides users of MS-DOS 5.00 with simple standard tools to remove most boot sector viruses without having to resort to formatting. This *FDISK* option is only available under MS-DOS version 5 - it does not apply to versions prior to 5.

It is still recommended that all PC users store clean write-protected backups of the boot sectors of their fixed disks on diskette.

Non-Compliance

Many resident scanners, monitors and standalone checksumming programs provide the option for PC administrators to customise the screen message which appears when these programs detect virus activity. System administrators can thus instruct the user to contact the relevant PC Support desk and provide other information consistent with company policy.

Unfortunately, a user of unauthorised or stolen software, upon seeing a virus alert reported on his screen may decide not to report the incident to PC Support, particularly so if company policy entails severe disciplinary action for illicit software use. Subsequently such a user may attempt to disinfect his machine which may result in compounded damage.

The problem of the non-compliant user is an extremely difficult one to solve. At the recent *Sysguard 91* conference in London, Noel Bonczoscek of the *Computer Crimes Unit* suggested that developers of memory-resident software might consider incorporating a completely customised alert banner into their monitors - upon detecting a virus the on-screen alert could thus be configured in such a way as not to arouse the suspicions of the 'untrusted user'.

This is an interesting suggestion which merits consideration.

Using *Virus Guard* (from *Dr. Solomon's Anti-Virus Toolkit*) as a representative example, this resident monitor, upon detecting a virus, currently flashes a message to screen which states prominently:

```
Virus Alarm

Dr. Solomon's Anti-Virus Toolkit has Intercepted a
virus:

Close everything down normally, then consult Toolkit
manual for remedy
```

However, Bonczoscek said that PC administrators might prefer to configure a more subtle display along the lines of 'Internal Error: Do Not Proceed, Contact PC Support/ Tel Extension 203'.

This banner would have no reference to a suspected virus attack or the anti-virus software that had detected such activity but would be sufficient to ensure that the more naive user contacted the appropriate support staff.

This is only one suggested (and, to our knowledge, untested) proposal to encourage user-compliance. The potential pitfalls of such a tactic are readily apparent - not least the fact that a genuine operating system error would not direct the user to an 'in house' telephone extension! Any user recognising this fact might well be capable of dealing with a virus attack in the first place!

However, Bonczoszek's underlying point is that computer administrators should never *assume* that users will automatically comply with company policy and report virus outbreaks or other incidents. (To quote Thomas Harris: 'Never assume anything - you'll make an *ASS* out of *U* and *ME*'). Compliance auditing is one of the most complex tasks in computer security - making sure that people understand the rules and that they are following them is a formidable task and technical solutions do not lend themselves easily to it.

Little wonder that the wisest and most experienced computer security managers don't allow users anywhere near detection and diagnostic software - they know that either it will not be used, or that it will be used incorrectly (often with dire results), or that alerts and warnings will simply be ignored!

Rage Change

An amended scan string to detect the Rage virus (*VB*, October 1991, p. 21) has been supplied by Andrew Busey of *Microcom Software Division Inc*. The scan string contains no addresses and should be used in preference to either of the previously published patterns.

```
Rage B9FD 018A 2451 8AC8 D2C4 598B 24FE C046
```

Nobbled Nibble

A one-nibble error in the search pattern published for the Liberty virus in last month's *VB* effectively invalidated the string. An amended and corrected pattern appears below.

```
Liberty   B931 2833 D2CD 1306 BB5C 0653 CB2E
          803E BC06 0A74 4633 C08E
```

Anti-virus software developers should take note that the original pattern for this virus (last published in July 1991) should be maintained as essential scan data. This pattern was extracted from an earlier version of the virus which is not consistently detected by the pattern above. The pattern is repeated here:

```
Liberty   0174 031F 595B 5053 5152 1E06 1E0E
          1FE8
```

KNOWN IBM PC VIRUSES (UPDATE)

Updates and amendments to the *Virus Bulletin Table of Known IBM PC Viruses* as of 20th October 1991. Hexadecimal patterns may be used to detect the presence of the virus with a disk utility program, or preferably a dedicated virus scanner.

Type Codes

C = COM files **E** = EXE files **D** = Infects DOS Boot Sector (logical sector 0 on disk)
M = Infects Master Boot Sector (Track 0, Head 0, Sector 1) **N** = Not memory-resident after infection
R = Memory-resident after infection **P** = Companion virus **L** = Link virus

864 - CN: This virus adds 864 bytes in front of the files it infects. Awaiting analysis.

```
864             B04D B449 B742 473A 2575 153A 7D01 7510 3A45 0275 0BC6 4502
```

1876 - CER: This 1876-byte virus is probably of Polish origin. Awaiting analysis.

```
1876            8EC0 33FF 33C0 B9FF 7FFC F2AE 26F6 05FF 75F8 83C7 038B D72E
```

Best Wishes-970 - CER: This virus is detected by the search pattern for the Attention virus, but not by the pattern for the Best Wishes-1024 virus. This variant is not able to infect .EXE files properly.

Black Wizard - EN: A variant of the 'Old Yankee' virus and detected by the pattern for that virus. This variant is 2051 bytes long and plays a different tune than the original virus, but is otherwise similar.

Bulgarian 123 - CN: A simple 123-byte virus from Bulgaria, which does nothing but replicate. It may infect the same file repeatedly.

```
Bulgarian 123   B103 8D54 F4B4 40CD 21B4 3ECD 21B4 4FCD 2173 AFBB 0001 FFE3
```

Copmpl - CER. This is a 1111 (COM) or 1114 (EXE) byte Polish variant of the Akuku virus. The name is derived from the following text, which can be found inside the virus 'Sorry, I'm copmpletly dead' (sic). The only effect of the virus is to play a tune.

```
Copmpl          80E6 0F8A D680 FA00 7407 80FA 0B76 06B2 02B4 0ECD 218C C88E
```

Copyright - CN: A 1193-byte virus from East Europe, which contains a fake Award BIOS copyright message. Awaiting analysis.

```
Copyright       AB4A 75F2 E2EA 33C0 CD16 B800 06B7 0733 C9B6 18B2 4FCD 10E9
```

DIR-II - LCER: A 1024-byte 'link' virus from Bulgaria. 'Infects' all COM and EXE files in each directory on a single pass. If the virus is resident, 'infected' COM and EXE files can be disinfected by renaming their extensions. (*VB*, Nov 1991).

```
DIR II          BC00 06FF 06EB 0431 C98E D9C5 06C1 0005 2100 1E50 B430 E824
```

DM-400 - CR: This 400-byte virus does not seem to do anything but replicate. It contains the text '(C)1990 DM'.

```
DM-400          80FC 4B74 3380 FC56 7419 FE04 80FC 3D74 12FE 0480 FC3E 751C
```

Europe '92 - CR: This 421-byte virus activates if the year is set to 1992, when it displays the message: 'Europe/92 4EVER!'

```
Europe '92      B450 CD21 8CD8 488E D8C6 0600 005A 891E 0100 8916 0300 53B8
```

Fake-VirX - CN: A 233-byte virus from Finland which activates on any Friday the 13th, when it displays the message 'VirX 3/90'.

```
Fake-VirX       408B D5B9 0600 CD21 B801 575A 59CD 21B4 3ECD 21B8 0001 FFE0
```

Gergana - CN: Four variants of the Gergana virus, which are longer than the original with improved error handling.

```
Gergana-222     BF80 FFB9 3000 F3A4 E9C6 FD5E 81C6 0001 BF00 01B9 DE00 F3A4
Gergana-300     BF80 FFB9 3000 F3A4 E985 FD5E 81C6 0001 BF00 01B9 2C01 F3A4
Gergana-450     BF80 FFB9 3000 F3A4 E97E FD5E 81C6 0001 BF00 01B9 C201 F3A4
Gergana-512     BA00 FAB4 3FCD 21C3 B900 02B4 40CD 21C3 B801 572E 8B0E 5001
```

Gosia - CR: A 466-byte virus from Poland. It contains the text 'I ♥ Gosia'. ♥ is the ASCII character (03))

```
Gosia           0275 10AC 268A 2547 3AC4 7405 80CC 203A C4E2 EE9F 03F9 8B1D
```

Gotcha - CER: Two related viruses from East Europe, 879 and 881 bytes long. They contain the text string: 'GOTCHA!'.

```
Gotcha          9C3D DADA 7428 80FC 3D74 0A3D 006C 7405 80FC 4B75 1306 1E50
```

Hero-394 - ER: Related to the 506-byte Hero virus, but does not damage the files it infects. Awaiting analysis.

```
Hero-394        B98A 0133 C0BF 0002 0305 83C7 02E2 F929 069C 03B8 0042 33C9
```

Hungarian-482 - CR: This 482-byte virus from Hungary activates on November 7th. If an infected program is run on that date it will display the string 'Format ...' and proceed to format the hard disk.

```
Hungarian-482   5603 F7AC 0AC0 740A D0E8 B40E B307 CD10 EBF1 B901 00BA 8000
```

Iron Maiden - CN: A 636-byte virus, which contains the text 'IRON MAIDEN' near the end. It has not been fully analysed, but contains destructive code (INT 26H calls). The original sample of this virus was also infected with the Polish W13-A virus.

```
Iron Maiden     2425 CD21 5F0E 1F8B 8557 02A3 0001 8AA5 5902 8826 0201 B41A
```

Leningrad - CN: Two viruses, 600 and 543 bytes long, first reported in Leningrad (now St. Petersburg), and probably written by the same author. The 600 byte variant has not been analysed, but the other variant will activate on any Friday the 13th, and display the message 'That could be a crash, crash, crash!'. These viruses were described last month as Sov1 and Sov2.

Murphy-Brothers - CER: A 2045-byte Murphy variant. Contains the text: 'Brothers in arms'. Detected by the HIV pattern.

Omega - CN: A 440-byte virus from Finland. It overwrites the beginning of the first two hard disks, trashing the Partition Table.

```
Omega           B05C AA89 7E2E 83EC 15B9 1500 8BFC 8BF5 A4E2 FDE8 1D00 B915
```

Path - CN: A 547-byte virus from East Europe, which searches the path for files to infect.

```
Path            B90D 0057 8A07 8805 4347 E2F8 C605 005F B801 43CD 21B8 023D
```

PC-Flu - CR: This 802-byte virus from Poland was made available with the original commented source code from the author. It seems to be intended to bypass three specific anti-virus programs, *Flushot*, *Vstop* and *Virblock*, but this has not been tested.

```
PC-Flu          501F BB00 0180 3FE9 7537 4380 3F15 7531 4380 3F05 752B B800
```

PC-Flu-2 - CER: A 2112-byte variant of the previous virus using self-modifying encryption. No simple search pattern is possible.

Plovdiv, New Bulgarian 800 - CR: This virus is 800 bytes long, but the increase is hidden while the virus is active. It contains the text '(c) Damage inc.Ver 1.1, Plovdiv,1991', but has not been fully analysed yet.

```
Plovdiv         80E2 1F80 FA1E 7506 2681 6F1D 2003 079D 5A5B EB02 CD32 559C
```

Polish Color - CN: A simple 376-byte Polish virus, which does nothing but replicate.

```
Polish Color    01BF 0000 B900 01F3 A45E 8BC6 BF00 00B9 0001 F3A4 5E8B C605
```

Polish Minimal-45 - CN: An attempt to create the world's smallest virus. It overwrites the files it infects which cannot be disinfected.

```
Polish-45       023D CD21 8BD8 B440 BA00 01B1 2DCD 21B4 3ECD 21B4 4FEB DCC3
```

Polish Pixel - CN: Two Pixel variants from Poland, which contain crude self-modifying code. They are 457 and 550 bytes long, and detected by the Pixel-277 pattern.

Rybka - CER: This is a variant of one of the Vacsina (TP-series) viruses. It may infect the same file over and over, increasing its size by 1344 bytes each time. Detected by the Vacsina pattern.

Something - CR: A 658-byte virus, which attaches itself in front of .COM files. It appears destructive, containing code to delete files.

```
Something       8BD8 B9FF FF1E 5233 D22E 8E1E 8303 B43F CD21 725F 3D00 E873
```

SVC-1740 - CER: This 1740-byte virus is closely related to the 1689-byte variant (SVC 4.0) and is detected by the same pattern.

SVC 5.0 - CER: An improved version of the earlier SVC viruses, and fully 'stealth'. Awaiting analysis.

```
SVC 5.0         5606 86E0 35FF FF8E C00E 1F33 FFB9 990B FCF3 A607 5E74 03B9
```

Vienna-726 - CN: A 726-byte variant, detected by the Vienna (4) pattern.

Vienna-Polish 634 - CN: This modified version is detected by the Vienna (1) pattern.

Vienna-776 - CN: A 776-byte variant. A similar 757-byte variant has also been found. Awaiting analysis.

```
Vienna-776      B44E BADD 0003 D6B9 0300 CD21 EB04 B44F CD21 7302 EB9F 8B84
Vienna-757      B44E BA5B 0003 D6B9 0300 CD21 EB04 B44F CD21 7302 EB9F 8B84
```

Voronezh-370 - CR: This virus is related to the Voronezh and USSR-600 viruses, perhaps their common ancestor.

```
Voronezh-370    0500 018B F0BF 0001 FC8A 0434 EB88 0546 47E2 F6B8 0001 50C3
```

W13-C - CN: A minor modification of the 507-byte W13-B variant. The only modification is that this variant sets the month field to 12, not 13, which makes all files created in December immune to infection. Detected by the W13 pattern.

W13-361 - CN: A member of the W13 group of Vienna-related viruses. It is detected by the W13 pattern, but does not function properly, as infected programs (second generation) will never run. A 377-byte variant also exists which replicates without problems.

Words - CER: A series of 4 Polish viruses, 1069, 1085, 1387 and 1503 bytes long. The two longest variants use self-modifying encryption, and no simple search pattern is possible for them. The other variants can be detected with the following pattern:

```
Words           8066 0EFE 5958 8BC1 5E5D 9DCF 528B D6B4 409C 2EFF 1E0D 005A
```

Yankee-1150 and **Yankee-1205** - CER: Two stripped-down versions of the Yankee virus which only replicate.

```
Yankee-1150     CB5B 5383 EB44 C32E 80BF 0100 0074 0681 FCF0 FF72 E58C D848
Yankee-1202     CB5B 5383 EB45 C32E 80BF 0100 0074 0681 FCF0 FF72 E58C D848
```

TOOLS & TECHNIQUES

David M. Chess

High Integrity Computing Laboratory
IBM Thomas J. Watson Research Center
Yorktown Heights, NY, USA

Virus Verification and Removal

The first line of defense against computer viruses consists of programs that detect that something is probably wrong. These include modification detectors, integrity shells, known-virus scanners, access-control programs, and similar things. Their main function is to alert the user of a machine that a virus, some virus, is probably present. The important thing is the alert; since something is likely to be wrong, the user should stop what he is doing, and take action to correct the problem. It doesn't matter much at this stage what the alert says; a first-line anti-virus system that always said simply 'Something virus-like may be going on!' would be sufficient for most environments, if it were usually right.

Once the alert has been given and the infected system taken out of immediate contact with other systems, other kinds of software become important. Before we can decide how to clean up an infected system, and even where else to look for infection, we need to know exactly what the infection consists of. This paper is a description of one part of the second-line toolbox, the virus verifier (and remover).

Virus Verifiers

A virus verifier is a program that, given a file or disk that is probably infected with a given virus, determines with a high degree of certainty whether the virus is a known strain, or a new variant. This is, of course, important to know: if the virus is different from any known strain, it will have to be analyzed for new effects before we can be confident that we know just what to do to clean up after it.

On the other hand, if the virus is identical to a known strain, we already know what to do. It is particularly important to perform verification in a program that attempts to remove the virus infection from an object automatically, restoring it to its original uninfected form.

In abstract, a verifier is a program that, given another program as input, determines whether or not the given program is part of the set of possible 'offspring' of a particular virus. For many classes of viruses, including all the viruses actually widespread at the moment, this is easy to do. Almost all known viruses consist almost entirely of code that does not change from infection to infection, except perhaps for a simple XOR-type garbling, and data areas that are either constant, or change in simple ways (or that can be ignored entirely for the purposes of verification).

Given a suspect file F and a known virus V, it is therefore always relatively simple to answer the question 'is F a file that could have been produced by infection with virus V?'. It is an open question of some theoretical interest whether or not some future virus might make this harder to do! Reliably determining whether a file is infected with any virus at all is of course known to be impossible, but we have no similar constraint on determining the presence of a specific virus.

Trade-offs and Development Decisions

There are various concrete decisions and trade-offs involved in writing a virus verifier; this section will list a few of them, while the next sections will describe the verifier currently being developed and used at the *High Integrity Computing Laboratory* at *IBM's Watson Research Center*.

A verifier may be an independent tool, or it may be integrated into a virus detector. An integrated detector/verifier can be quicker and more convenient, since there's no need for a user to find and run a verifier once the detector goes off. On the other hand, since most copies of any virus detector will never in fact detect a virus (most of the world's computers are not infected, after all), integrating a verifier along with the detector is in some sense inefficient, in that it adds significant code to the detector that may never be used. However, given how much more expensive human time is than CPU time and disk space these days, integrated tools are likely to be more cost-effective in the long run.

Detection and verification will always be two different activities, because it is very desirable for a detector to detect small variants of known viruses as viruses, whereas a verifier must be able to identify any variation as a variation. Detection algorithms are typically run very often, and must be fast. Verification algorithms, on the other hand, are run rarely (only when a virus is detected), and speed is typically not a major issue.

To determine whether or not a given object is infected with a known strain of a virus, a verifier must know what the known strain looks like. This may be done either with an actual copy of the code of the known strain of the virus, or by using a CRC or similar modification-detection algorithm. It's not generally desirable to include the entire code of a virus with widely-distributed tools, for obvious reasons! On the other hand, even a good difficult-to-invert digital signature algorithm is not as reliable as a byte-for-byte comparision, and it is vulnerable to a virus author intentionally creating a variant that looks to the verifier like a known strain. (This can be made more difficult through the use of cryptographic check-summing and related technologies, at some increase in runtime and complexity.)

Lastly, a verifier may use either special-purpose code, with one or more routines being written in some compiled language for each new strain discovered, or it may be written as an interpreter for a high-level virus-description language.

A high-level language is generally simpler to program in reliably; on the other hand, this is only true because it is less expressive, which implies that there will be cases (exotic self-garbling viruses, for instance) in which it will be necessary to drop into the lower-level programming language again.

VERV - A Prototype Virus Verifier

At *HICL*, we are currently using and developing a virus verifier called 'VERV' for PC-DOS viruses. The current version can verify over 30 different viruses and variants, which accounts for nearly all of the actual infections that we see in day-to-day operation. As well as being used in the lab, and as a research prototype, VERV is used by *IBM's* internal *Computer Emergency Response Teams* (CERTs), as part of routine incident handling.

It is an independent tool at the moment; in the long run, we expect to integrate it with our other anti-virus programs. It can use either the CRC algorithm or a byte-for-byte comparison to verify the identity of a virus. In the laboratory, we use the byte-for-byte comparison to test new samples against old ones. In the field, the CRC algorithm is used to verify the virus in infected objects before applying cleanup measures.

VERV includes an interpreter for a small virus-description language. Virus-description languages, for this and other purposes, have been around for some time; Christoph Fischer at the *University of Karlsruhe*, Morton Swimmer at the *University of Hamburg*, Alan Solomon in the UK and no doubt many others in the field, have worked on similar things (a personal note; one motivation for publishing this paper is to encourage others, who have perhaps done it better, to publish their work).

VERV's language is very simple and provides for lower-level hooks (instructions to call special-purpose 'C' routines) when a virus requires actions that cannot be described in the high-level language.

The part of the language that is currently implemented supports only virus verification; features to support virus removal ('disinfection') as well have been designed, but not yet implemented.

We will describe the language in some detail, not because it is particularly interesting as a language, or because we think we have it all correct and optimal, but rather so that other people working on the same sorts of things can benefit from our ideas and learn from our mistakes.

We hope this paper will help inspire continued discussion and exchange.

VERV's Virus Description Language

The file from which VERV reads virus descriptions consists of a number of virus-description blocks. Each block has the following structure:

One or more VIRUS records
A NAME record
One or more LOAD records
Zero or more DEGARBLE and related records
Zero or more ZERO records
One or more check records
Zero or more REPAIR blocks

For instance, the block for the Slow-1721 virus currently looks like this:

```
VIRUS slow slow-1721
NAME the Slow-1721 virus
LOAD P-COM 0 6B4
LOAD S-EXE 0 6B4
DEXOR1 001E 06AD 0012 0000      ; Degarble the code
DEXOR1 00EB 0159 0061 0001      ; and the data area
ZERO 0012 1                     ; Zero out the one-
                                ; byte garble key
ZERO 0061 1                     ; and the data-garble
                                ; key
CODE 0000 00EA 38d5dc08         ; Code up to first
                                ; data area
CONST 0144 014E 0ff22ad9        ; COMMAND.COM
CODE 015A 063C 74e00962         ; Code between data
                                ; areas
CODE 0657 06AD ad3b0b41         ; After the second
                                ; data area
                                ; Randomizer
```

The VIRUS records simply give a list of one-word aliases for the virus, that are used on the command line to tell VERV which virus to look for. These aliases are not the full primary name of the virus (that is given on the NAME record); they are just short abbreviations that the user can use on the command line.

A very useful extension here would be for VERV to support virus families, so that a single command would cause testing for all members of the Jerusalem family, or the Flip family, and so on. When integrated into the virus detector, of course, the detector will directly inform the verifier which virus or viruses to test for.

The LOAD records describe where in an infected object of a given type the virus can be found. The tokens on a LOAD record are an object type, followed by either an offset and a length, or the word SPECIAL and a number. The offset tells VERV where, relative to the effective entry-point of that sort of object, to start loading; the length tells how many bytes to load. For viruses that are not always at a fixed offset from the initial entrypoint, the SPECIAL keyword causes VERV to invoke an internal routine, coded in C, to perform the loading.

The Slow virus is an EXE-infector, and a prepending COM infector; the LOAD records in this example tell VERV to load the first 06B4 bytes of a COM-format file, and the first 06B4 bytes after the entry point of an EXE-format file. (EXE-format files are those that begin with the letters 'MZ'; DOS loads these differently from COM-format files, which begin with any other bytes.) Other object types supported include:

- E9-COM, for viruses that infect COM files by changing the first three bytes to a long jump to the virus (E9 is the hex code for a long jump),
- E8-COM, for viruses that infect COM files by changing the first three bytes to a long CALL to the virus (E8 is a long call),
- MBR, for viruses that infect hard disk master boot records and diskette boot records, and fit in a single sector,
- DISKETTE, for other sorts of diskette infectors (those that do not fit in a single sector),
- HARDDISK, for other sorts of hard disk infectors (those that infect system boot records, and/or occupy more than one sector).

A description block will have as many LOAD records as there are types of object that the virus can infect.

The DEXOR1 records tell VERV to perform a certain common type of degarbling: a one-byte XOR with data to be found at a fixed offset in the virus. The details are not terribly important here. A more general record, consisting of just the word DEGARBLE followed by a number, causes VERV to invoke an internal C-language routine to perform degarbling.

Once the loading and degarbling have been done, VERV has a complete 'virus image' in its internal buffer. A command-line switch (described later) can instruct VERV to save the contents of this buffer to a file, for later examination.

The ZERO records describe variable areas within the virus, that should be set to zero before checks are done. This is really just a convenience, to reduce the number of check-type records needed.

There are three basic types of check records, describing different tests to be done on the degarbled and zero'd virus image now in the buffer:

- CODE records describe areas of virus code. The numbers given are the start and end offsets of the area, and the expected CRC value of the data there. VERV uses a 31-bit CRC, with a custom polynomial. This is not strongly resistant to intentional reverse engineering; a more difficult-to-invert algorithm may be desirable later on. If any CODE areas are found to be different than expected, VERV will report that this is not the usual strain of the virus.
- CONST records describe constant areas that should not change, and whose values effect the actual running of the virus. CONST areas are currently treated exactly like CODE areas.
- TEXT records describe areas of the virus that are not expected to change, but do not significantly affect the operation of the virus. If a sample differs from the given description only in one or more TEXT areas, VERV will report a 'text variant' of the virus. This is useful for message areas within a virus that are not actually used, or that are simply displayed to the user. These areas can be interesting in tracking how the virus is spreading, by correlating incidents that involve the same 'text variant', but they do not affect cleanup or prevention.

Normally, VERV performs its CRC calculation on each area within the virus, and compares the results to the expected values. A command-line switch (described in more detail below) can be used to tell VERV to read a standard copy of the virus from another file instead, and do byte-by-byte comparison between the two. This is more reliable, but of course it requires having a sample of the usual strain of the virus present against which to verify.

Another example, illustrating the use of special C routines, is the block for the 1701 virus:

```
VIRUS 1701
NAME the 1701 virus
LOAD E9-COM -1 06A5
DEGARBLE 1
CODE 0001 0026 19989c7e      ; Degarble, MOV, jmp-in
CODE 0076 06A4 c03a91c5      ; Main code
```

Here, the 'DEGARBLE 1' record causes VERV to invoke an internal routine to degarble the data in the buffer, using the 1701's own algorithm. It would be possible to enhance the virus-description language enough that the 1701's degarbling algorithm could be expressed in it directly. This would complicate the language considerably, though, and would somewhat lessen the advantage that a special high-level language has over native C code; so far, we have decided against such enhancements.

Repair

For many viruses and many infected objects, it is possible to restore the object to what it looked like before it was infected, or at least to a state in which it will function in just the same way. Unfortunately, this isn't always possible; the classic example is the 1813 (Jerusalem) virus infecting an EXE-format file. While it's usually possible to undo the infection, sometimes the resulting file is missing data that was in the uninfected original, and it's not always possible to tell that this has happened. The best a Jerusalem 1813-remover can do on the EXE file, therefore, is something that is quite likely to

work, but might not. In most cases, though, reliable repair is possible and particularly in cases of widespread infections on non-critical machines, repair is sometimes a cost-effective option.

Our current design for a repair block in VERV calls for one block per type of object that the virus may infect. Each repair block consists of a header record 'REPAIR <object type>', followed by one or more repair-operation records.

Currently defined repair operations include:

- a REMOVESTART record that removes the first so many bytes from the file being repaired,
- a REMOVEEND record that removes the last so many bytes, shortening the file,
- a COPY record, that copies so many bytes from a given offset in VERV's internal buffer (which initially holds an image of the virus) to a given offset in the file being repaired,
- RLOAD and RENDLOAD records, that load a given number of bytes from a given offset (relative to the start or end of the file) into VERV's buffer,
- an RSPECIAL record, to cause VERV to invoke an internal C routine to perform some function.

For instance, the repair blocks for the usual 1813 or Jerusalem virus might look like this:

```
* Remove from a normal EXE file
REPAIR S-EXE
RSPECIAL 2                  ; Check header length and
                            ; adjust, give warning
REMOVEEND 0710
COPY 0043 0010 2            ; Fix initial SP
COPY 0045 000E 2            ; and SS
COPY 0047 0014 2            ; and IP
COPY 0049 0016 2            ; and CS
COPY 0051 0002 4            ; Length, adjusted above

* Remove from a normal COM file
REPAIR R-COM
REMOVESTART 0710 ; That was simple!
```

The RSPECIAL record would perform 1813-specific processing, such as checking to make sure there is nothing in the file past the end of the outermost 1813 infection (if there is, it probably means that the 1813 virus has overwritten part of the original file due to one of its many bugs), computing the approximately correct EXE length for the repaired file's header and warning the user that EXE files that have had the 1813 virus removed from them do not always work correctly.

Some of these functions are so common that we will probably incorporate them into the language itself at some point.

After repair is completed, VERV should restart processing on the repaired file, to ensure that there is not another instance of the virus present. Once VERV is integrated with a virus scanner, the repaired file will be re-scanned automatically for all viruses, and if one is found the file will be re-verified.

Repair processing is only performed if the user has requested it on the command line and if VERV finds that the virus is indeed exactly the known strain of the virus. In small infections, or in situations where correct operation of the objects involved is particularly crucial, we continue to recommend that infected objects be destroyed (files erased, diskettes formatted and so on), and replaced from uninfected sources.

VERV Options

The functions of VERV's command-line switches include:

- Reading the virus to be tested from an image file, instead of from a normally-infected object; this can be useful, for instance, in testing a boot-sector infector that has been received as a binary dump of the boot sector rather than on diskette.
- Overriding the default virus-description file (contained within VERV.EXE itself), allowing easy testing of new or experimental descriptions.
- Producing detailed progress messages and data displays during processing, to help pinpoint differences found or errors encountered.
- Specifying that, rather than using a CRC, VERV should compare the relevant parts of the object to be tested with a standard sample of the virus stored in an image file, or a standard infected specimen.
- Producing a dump of the virus image, after all degarbling, but before any zeroing has been done. This image can then be used for storage, analysis, or transmission, or for later use as input to VERV for byte-by-byte comparisons.

Status and Future Goals

VERV is currently in use by a small number of people within *IBM* who deal with virus infections. Its availability has greatly reduced the time spent by technical people in doing semi-manual verification, and has therefore sped up the response time to virus incidents. Adding a typical newly-analyzed virus to VERV is generally quite simple, involving a few lines in the VERV language, and sometimes a small piece of C code to handle a new garbling algorithm.

Our near-term plans for VERV include support for families of viruses, and the ability to verify a virus in a number of objects at once. This will ease integration with our virus detectors; when a detector detects a signature that corresponds to a virus, or a family of viruses, in a number of files, it will be able to verify the identity of the virus with a single call to VERV.

Once the virus-removal language is implemented, disinfection could be done at the same time (although we often recommend replacement rather than repair).

The Ideal Scenario

If transmission bandwidth, CPU cycles, and disk space were free, and programming were easy, every workstation would be protected by a seamless 'immune system'. Objects infected with existing viruses would be detected automatically, the identity of the virus verified and reported to a central location, and the object destroyed or repaired, with minimal user intervention. New viruses would be detected automatically with some high degree of confidence, first-pass signature patterns would be extracted automatically where possible and communicated to a central clearing house, along with a sample of the suspicious object. Viruses would very rarely, if at all, spread widely.

One of our main focuses at *HICL* is studying what part of that ideal scenario is feasible, in both current and future systems. The prototype VERV is a small part of our experimentation with parts of that system that are also immediately useful to users in the near term. We would welcome similar descriptions by others in the field, of work that they are doing in similar directions.

Virus Bulletin Conference 1992

Call For Papers

Abstracts of between 300 - 1,000 words are invited for papers to be presented at the *Second International VB Conference* in September 1992.

The conference will be in two streams: Stream one will address the management of the virus threat within the corporate environment, while stream two will concentrate on technical developments including virus disassembly, detection and classification.

Abstracts are welcomed from individuals or groups active in research, software or hardware development, quality assurance, the law, corporate security management, or any other field related to countering computer viruses and malicious software.

Abstracts, should be completed by February 15th 1992 and should be sent to The Editor, *Virus Bulletin*, 21 The Quadrant, Abingdon Science Park, Abingdon, Oxon OX14 3YS, UK. Fax 0235 559935.

VIRUS ANALYSIS 1

Jim Bates

DIR II - The Much Hyped 'Linking' Virus

Once again certain sections of the anti-virus industry have been crying wolf. Global virus alerts and panic bulletins have been posted on countless BBS forums - the end of computing, we have been assured, is imminent. While this activity undoubtedly does no harm to their sales figures, once again it is the user who is left confused and frightened by unconfirmed reports of uncontrolled infections in Eastern Europe by a new virus. One report describes it as '*A new, fast moving, and very destructive virus ... uses a completely new technique ... cannot be easily identified or removed*'. Small wonder that normal research work has been punctuated by frequent telephone calls from concerned users enquiring about the imminence of the annihilation of their programs and data.

Facts Not Fiction

The facts simply, are these: the virus uses a new infection technique; it spreads rapidly within a machine environment by virtue of its infecting whole directories; and (incidental to its method of operation) it displays a certain stealth capability. However, it is not 'very destructive', it is extremely *easy* to identify and it can be disabled and removed with no risk to existing files. Its potential for corrupting existing data is limited to a single cluster (probably unallocated) on each infected disk. All of this information, together with a simple and effective disinfection technique, was gleaned from disassembling and analysing a sample of the code and took under 24 hours of research effort.

The virus comes from Bulgaria. It is known as DIR-II and the fact that most researchers agree on this name is an indication of how rapidly a sample was distributed amongst researchers around the world. The fact that the sample was accompanied by unconfirmed tales of the virus 'running rampant' in Eastern Europe is probably the reason for the panic, as to date only a single report of it being 'at large' (which subsequently turned out to be false) has been received from my own contacts.

General Analysis

Since this virus does not attach itself to executable file contents, the term 'linking' virus has been coined to differentiate its infection method. The code is 1024 bytes long and is written in assembler. It does not subvert any of the interrupt vectors although it does collect one of the antique access points designated for compatibility with CP/M. Access to the system is gained by attaching the code to the existing device drivers which handle part of the DOS file services.

VIRUS BULLETIN ©1991 Virus Bulletin Ltd, 21 The Quadrant, Abingdon Science Park, Oxon, OX14 3YS, England. Tel (+44) 235 555139.
/90/S0.00+2.50 This bulletin is available only to qualified subscribers.

The infection technique links a single cluster containing the virus code to each file that is infected while preserving the original location of the start of the file in an encrypted form within the directory entry. This means that the actual contents of each infected file are not altered in any way and can be accurately and completely recovered.

The virus makes no attempt to encrypt its code and it is therefore easily identified by even the simplest scanning program when used 'clean' to scan suspected floppy disks. However, it should be noted that if any other parasitic virus occurs on such a disk, the scanning program will not find it until the DIR-II infection has been removed.

The virus infects all EXE and COM files (including COMMAND.COM). This virus will **not** propagate across a network unless an infected file is copied to the file-server.

Installation

The virus code begins by setting up its own local stack and then incrementing a single word data value. This word is not referenced elsewhere in the code and may be a simple infection counter. The Interrupt Table is then examined and an address is collected from a position reserved solely to maintain compatibility with CP/M type programs. This address points to an entry point within the operating system and after a small modification (to skip certain initialisation code) it is stored via the local stack in the virus' data area for later use as the main link between the virus code and DOS.

A normal interrupt request is then issued to determine the version of DOS and the virus then attempts to initialise a data value. The attempt fails because of a bug in the code but this does not affect the overall operation of the virus.

After modifying the amount of memory used by the virus, the code examines the chain of device drivers and modifies all those found to be resident within the DOS segment so that they become linked to the virus code. Calculations are then performed on specific memory allocations to determine whether the virus code is being processed before COMMAND.COM is loaded. This information is vital to the successful installation of the virus in low memory.

Once the virus is installed as an extension of the operating system, the addresses of the Strategy and Interrupt routines of the device driver are collected into the virus' data area for use later. The next part of the installation routine is reminiscent of some other Bulgarian viruses and begins by collecting the Disk BIOS address from within the device driver. This is stored within the virus' data area but an additional routine searches for any ROM modules and examines them for their own Disk BIOS address. If found, this address is used in preference to the one from the device driver.

The final part of the installation routine collects the name of the host program and executes as a child process via the normal LOAD and EXECUTE function of DOS. After execution, any returned error code is collected and the code exits normally to DOS. Provision is made for the execution of COMMAND.COM since it is possible for the virus code to be processed before this file is loaded.

Operation

Once installed, the virus code remains resident and is invoked whenever DOS issues commands to the device drivers associated with the Disk BIOS. Thus all disk accesses requiring collection of the directory structures become routed through the virus. The connection to the device drivers functions in a similar way to the redirection of interrupt services in other viruses - various commands are intercepted and others are allowed to continue unchanged. In this case only the Read, Write, Write/Verify and Build BIOS Parameter Block (Build BPB) commands are intercepted.

The Read and Write commands are intercepted *before* processing by the device driver, while the Build BPB command is intercepted *afterwards*.

Build BIOS Parameter Block

The interception of the Build BPB command is implemented to prevent possible corruption of the disk structure either by DOS or by *CHKDSK* and similar utility programs. The BIOS Parameter Block is of vital significance to DOS when calculations are undertaken to ascertain the location and volume of available space on the disk. Remember too that the BPB is constantly being rebuilt as different disk types and sizes are accessed.

After the device driver has completed the command, the virus code regains control and copies the newly built BPB into its own buffer. The contents of the request header are modified to point to the virus copy of the BPB and the block itself has an appropriate number of sectors (corresponding to the virus' requirements) subtracted from its Total Sectors field. Thus, on an infected machine, *CHKDSK* does not indicate any errors or orphan clusters. Note however, that if an infected machine is booted from a clean system disk, *CHKDSK* will indicate an indeterminate number of cross-linked and orphan clusters.

Output and Output/Verify Commands

These commands, which are issued to the device driver when directory sectors must be rewritten, are intercepted by a single routine in the virus before the device driver receives them.

This routine begins by checking to see whether the media has changed in the target drive (new floppy, etc.). If the media has changed, the original output request is issued to the device driver request and when processing returns, the virus continues with the disk infection routine. If the media hasn't changed, processing jumps to the sector infection routine.

Input Commands

Intercepting the Input commands to the device driver involves first checking to see whether the media has changed. If it hasn't then the sector infection routine is called - if it has then the disk infection routine is invoked.

Infection Routines

The two infection routines are where this virus differs from other types and it is important to understand the mechanisms involved before describing the routines themselves.

On a single infected disk (hard or floppy) the virus code usually reside in only one place - the final one or two clusters on a disk. Differentiation between disk types is according to how many sectors are allocated to each cluster on that particular media. The routine that locates and places the virus code in this area is referred to here as the disk infection routine. Once a disk has been infected in this way, the other infection routine is used to link the virus cluster to target files. This is done by directly accessing the directory sectors on the disk and modifying the directory entry for each suitable file.

"Far from being the dangerous and frightening beast that these 'experts' imply, this virus is one of that rare breed that can be removed completely by any reasonably literate user without the use of any specialist tools whatsoever!"

Only COM and EXE files are affected and modification consists of replacing the First Cluster Pointer (FCP) of each file entry with a pointer to the single virus cluster (thus effectively cross-linking the files). The original FCP is then encrypted and stored in a different section of the directory entry which is apparently unused (marked as 'reserved' by the reference books). This routine is referred to here as the sector infection routine and it should be noted that regardless of size, the whole of a target directory is infected in a single pass. It is this process which makes the virus spread so rapidly within a single PC [i.e. *the intra-machine environment. Ed.*].

It is interesting to note that the virus only examines the *file extension area* of the directory entry and does not check to see whether the target file has been deleted - although this has no effect on the normal virus operation, it does make this the first virus I've seen which actually 'infects' deleted files!

Disk Infection

This routine completes a simple series of calculations to determine the number of the last available cluster on the target disk. The relevant FAT entry is then inspected and if a bad cluster is indicated the position is decremented and the routine tries again.

Once a usable cluster is located, the FAT entry is marked with an End of File (<EOF>) indicator and the virus code is copied to that cluster. The <EOF> marker used is 0FFFEH instead of the more normal 0FFFFH, this is perfectly acceptable to DOS and enables the virus to detect whether the disk is already infected. This is the only section of the virus code that might cause damage to an existing file since no notice is taken of whether the cluster is already allocated to a file. If the cluster was allocated, the file to which it belonged will be irreparably damaged. Deleting the file will temporarily free the virus cluster but the disk infection routine will immediately re-allocate it and the only side-effect will be that any subsequent portion of the file's cluster chain will become orphaned.

During the disk infection routine, the master encryption key (based upon part of the virus cluster number) is generated and stored within the virus code.

Sector Infection

This routine first calculates the size of the target directory and then calls an infection toggle routine. The toggle process examines each directory entry in turn to see whether the FCP points to the virus cluster. If it doesn't the existing FCP is collected, encrypted and placed in the reserved area of the directory entry. The true FCP field is then altered to point to the virus cluster. If the entry is already infected, the reverse happens and the virus disinfects the entry by repairing and replacing the original FCP.

The process of switching First Cluster Pointers is incidentally responsible for giving this virus a spurious stealth capability since while the virus is resident, DOS operations will only occur on disinfected files and cannot access the virus cluster.

This toggling of infection/disinfection provides a useful Achilles' Heel through which the virus can be de-activated and made to clean up infected disks and machines. Classical scholars will recall that the original Achilles only had one vulnerable heel - this virus has two, and a glass jaw as well! These flaws in the virus are described in detail later.

Observations

As noted above, scanning floppy disks on a clean machine will instantly reveal the presence of the virus. In fact, if the

target disk is infected with the virus (rather than containing just a single program containing the virus code), the result of a scan will indicate all COM and EXE files in a particular directory as infected, thus reducing the possibility of false positive identifications from a single file which just happens to contain a sequence similar to the recognition string.

However, the infection technique does pose some new problems for users when dealing with infected machines and disks. To explain these problems it may be easier to describe various sequences of actions and their results:

Since without the virus being resident and active in memory all infected files appear to point only to the virus code, copying (on a clean machine) an infected file from an infected diskette will result in the destination file being exactly 1024 bytes in length. This is regardless of the length reported by any directory listing of the source diskette. If this copied file is then executed, nothing will appear to happen (the DOS prompt will just re-appear). However, the virus will load into memory and activate and the files on the infected floppy will now appear 'normal' to the system and may be freely copied and executed.

Infection of the fixed disk in this way will almost invariably lead to the infection of COMMAND.COM (or whatever the command interpreter is named) and thereby ensure the survival of the virus at the next reboot. If instead of copying the original file from the floppy, it is simply run from the diskette, the same result will occur - the virus will be resident and active and infect the fixed disk (and probably COMMAND.COM).

Detection

The virus employs no encryption and is easily detected on disk. As with all cases of suspect virus infection the PC should be booted from a clean, write-protected system diskette before scanning commences. Note that the virus will be found in all COM and EXE files in each directory scanned - if the presence of the virus is detected in only a single file, a false positive should be suspected.

The following hexadecimal pattern reliably detects the DIR II virus:

```
DIR II   BC00 06FF 06EB 0431 C98E D9C5 06C1
         0005 2100 1E50 B430 E824
```

Disinfection

Once a machine becomes infected, COM and EXE files cannot be backed up by copying them to an appropriate floppy disk because doing so will infect the target diskette. Booting the machine from a clean floppy is also no use because without the virus active in memory, all infected files will only contain the 1024 bytes of virus code.

This apparent dilemma is described in one panic message as a 'Catch 22 situation'.

In fact the solution is simple and obvious. Far from being the dangerous and frightening beast that these 'experts' suggest, this virus is one of that rare breed which can be removed *completely* by any reasonably literate user without the use of *any* specialist tools (i.e. anti-virus software) whatsoever. The removal procedure (which is astonishingly straightforward) goes like this:

Boot the machine normally from its hard disk so that the virus is resident and active, and then rename all COM and EXE files to different extensions (perhaps .CO and .EX), thus:

```
C:>REN *.COM *.CO

C:>REN *.EXE *.EX
```

The result of this renaming process is that the virus fails to identify the .CO and .EX extensions as suitable target files as its own method of infection requires the presence of full COM and EXE extensions. Note that this renaming process must be undertaken in **all** directories.

The machine should then be rebooted from a clean system diskette to remove the virus from memory and the renamed files can be renamed back to their original COM and EXE extensions - the machine is then clean. It really is that simple!

By following this procedure the execution route of the virus is *completely* severed and the link by which the virus points to COM and EXE files is broken. The final virus cluster which is unallocated (i.e. an orphan cluster) can be recovered using *CHKDSK* in the normal way.

There are two caveats to the above procedure. First, if your machine uses something other than COMMAND.COM as a command interpreter this file must be replaced under clean conditions with a known clean copy. The name and location of the command interpreter is usually displayed as the *COMSPEC* variable which may be displayed by typing SET at the DOS prompt.

Second, in order to determine whether your COMMAND.COM file is infected, you should boot your machine from a clean write-protected system diskette and then copy the suspect file onto a separate floppy disk. If the file is infected, the resulting copy will only be one cluster (usually 2048 bytes) long.

It should also be remembered that as the virus infects a disk, it overwrites the last usable cluster on the disk whether it has been allocated to another file or not. Thus it is possible for one file to be corrupted by this process. Since the average user has no way of knowing exactly where particular files reside, he should use a scanner capable of exhaustively searching all bytes of all files to check whether this has happened.

The Patch

Another interesting facet of this 'now you see it, now you don't' virus, is that we can appeal to its 'conscience' by changing just 7 bytes. If these changes are introduced into the virus cluster on the disk, a subsequent reboot makes the patched code automatically disinfect any infected files until there are no more and thereby 'self-destructs' by becoming completely disconnected from system operation. Using this patch also has the effect of making the machine immune to further infections of this virus from external sources as long as the orphaned cluster at the end of the disk is not removed.

It would be irresponsible to publish exact details of the patch here since eager pimplies would be queuing up to produce version two [*DIR II-II? Ed.*]. An information sheet giving details of the patches is available from the *Virus Bulletin* to bona fide users.

The use of either the patching method or the renaming process may also show which file was responsible for introducing the virus if it was originally copied from an infected floppy. What happens is this - all infected files maintain their integrity even under infection. Disinfecting these files restores their original condition and since the originally copied file consisted only of the virus code, it will be restored along with the rest. Thus if a scanner is run after disinfection and finds a one-cluster file which contains the virus code at its beginning, this is almost certainly the one that caused the problem in the first place. Exhaustive scanning is desirable to locate the single file that may have been actually damaged by the virus code.

The Big Myth

The myth that the Bulgarian viruses are somehow 'cleverer' than other common specimens needs to be exploded once and for all - the DIR-II virus is not clever; it's ingenious but stupid, complex and totally impractical.

The Bulgarian (and other Eastern bloc) authorities should get their acts together and introduce legislation to make this childish pastime illegal. Any computer technology from these countries carries a high degree of risk. If the authorities let them play like this with software, who is to say that some of them haven't introduced malicious or mischievous code into some machine ROM chips, or even safety-critical hardware.

Less Than Twenty Four Hours Of Research

The DIR-II code has obviously been painstakingly pared down to fit into 1024 bytes, further tampering or optimisation of this virus appears to be unlikely. I suspect that probably many weeks of some individual's programming and testing went into this latest production, but however long it took him to produce, his effort was negated in less than 24 hours of research.

His brainchild is now nothing more than a curiosity.

VIRUS ANALYSES 2 & 3

Fridrik Skulason

In the Wild - Music Bug and Form

This article will examine two boot sector viruses, which are fairly common 'in the wild', although less likely to be encountered than the 'New Zealand' virus. Neither virus is interesting from a technical point of view but publication of more detailed analysis than hitherto available was prompted by an increase in reports of infections in the field. Much of the information presented here is derived from commented disassemblies provided by Dr. Andrzej Kadlof of Poland.

Music Bug

The Music Bug virus contains a couple of text strings. One is 'MusicBug v1.06 MacroSoft Corp.'; the other is '— Made in Taiwan —', and it appears to have been created there, although it has now spread all over the world. Like the Azusa virus, the Music Bug virus infected the computers of a Taiwanese producer of VGA-driver software, which then distributed infected, shrink-wrapped and write-protected diskettes to unsuspecting users.

Music Bug contains a simple self-test algorithm - the very first task performed by the virus is to compute a checksum for a 144-byte area on the boot sector, which, surprisingly is not a part of the virus code itself. If the checksum does not match the expected value the virus will overwrite the first 64 Kbytes of memory. As this area includes the virus code itself - located at 0:7C00H, this will result in a system crash.

If the checksum is correct, the virus will read the rest of the virus code, which consists of 7 additional sectors. This code is carefully designed to allow up to 4 retries if the virus gets a read error. It then creates a 4 Kilobyte 'hole' at the top of RAM, by lowering the value stored at 40:13H, and moves itself there.

The virus will then load the original boot sector and hook INT 13H - a normal practice for any boot sector virus. It next checks the current day. If no real-time clock is installed, nothing further happens. Otherwise, the current date is compared to the date of the infection of the boot sector. If it appears to have been infected at least four months previously, a flag is set, which enables the 'music' effect. The author's idea was probably to increase the virus' chances of spreading, by making it stay silent for the first four months after it infects a system.

Finally, the virus simply passes control to the original boot sector so that execution proceeds as normal.

Music Bug INT 13H Handler

Whenever INT 13H is called the virus checks when it was last called. If it was less than one second before, the virus will simply call the original INT 13H handler.

If the drive being accessed is A:, B: or the first hard disk in the system, it is checked for an existing infection. The virus reads the DOS boot sector and checks whether the word at offset 40H matches the corresponding word in the virus - this word is the checksum mentioned before. If it matches or if some error occurs, the virus will not attempt to infect the drive and instead it checks whether it should produce a sound.

The Music

Music Bug uses the system timer as a random generator to determine whether it should play a tune or not. The chance of that happening is close to 14 percent - 35/256 to be exact.

The tune it plays is a sequence of 36 notes, each of which is selected at random from a list of eight basic notes. As the random number generator is written, it may return the same number several times in a row which produces a sequence of 36 identical beeps. The virus uses a timer-controlled delay to compensate for variance in clock speed between different processors.

After playing the tune the virus proceeds with the original INT 13H request.

Music Bug Infection

Music Bug infects the DOS Boot sector, i.e. the boot sector of the active DOS partition on the hard drive. It also recognises 360 Kbyte and 1.2 Mbyte 5.25" diskettes, but will not attempt to infect 3.5" diskettes. It assumes the diskettes always have 12-bit FAT entries and hard disks use 16-bit FATs, so it might be quite destructive when this is not the case.

It searches the FAT for four consecutive empty clusters, which may result in the allocation of 16 kilobytes or even more, instead of the 4 kilobytes which are actually necessary to store the virus code. The clusters are not marked as 'bad', but instead as 'End-of-file'. Running *CHKDSK* on an infected disk will produce a message about lost clusters, and if *CHKDSK* is instructed to fix the problem, it will make the clusters available, probably resulting in the virus code being overwritten later.

There is also a small mistake in the code which computes the location of the first data sector, where the number of root directory entries is shifted right by 4 bits, multiplying it by 16, instead of 32.

Music Bug Removal

It is possible to remove the virus from infected diskettes with the *SYS* command, and then recover the allocated clusters by running *CHKDSK*, but the use of a virus-removal program is recommended - in particular when removing the virus from an infected hard disk.

The infected boot sector retains the location (head, track, sector) of the original DOS Boot Sector. The virus collects this data and points to this location after executing its own code, thus enabling the true DOS Boot Sector to execute in standard fashion. All the disinfection tool has to do is to read this information, access the original DOS boot sector and restore it to its rightful place. This process can also be undertaken using *The Norton Utilities* or *PC Tools*. Cleaning up the FAT is not necessary, as it can be done by *CHKDSK*.

The Form Virus

This boot sector virus from Switzerland makes noises, like the Music Bug virus, and infects the DOS Boot Sectors of diskettes and hard disks, but there the similarity ends.

When the Form virus is executed, it reserves 2 kilobytes at the top of RAM and reads the second half of itself from disk. The virus does not retry the read operation if it gets a read error, but simply hangs the machine. This means that booting from an infected floppy may often result in the machine hanging, because a timeout error is somewhat likely at this point. The virus will then read the original boot sector and attempt to infect the hard disk.

Hard Disk Infection

The Form virus reads the Partition Table and locates the active DOS partition. It will then read the DOS Boot Sector of that partition and check whether it is already infected or whether the sector size is something other than 512 bytes. If it is a 512 byte sector, the original DOS Boot sector is written to the last sector of the partition and the second half of the virus is stored in the preceding sector. Finally the virus overwrites the first sector of the partition (i.e. the location of the original DOS Boot sector) with the first half of itself. As the sectors used by the virus are not allocated by the virus they have a chance of being overwritten, but this will only happen if the partition fills up completely.

Activation Day?

After having infected the hard disk the virus hooks INT 13H, but if the current date is the 24th of any month, the virus also hooks INT 09H - the keyboard interrupt. The new INT 09H

handler produces a click whenever a key is pressed - a harmless, but annoying effect.

The INT 13H Handler

The Form virus only intercepts requests to read from Track 0 on drives A: and B: - in all other cases control is simply passed directly to the original INT 13H handler. This will generally result in the infection of diskettes the first time that they are accessed.

Diskettes are infected in standard fashion. The virus attempts to infect all densities of diskettes as long as the sector size is a standard 512 bytes. As in the case of hard disk infections the virus starts by verifying that the sector size is 512 bytes and that the boot sector is not already infected. It then proceeds to locate an unused cluster, marking it as 'bad' in the FAT and moving the original boot sector there, as well as the second half of the virus code.

This is all quite ordinary boot sector virus activity - indeed there is very little remarkable about the Form virus, except maybe the following text message which it contains:

```
The FORM-Virus sends greetings to everyone who's
reading this text. FORM doesn't destroy data! Don't
panic! Fuckings go to Corinne.
```

Form Removal

The simplest way to disinfect disskettes is to boot from a clean write-protected system diskette, transfer data or executables using the DOS *COPY* command and then format the diskette. It is essential that this process is done in a clean DOS environment. Do not use *DISKCOPY* as this is an image copier and will copy the infected diskette *exactly* and in its *entirity* - including the virus in logical sector 0 of the diskette.

The form virus can be removed from the active hard disk partition using a method similar to that used to remove Music Bug - simply by locating the original boot sector and writing it back to its original location.

The FAT might need slight fixing to recover the lost clusters, but that is not strictly necessary and should only be done with virus removal tools which recognise the Form virus.

Finally...

For any boot sector virus to spread and do damage it must first be *executed*. A machine will only become infected if you boot from an infected diskette. **Remind all PC users never to leave diskettes in drives for longer than necessary.**

ROM start-up code always tries to boot from the diskette drive in preference to the hard drive and if an infected diskette is present, the virus will be read into memory.

FIAT LUX

A Pervading Myth: The CMOS Virus

For the uninitiated, CMOS (the universally accepted acronym for *Complimentary Metal Oxide Semiconductor*) is a hardware feature which appears on virtually all personal computers. The particular feature of CMOS that makes it so useful is that the technology can retain information over long periods of time with only a tiny power requirement.

The CMOS feature on most PCs consists of a memory chip and associated service hardware which enables the contents of the memory to be read or written by appropriate software. The service hardware usually includes a crystal oscillator which maintains a constant 'heartbeat' while power is applied. The power requirement for this little cluster of chips comes from a battery contained in the PC and this will maintain memory and oscillator integrity for months (possibly even years).

On many PCs the battery is rechargeable and the normal machine power supply enables a trickle charge circuit when the machine is switched on, thus prolonging life even further. The amount of memory actually available in PC CMOS memory varies widely from machine to machine - on IBM ATs it is 50 bytes.

CMOS enables the machine to retain information even when it is switched off. Most machines use CMOS for two main purposes: to retain configuration information which the machine will need to recognise its various peripherals; and to keep a running time value which is converted to a clock/calendar setting. Thus when a machine is switched off, the clock/calendar keeps running (courtesy of the oscillator and associated circuitry).

Can CMOS Be Infected?

A description of CMOS is all very interesting, but its *relevance* to the virus problem is not immediately apparent. The major (hypothetical) concern is the fact that if information can survive a power-down, a clean reboot which (as hardened *VB* readers know) is essential to successful computer virus detection and disinfection, might be circumvented. Such a hypothetical 'virus' might gain control of the machine before the standard bootstrap code.

The vague feelings of uneasiness about a virus located in CMOS memory (and therefore active when you switched on regardless of how you rebooted) has persisted among uninformed users and has actually been perpetuated by certain people who should know better. So perhaps it is now appropriate to shed a little light on this subject.

Any sort of information can be stored in memory, data, program code, documentation, hardware flags and so on.

CMOS memory is no different to ordinary RAM in this respect and could certainly contain executable code in the form of a virus. However, there are additional factors which make it *extremely* unlikely.

The CMOS memory is not connected to the normal bus of a PC in the same way as normal RAM is. The memory chip can only be accessed by its controlling software and even if executable code were stored there it could not be executed *in situ*!

The contents of CMOS are actually collected into the processing area by access to two input/output (I/O) ports in the machine's standard configuration. This is similar to the access that PCs gain to the Asynchronous Serial Communications port and is limited to bringing one byte (or word) at a time into main RAM. So, just as a program file can be received byte by byte at the serial port (and may be executed later), information from CMOS memory can only be collected *serially*.

Obviously it is still possible for CMOS to contain executable code but in order for it to be executed it must first be collected (byte by byte) into main RAM and then executed! This would require a specially written 'loader' program to collect the code and then execute it. This loader program would occupy the same place in our affections as the decryption routine in other viruses - it would be obvious, detectable and (most important) would need to be *executed* in order to collect the virus. So we're back to checking for executable code stored on disk!

Some other factors which render the 'CMOS virus' a virtual impossibility are a lack of standardisation and inter-machine propagation. Across different machines there is virtually no standard for what CMOS memory *actually* contains and whereabouts it is stored. Thus any virus written to be stored in the CMOS of one machine would be likely to corrupt vital information in another.

The final nail in the coffin of the CMOS virus myth is this - *how would such a virus propagate from machine to machine*? Once it started to write its code to disk in order to attempt infecting another machine, it becomes no different to any other virus and is immediately exposed to detection and eradication.

To date, no virus has been reported that attempts to use the CMOS for code storage; even if a working sample were developed it would strictly be of academic interest. For similar reasons, the less well known but equally fatuous myth of the 'Printer RAM virus', should also be consigned to the realm of fairy tales.

The next time some bewildered person tells you about the 'CMOS virus', smile at them gently and be kind. After all, it certainly isn't impossible but then neither is a trans-polar circuit of the globe by bicycle - but would you bet on it?

ON COMPUSERVE

Troublesome Concubines in the Anti-Virus Harem

In the course of testing virus scanners, *VB's* evaluators have become increasingly aware of incompatibilities and inconsistencies when more than one anti-virus package is installed on a PC's hard disk.

At the moment this problem is prevalent in the case of *Central Point Anti-Virus*. Messages on *Central Point's* forum on *Compuserve* include a number of complaints about false positives. Competitive scanners - including *VISCAN*, *DOCTOR* (part of *Virus Buster*, see pp. 18-21) and *Vi-Spy* - to mention just three - are detecting the Flip virus in four of *Central Point's* anti-virus programs, namely: VSAFE.COM, VSAFE.SYS, VWATCH.COM, VWATCH.SYS.

The reason for these false positives is that all of the *Central Point* programs mentioned above contain the decryption code for the Flip virus - the same code used by scanners to detect *real* infections caused by this virus. This is indeed the *only* safe code sequence to use in this case.

With PC Support and security managers using scanners from diverse sources (many major PC users subscribe to as many as five dissimilar scanners), the manufacturers of these products should strive towards harmony rather than discord. QA should include efforts to ensure that products do not clash i.e. false-positive alerts should be tested for in as many competitors' products as possible. (One company, *S&S* actually subscribes to its competitor's products to undertake such testing. *Central Point* itself is known to subscribe to the *VIS Utilities* which contain the aforementioned *VISCAN*.)

Furthermore, no product should contain virus identification patterns in plain code (as *Central Point's* product clearly does) - search data should be encrypted.

In *Central Point's* case, the reason for this company adopting the same or a very similar encryption algorithm to that found in the Flip virus is unclear - one theory is that it is being used as a means to protect certain files. Whatever the reason, the practice should be discontinued.

It is brave (or foolhardy) to ignore alarms on the supposition that they are false. Suppose files which produce a string of suspected false positive indications actually are infected by the *real* virus?

Manufacturers should do their best to ensure that their products co-habit in perfect harmony. In the anti-virus harem which users maintain these days, offending scanners, like troublesome concubines, are likely to be evicted and replaced with more mild-mannered counterparts.

SCANNER TACTICS

Living Together - Without False Alarms!

One of the most serious problems which has arisen as a direct result of virus-specific detection methods is the nuisance and loss of time caused by 'false positive' indications. These can take several forms and all of them can cause significant loss of time and money, especially to large corporate PC users.

Before looking at the various facets of the problem let us first define what we mean by 'false positive':

Whether using a scanner, a resident monitor or some form of file integrity checking, the positive identification of virus code or virus-like activity is the name of the game. Unfortunately, the information used to identify the virus is always incomplete - unless the virus is non-encrypting and a complete copy of its code is used for comparison! Thus, there will be times when even the best packages indicate the presence of a virus when in fact what they have found is innocent code which matches their identification criteria. These inaccurate indications of infection are known as 'false positives'.

A common reason for false positive indicators is the use of insufficiently specific search patterns. Another reason is when two packages use similar patterns, at least one of which is not stored (either on file or in memory) in encrypted form. A third problem occurs on memory checking programs when a second run is made after finding a virus in a disk file.

With integrity checking programs, slightly different problems occur. The user must be aware of all of these problems in order to optimise his response to the possibility of false positives. Some of the most common problems can best be explained by citing examples.

Pattern Clashes In Files

At least one anti-virus package stores its search patterns in unencrypted form on disk. This has caused false positive indications from other packages, particularly where the identification of simple encrypting viruses is concerned. With such viruses, the decryption stub used for primary identification is drastically reduced and the chance of duplicate patterns being extracted by different vendors is thereby increased. Thus package A might identify a virus in package B simply because package B did not take the elementary precaution of encrypting its signatures before storing them on disk.

'Unique' Signatures

With most viruses, the extraction of a sufficiently specific recognition pattern is relatively straightforward. With some of the encrypting and randomising samples and certain viruses written in high level languages things become a little more difficult. It is here that vendors may trade off overall accuracy and security in order to avoid what they consider an unacceptable level of false positive reports. The technology that each vendor uses may thereby allow a greater or lesser risk of clashing with other packages or producing erroneous reports. Whether it is better to sacrifice security to avoid an occasional false positive is entirely a decision for the user.

False Positives in Memory

In these instances, both package A and B might have their recognition information stored securely on disk but when they are loaded into memory the patterns must be decrypted before use. As long as the package conscientiously 'cleans up' before it exits all will be well. However, there are packages which do not complete this cleanup process and in this case, running package B immediately after package A (from the same boot sequence) may produce a false positive in memory when B finds a discarded but recognisable pattern belonging to A.

Another false positive in memory can be caused as the result of a scanner finding a real virus in a file. When scanning files, their contents are read into memory through the disk buffers (and any cache system that may be present). Thus if a file is read and found to contain a recognisable chunk of a virus, the disk or cache buffers may still contain virus code (albeit not in active form). An immediate re-scan may well detect this in memory and display a false indication that the virus is 'resident' (i.e. operative) in memory.

Generic Integrity Checking Programs

If Program A collects and stores an integrity profile of specific executable files, and then Program B is run to do roughly the same thing, as long as both programs have made no change to the files under observation there will be no clash.

However, there are (believe it or not) commercial programs which will modify a file's profile to indicate that it has been checked. Some vendors even add integrity information to the end of the file which can be referenced (only by their software of course) as a future check on integrity. Temporarily disregarding the fact that such methods are useless against stealth viruses, any modification of file content, date/time or attributes constitutes a violation of the file's integrity which any reasonable integrity checking program will detect!

If only Program A changes files in this way then as long as Program A is run first, there should be no clash. However, consider if both A and B use different versions of this method - the first time each program check is run there will be no problem reported. Subsequent checks will report integrity violations only if their own modification is no longer visible because the other program has added its own modification and covered it up! Thus running B after A or A after B will have the poor user running around in circles. There are even commercial checking programs which manipulate the invalid values of the date and time fields (just like some viruses)!

Guidelines

From the user's point of view, it makes good sense to use a number of different anti-virus packages in order that they may each confirm the findings of the other. One of the major reasons for such an approach is to limit the problems posed by false positive indications. Unfortunately, the careless or self-centred approach of many vendors means that their packages may actually *cause* false positives in other packages.

To avoid the confusion and inconvenience caused by false positives there are certain guidelines:

- Use several scanners from dissimilar sources. The more search data that is available the better - this increases the likelihood of detecting genuine infections while providing a means to diagnose suspected false alarms. No single virus-scanner can provide 100 percent protection!
- Always run scanning and integrity checks on a freshly booted system. Boot from power off between subsequent checks.
- Remember that false positives can result from scanning with anti-virus software from dissimilar sources. Either remove such software from the disk under inspection or ignore any warnings limited solely to it.
- It would be highly unusual to find just a single occurrence of a parasitic (program infecting) virus on a working hard disk. Once a virus is invoked its main purpose is to spread, so you would expect to find several occurrences within a working environment. Floppy disks on the other hand, could quite easily contain just a single infection.
- Avoid the use of integrity checking programs which add *modifications* to actual file profiles. These are often advertised as providing a checking system which will travel with the file but they are worse than useless when used in conjunction with other integrity checking software that completes a reliable check.
- When a virus infection has been indicated, you should attempt to verify its existence via other methods (integrity checks versus scanning methods etc.). These include checking along possible infection paths and testing with other software.

You should also remember to check with the vendor of the package - if there are any false positive problems, they are likely to know about them and be able to put your mind at rest. In any case they should be informed so that they can make efforts to correct the problem (assuming it is a problem they *can* address).

Finally, if a package continues to produce an unacceptable number of false positive indications it should be discarded. The whole point of anti-virus software is to save time and worry - not generate it!

PRODUCT REVIEW

Mark Hamilton

Virus Buster

Virus Buster is an Australian package from *Leprechaun Software International*. The package consists of a 320-page perfect-bound paperback manual, two 360 Kbyte and one 720 Kbyte diskettes in video-cassette type packaging.

The software consists of three main programs, *BUSTER*, *WATCHDOG* and *DOCTOR* and a number of complementary files, three of which are simply included to install the package on a hard disk or floppies. The installation process scans memory and the destination disk before copying and installing the constituent parts of the package.

Installation

The package refused to install onto the hard disk of my Apricot 486, but it installed without any problem to the hard drive of my Compaq DeskPro 386/16. An inauspicious start to the review.

Alternatively, you can simply copy the 22 files into a sub-directory, and using the instructions in the documentation, configure the software to suit your preferences.

Buster

BUSTER is a checksumming program which detect changes in files. The first time it is run, it creates an encrypted data file, BUSTER37.DAT, which contains details of the file's path name, date, size, header and checksum. This information is used by *BUSTER* for subsequent checks. By default, *BUSTER*, checks all the normal executable file types, but you are able to add to or remove from the list of these to suit your personal preferences.

When *BUSTER* is re-run, any changes to the recorded details to any of the files (or disk's system area) are reported in a pop-up window. You can add details of any new program; change *BUSTER's* record for a file; rename the file; wipe the file; or, generically restore the file to its former self. *BUSTER* is intelligent enough to know whether it can restore a particular change successfully and disables this option if it can't.

On my Compaq Deskpro *BUSTER* completed its checks on 419 executable files (14 Mbytes) in just 2 minutes, which works out at 118 Kbytes per second. It took less than one second longer to create its database initially.

Like all generic checkers, it tripped-up over self-modifying programs - such as some of the shareware text editors which

save their configuration information within themselves. In such cases, *BUSTER* reports changes to both the date and contents of the file. Provided you are aware of which programs exhibit this behaviour, you can safely update your chosen generic checker's database with a deal of confidence.

Watchdog

WATCHDOG is a dual-function memory-resident (TSR) monitor program. It warns of attempts to modify executable files, including those of the operating system, and to sensitive disk areas, such as the Master Boot Sector. (However, under working conditions it totally failed to prevent the infection of the DOS Boot Sector of a *WATCHDOG* protected machine with the Anthrax virus.)

WATCHDOG will conflict with such innocent programs such as *FORMAT* or any aforementioned self-modifying program. There is a companion program, *PROTECT*, which bulk-immunises program files, by placing a checksum in the free-space beyond the end of file.

When you run a program, with *WATCHDOG* resident, *WATCHDOG* calculates a checksum for the program and it looks for this marker. If it finds it and it agrees with the calculated value, the program is executed without further interference. If the checksums do not match, or if there is no checksum marker, a pop-up window appears and gives you the option of not running the program, turning off *WATCHDOG*, or continuing.

Doctor indicating a false positive in *Central Point Anti-Virus*. A number of virus scanners have tripped over *CPAV* - the fault lies with *Central Point* not *Leprechaun*! (See text and article, page 18.)

Possible Drawbacks

There are three disadvantages with this strategy. As with *BUSTER*, there is a problem with self-modifying programs. However, the way in which the *WATCHDOG* markers are stored may prove a more serious problem.

Both *PROTECT* and *WATCHDOG* append the 4-byte marker to the file and place it in the slack space at the end of most files - the file's directory entry is not altered to account for the extra bytes. Therefore if a file is moved (by a disk defragmentation utility such as *PC Tools' COMPRESS* or *Symantec's SPEED DISK* utilities, for instance) or just copied, the marker is lost.

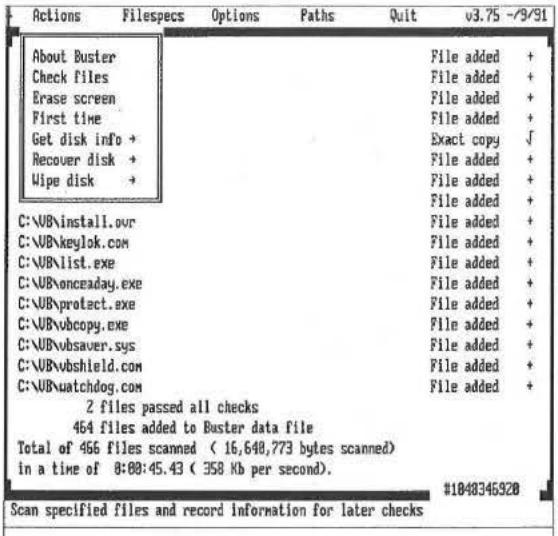
Furthermore, it is becoming increasingly common for program files to be constructed such that they fill disk clusters exactly. This is done to counter the threat of infection by viruses such as Nomenklatura which reside in slack space. These padded files are not protectable by *PROTECT/WATCHDOG* since there is no slack space. On most disk systems, 2 or 4 kilobytes is the normal cluster size but DR-DOS 6's *SPEEDSTOR* disk compression system allocates disk space in multiples of 512 byte sectors - the generic *Addstor* and *Stacker* products have similar characteristics. This reduces the number of files that can potentially be protected since, statistically, more files will fill 512 bytes than its multiples, 2048 or 4096.

Other Components

FIDO is a *Windows 3* specific program which serves as an interface to *WATCHDOG* while working in the *Windows* environment. To install *FIDO*, you simply include a line similar to 'LOAD=FIDO.EXE' in the [*Windows*] section of your WIN.INI file.

VBCOPY checks files for viruses while they are being copied and provides most of the functionality of the DOS *XCOPY* utility. *DISKLOK* prevents the hard disk of a PC being accessed after the PC is booted from a floppy and can automatically replace Master and DOS Boot sectors thus providing transparent protection against common boot sector viruses. *KEYLOK* locks the keyboard during periods of absence from a running PC. Both these utilities are included to provide a medium level of security. All the utilities can be configured using the *INSTALL* program which can be re-run, at will, as a configuration utility. This, like *PROTECT*, *BUSTER*, and *DOCTOR* has a full screen, menu-driven interface with on-line help and is a genuine pleasure to use.

Leprechaun also provides a small device driver, *VBSAVER*, of around 200 bytes which is designed to circumvent infection by stealth viruses which manage to avoid detection by most disk scanners.



Buster, Leprechaun Software's checksumming program indicating the installation of new files which have not yet been checksummed. The program searches for unknown files. Notice the 'EXACT COPY' diagnosis indicating a checksum match.

Doctor

In common with the other elements of this package that feature a full screen, menu-driven environment, *DOCTOR*, the scanner, can be run in batch mode. In full screen mode, it shares a really annoying feature with *BUSTER* and *PROTECT*. At the bottom right of the screen a continously repeating sequence of product name, producer, telephone number (in Australia) and two user-defined messages are displayed, flashing, changing colour and generally causing annoyance. It might be eye-catching, but I sorely wished I could disable the wretched thing.

The interface is clean, easy to use, (although not too responsive to a mouse click) and well thought-out. I was pleased to discover that, although a text-mode application, the all-too familiar mouse cursor blob had been replaced by a yellow cross on a suitable contrasting background (red or brown).

DOCTOR's has a respectable accuracy percentage, detecting 340 out of the 363 file infections of my current test-suite (refer to *Virus Bulletin*, September 1991, page 18 for full details of the test-set and test conditions employed).

Like so many scanners, *DOCTOR* fell foul of the notorious *Central Point Anti-Virus' VSAFE* and *VWATCH* program sets. There have been a number of messages in *Central Point's* support forum on *Compuserve* from customers complaining that other scanners were tripping over these files, but *Central Point* has done nothing so far to resolve this major problem (for more information see article on page 18).

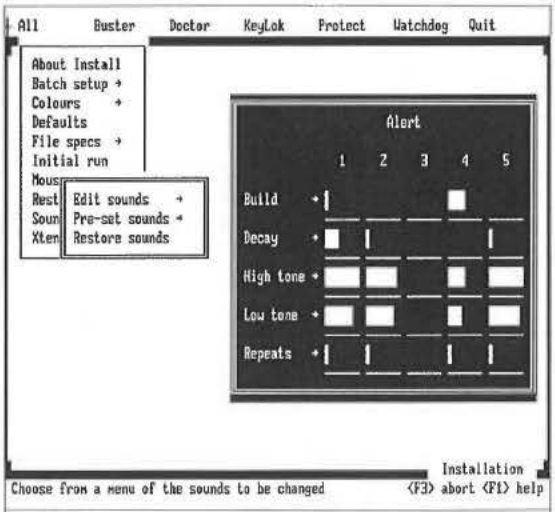
DOCTOR fared quite well with the encrypted virus test which was introduced last month. Apart from missing three of the 20 COM file Flip infections (it found all 20 Flip EXE infections) and one of the 50 V2P6 infections, it detected all the rest. In fact this product fared better than most as it found all the multiple infections of the encrypting viruses known to be at large, including Tequila and Spanish Telecom 1 and 2.

It had absolutely no problem detecting the various boot sector infections and scored a straight 100 percent detection rate.

In its Turbo mode, *DOCTOR* proved to be one of the fastest scanners I have come across. However, in its secure mode, it took over ten minutes to scan all the files on the Compaq's hard disk.

In checking just the 421 executable files (approximately half the total number of files) in its turbo mode, it sped along and completed its checks in a shade over two minutes. Its secure floppy test was completed in just under 11 seconds and in its turbo mode, an astonishing two seconds!

When *DOCTOR* discovers what it thinks is a virus infected file, it displays a large warning red box which provides details of the file and the suspected virus, with a menu box beneath. This gives you the option of changing the file's name, removing the virus, ignoring the threat or wiping the file clean. One of the options dictates how wiping is achieved -



The virus alert includes a broadcast from the speaker - in default mode this sounds like an American police car. Tastes vary and the men from Oz have provided a sound editor. Very thoughtful - but is it *really* necessary?

that is to say whether the file is wiped according to *Australian Government Rules* (whatever they might be - a form of rugby played in Canberra?) or not. [The *Government Rules* are explained on pages 119 and 161 of the manual. Ed.]

Conclusions

Virus Buster has obviously been designed and implemented as a result of thorough technical research and development. It is to be hoped that the men in cork-brimmed hats continue their good work in producing a steady stream of regular updates of this quality. This package is a rising star and looks set to figure prominently in *VB's* ongoing comparative reviews. It isn't quite what tinnies of *XXXX* are to sheep shearers (essential), but it's close.

VIRUS BUSTER

Product and Version	Virus Buster 3.75
Scanning Speeds [1]	
Test 1 (i) Hard Disk - Turbo	2m 2s
Test 1 (ii) Hard Disk - Secure	10m 57s
Test 2 (i) Floppy Disk - Turbo	2s
Test 2 (ii) Floppy Disk - Secure	38s
Scanner Accuracy [2]	
Parasitic Viruses - Turbo	340/363
Parasitic Viruses - Secure	340/363
Boot Sector Viruses - Turbo	8/8
Accuracy Perecentage - Turbo	93.8%
Accuracy Percentage - Secure	93.8%
Stamina Test - Encrypting Viruses [3]	
Multiple Test: Flip	Fail
Multiple Test: Suomi	Pass
Multiple Test: Tequila	Pass
Multiple Test: Spanish Telecom	Pass
Multiple Test: Group 2	Pass
Multiple Test: Group 3	Fail

[1] This speed test is outlined in the test protocol described in *VB*, April 1991, pp 6-7.

[2] The test-set is outlined in *VB*, September 1991, p. 18.

[3] This test to determine a scanner's ability to detect encrypted viruses was first conducted in *VB*, October 1991, pp. 7-11.

Technical Details

Test Conditions: The testing for this review was conducted on two PCs. The first, a Compaq DeskPro 386/16 running under DR-DOS 6 was used for the speed tests. There are 27.5 megabytes in 920 files of which 421 files are binary executables and they occupy 14.6 megabytes.

For the floppy read speed tests, the 360 Kbyte Setup disk for *Microsoft* C 5.1 was used. This contains a total of 12 files requiring 354,804 bytes, of which four (238, 913 bytes in volume) are executable.

The virus identification testing was conducted on an Apricot 486/25 which houses the test library. For specific details of the viruses used in all the tests, please refer to *VB*, *VB*, September 1991 p. 18.

Product: *Virus Buster*

Developer: *Leprechaun Software Pty Ltd*, PO Box 184, Holland Park, Queensland 4121, Australia.
Tel +61 7 343 8866, Fax +61 7 343 8733, BBS +61 7 849 8727.

European Office: *VB Software*, Church Street, Cappawhite, County Tipperary, Eire, Tel +353 6275404.

USA: Toll-free number 800 521 8849.

Availability: IBM, PC, AT, PS2 or compatible running MS-DOS version 2.1 or greater. Hardware requirements include 256 Kbytes of memory and 700 Kilobytes of hard disk space.

Version Evaluated: 3.75

Serial Number: 1048346920

Price: £95.00. (Plus £50.00 for yearly maintenance including unlimited access to bulletin board update service and technical support).

Editor's Note

The hard disk speed results for *Virus Buster* shown in the table opposite are <u>not</u> directly comparable to those shown in the comparative scanner reviews which *VB* regularly publishes. The file configuration on the test machine has changed since the last comparative review - details of the file configuration used for the tests on *Virus Buster* are shown above. Floppy disk scan speeds and accuracy percentages shown in the table opposite remain directly comparable to those reported in *VB*, September 1991, pp. 16-17. All *comparative* reviews are conducted using *exactly* the same processor and file configurations. In future, *standalone* reviews such as this one of *Virus Buster*, results will become comparable due to the use of a static set of executable files against which timing tests will be run.

END-NOTES & NEWS

S&S International has reported **a clash** between its *FindVirus* scanner and the DOS utility SHARE.COM. If SHARE is executed in advance of *FindVirus* and if diskettes are subsequently examined in the drive from which the scanner was launched, *FindVirus* fails to examine the diskettes correctly and reports zero files on disk regardless of actual content. *S&S* report that the problem has now been fixed.

The **complete proceedings** of the *First International Virus Bulletin Conference*, St Helier, Jersey 1991 are now available at a cost of £50.00 (excluding postage and packing). The proceedings include slide sets and transcriptions. Contact Petra Duffield, *Virus Bulletin Ltd.*

The *National Computer Security Association* is hosting a two-day **Anti-Virus Product Developers Conference** on November 25-26 1991 in Washington DC. Representatives from *Symantec*, *McAfee Associates*, *S&S International*, *Novell* and *Certus* are included in the line-up of speakers. An ambitious programme aims to tackle ethics, certification, virus migration and epidemiology, networks, virus classification and a host of other topics. Information from *NCSA*, AVPD Conference, 227 West Main Street, Mechanicsburg, PA 17055, USA. Tel 717 258 1816.

Writeguard is hailed as a '**write-protect tab for your hard disk**'. This hard card lives between the hard disk and the disk controller and switches are used to write-protect specific tracks of the hard disk. The device aims to prevent boot sector viruses. The product supports ST5096, ST412, SCSI and IDE hard disk controllers. Information from *Marscot Ltd*, UK. Tel 0383 416089.

S&S International has launched ***PC Armour***, a memory-resident program with a 2 Kbyte footprint which can be configured to prevent unauthorised hard disk access, unauthorised program execution and to enforce password protected access control. *PC Armour* supports IBM PC compatibles and networks. Program authorisation is not yet available under *Windows*. Information from *S&S*, UK. Tel 0442 877877.

ViruGuard is a hard card which aims to provide 'complete virus protection'. The card is said to trap all modifications to executable files, detect interrupt changes, prevent disk formats, report data corruption, intercept direct BIOS calls and illegal DOS calls. Information from *Ports of Trade*, 6 Alcis Street, Newlands, Cape Town 7700, South Africa. Tel 686 8215, Fax 685 1807.

Sophos has announced the launch of ***D-FENCE***, a memory-resident utility which prevents the use of unauthorised disks. The program is designed to ensure that only 'authorised' disks can be used on company machines. According to Sophos, *D-FENCE* is 'the conceptual equivalent of equipping all PCs with 4 inch drives. Nobody can use floppy disks until they have been virus-checked and converted to *D-FENCE* format.' Information from *Sophos*, UK. Tel 0235 559933.

Digital Equipment Corporation and *Computer Security Consultants Inc.*, will hold a series of **seminars on disaster recovery**. The seminars take place on 18-20th November (Los Angeles), 10-12th December (Vancouver), 11-13th February (Orlando) and 27-29th May (San Diego). For information contact *CSCI, Inc.*, USA. Tel 800 925 CSCI.

VIRUS BULLETIN

Subscription price for 1 year (12 issues) including first-class/airmail delivery:

UK £195, Europe £225, International £245 (US$395)

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, 21 The Quadrant, Abingdon Science Park, Abingdon, OX14 3YS, England

Tel (0235) 555139, International Tel (+44) 235 555139
Fax (0235) 559935, International Fax (+44) 235 559935

US subscriptions only:

June Jordan, *Virus Bulletin*, 590 Danbury Road, Ridgefield, CT 06877, USA
Tel 203 431 8720, Fax 203 431 8165