Fortinet-1018

ISSN 0956-9979 NOVEMBER 1993

VIRUS BULLETIN

THE INTERNATIONAL PUBLICATION ON COMPUTER VIRUS PREVENTION, RECOGNITION AND REMOVAL

IN THIS ISSUE:

• **Wilding on Viruses.** Edward Wilding, *VB's* founding editor, presents his own views on the war against viruses and describes some of his experiences along the way.

• **Polymorphism.** The first part of a new series on viral developments: How do polymorphic viruses work, and how can we detect them?

• ***Norton Anti-Virus 3.0.*** The third revision, and *VB's* third review, of *Symantec's* scanner - a definite improvement, albeit with limitations.

CONTENTS

EDITORIAL

Risky Business

One example of a secure anti-virus policy is a strictly enforced regime of clean-booting, checksumming and the scanning of all disks as they enter and leave the company. This procedure is tried and tested and has been shown to work. Taking this a step further, hardware write-protection could be implemented, along with access control mechanisms and disk validation software. These additional measures undoubtedly provide a very high level of protection, but when taken to such extremes are, for most sites, totally impractical.

In the real world, such measures are almost always unnecessary, and defence strategies tend to fall between a 'maximum overkill' solution and no protection at all. The hardest part of designing an anti-virus policy is deciding what precautions are needed.

"You know the threat, you can evaluate the risks, and you can balance them against the cost of your countermeasures."

One danger when planning security measures is that it is easy to become too focused on one particular issue. Viruses are just one of the many dangers which businesses face, and should be treated as such. Therefore it seems logical to apply the usual risk assesment techniques when evaluating the best way to protect one's company.

Such risk analysis relies on being able to gauge the relative reliability of different countermeasures. For example, disinfecting files is inherently less reliable than replacing them from either the original master diskettes or a trusted backup - but by how much? Is the loss of security justifiable?

Consider the security requirements for a computer used to create the master copy of software which will be shipped to a customer. The right way to protect this machine would be clean booting and examining the machine with a checksummer - not a very usable solution, but a very secure one. This approach is needed because of the consequences of a mishap: 1 million infected copies of*Word for Windows*, for example, would doubtless cause a number of red faces.

However, when operating in a low-security (and often high threat) environment, little will be lost by using less reliable prophylactics such as TSRs. Does a department filled with machines used for word-processing need to be clean-booted and scanned once a day? Probably not.

The preceding discussion may seem obvious but it is very easy for a company to adopt an enterprise-wide solution - the 'one size fits all' approach to computer security. A policy set up in this manner will either be woefully inadequate for mission-critical systems or far too rigorous for most users. Compare this with a made-to-measure solution, where the level of security depends on how much each user actually needs. Both solutions will defend againt viral attack, but only one has the benefit of leaving the user as much freedom as possible. Obviously individually tailoring each machine's protection may not be practical: the level of the granularity of the policy will depend upon the type of company and the number of machines involved.

The bottom line in virus detection is price. If the actual cost of the countermeasures put in place exceeds the cost of the hazard which they replace then clearly the protection is not needed. Unfortunately the money spent on anti-virus software is only part of the expenditure necessary. Once the software is purchased, there are many operational costs in maintaining it.

If a policy requires that a scanner is updated on 5,000 PCs once a month, the time taken to carry this out must be entered into the equation. The next hidden cost is that of the man-hours taken up in training users on safe computing practices. The list could go on. If an anti-virus policy is not cost-effective, it is quite simply wrong.

All this is enough to make even the most stalwart IT Manager start muttering about camels and the eyes of needles, but this does not need to be the case. Where complex theoretical analysis cannot solve the problem, common sense usually goes a long way. You know the threat, you can evaluate the risks, and you can balance them against the cost of your countermeasures. The most secure solution is not always the right one.

NEWS

Variations on a Theme

Virus researchers worldwide have recently been given an early Christmas present from the computer underground: 250 new viruses.

The new virus collection which is being circulated among anti-virus vendors was sent to a number of different researchers. None of the viruses in the collection are completely new *per se*, but each is a very minor variant of an already existing virus.

The viruses all appear to have been modified in an attempt to evade detection by *McAfee's Scan*, and the changes made are sufficiently small that many other virus scanners are capable of detecting these variants. This follows the alleged publication of a number of the search strings used by the *McAfee* product on some virus exchange BBSs.

The virus collection has a number of features which make it somewhat more intriguing than most unsolicited collections.

Firstly, all the variants are based on very old viruses - typically two to three years old. Secondly, the naming convention used by the author appears to be completely arbitrary, almost as if the author had picked random words out of a dictionary.

Finally (and most worryingly), the file was received with the name PART1.ZIP, implying this is only the first instalment. With many scanners already bursting at the seams, it will be a test for research departments to keep up with this rate of flow if this becomes a monthly occurrence.

Nevertheless, users of current anti-virus products should not be alarmed by this news, as at this time none of these viruses are known to be in the wild. A full analysis of the collection will be published next month▮

More 2600 Mayhem

The latest edition of the quarterly hacker magazine,*2600*, contains more advice for the would-be virus author. Masquerading as advice on how to armour a virus in order to protect 'your virus from evil detectors', there is a one-page article on basic encryption. Readers should note that*2600's* code presents little threat to present scanner technology - any new virus will evade a virus scanner.

The current edition also contains a transcription of the report which the *2600* editor, Emmanuel Goldstein, submitted to the House Subcommittee on Telecommunications and Finance last June. It puts forth the traditional hacker argument that hacking in itself is not a criminal offence, even though the information and experience gained could easily be used for criminal purposes▮

Virus Prevalence Table - September 1993		
Virus	Incidents	(%) Reports
Form	21	42.9%
Spanish Telecom	5	11.6%
Flip	3	7.0%
Tequila	3	7.0%
Cascade	2	4.7%
Joshi	2	4.7%
V-Sign	2	4.7%
BFD-451	1	2.3%
DIR-II	1	2.3%
Exebug-1	1	2.3%
Helloween	1	2.3%
Italian	1	2.3%
New Zealand 2	1	2.3%
NoInt	1	2.3%
Quox	1	2.3%
Slow	1	2.3%
Vacsina	1	2.3%
Yankee	1	2.3%
Total	49	100.0%

Shattered Glass, Part III

According to a recent report by *Central Point*, a new *Windows* virus has been released. The virus, internally named as Cyber Riot, was sent to *Central Point* by a group who have named themselves 'The Chicago 7'.

According to the letter sent by the group, over 15,000 PCs are infected with the virus, though there is no evidence to corroborate this fact.

When the virus is executed, it first infects the file KRNL386.EXE. When this file is next run (ie next time *Windows* is started), every *Windows* program executed will be infected. The *Windows* Kernel file is infected in a different way to other files: since it behaves as a DLL to *Windows* calls, the virus can 'hook' all WinExec system calls.

Cyber Riot contains a malicious trigger routine: if *Windows* is loaded between April 29th and May 1st, the virus will attempt to destroy all data stored on the fixed disk. The trigger dates correspond to the dates of the Rodney King LA Riots, hence the name of the virus. The main weakness of the virus is that it needs to infect KRNL386.EXE in order to function, limiting it to systems running *Windows 3.1*. According to the virus authors, the Cyber Riot's source code will shortly be distributed among the computer underground, making it highly likely that variants will be encountered▮

IBM PC VIRUSES (UPDATE)

Updates and amendments to the *Virus Bulletin Table of Known IBM PC Viruses* as of 25th October 1993. Each entry consists of the virus' name, its aliases (if any) and the virus type. This is followed by a short description (if available) and a 24-byte hexadecimal search pattern to detect the presence of the virus with a disk utility or preferably a dedicated scanner which contains a user-updatable pattern library.

Type Codes

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

Black_Jec.232 **EN:** This 232 byte variant does not work. The reason is simple - it infects EXE files, as if they were COM files, and infected programs do not execute properly. Detected with the Bljec pattern.

Black_Jec.235, Black_Jec.284 CN: These two variants (235 and 284 byte long) are detected with the Bljec pattern.

Cascade.691 **CR:** Despite the fact that this virus is less than half the size of any other Cascade variant, there is no doubt that it belongs to that family. It has been stripped down, with encryption and unnecessary functionality removed. Detected with the Jojo pattern.

Cascade.1701.L **CR:** The encryption routine has been modified in this variant, presumably to avoid detection. Apart from this, it is unremarkable.

```
Cascade.1701.L  F607 0174 168D 5F23 BD82 0631 1F31 2F43 4D75 F8EB 0658 7472
```

Comasp.633 **CN:** Related to the Comasp.472 virus, which was originally reported under the name V472.

```
Comasp.633      01D6 31DB 8EC3 BB84 0026 8B0F 890C 890D 4646 4747 4343 268B
```

Digger.1512 **CEN:** Similar to the earlier variant, but 1512 bytes long.

```
Digger.1512     BB04 0051 B104 2ED2 0159 43E2 F65B 3D9B 1B74 08E9 80FB B89B
```

Dos 7 **CN:** Three small viruses (342, 376 and 419 bytes) which contain the text 'MSDOS 7 (C)1993 ANARKICK SYSTEMS DOS 6 Antivirus sucks. It missed this one!'

```
Dos 7.342       8B04 7216 3B06 0001 740D 8B44 023D 1560 740D EB4A EB5E 90B4
Dos 7.376       8B04 72C2 3B06 0001 74B9 8B44 023D 1560 7402 EB3F 5756 BE23
Dos 7.419       8B04 72A6 3B06 0001 749D 8B44 023D 1560 7402 EB3F 5756 BE4D
```

Fred **CR:** A 657 byte virus. Awaiting analysis.

```
Fred            80FC FF74 1A80 FC4E 740A 80FC 4F74 05EA ???? ???? E85C 00E8
```

Hates **CN:** A 213 byte virus that does nothing of interest other than occasionally displaying the message 'Jesus Hates You'.

```
Hates           B42C CD21 80FA 0A72 08BA 8000 B41A CD21 C3B8 0011 BB00 0EB9
```

Helloween **CER:** Several new variants have been reported recently. Two of them (1227 and 1447 bytes) are detected with the Helloween pattern, two other variants (1839 and 1888 bytes) are detected with the Helloween.1182 pattern, but the last variant (2470 bytes long) requires a new search pattern.

```
Helloween.2470  B440 EB02 B43F E816 0072 022B C1C3 33C9 33D2 B802 42EB 0890
```

Honey **CN:** The name of this 666 byte virus is derived from the text string 'Honey, I'm home . . .'.

```
Honey           80BC 3D01 E974 09B4 4FE9 76FF B400 CD21 B802 428B 9C3B 0133
```

Jerusalem.2223 **CER:** This 2223/2225 (EXE/COM) byte variant is detected with the Acad-2576 pattern.

Lockjaw.Black_Knight, Lockjaw.808 P: Two new variants of this companion virus, 520 and 808 bytes long.

```
Black Knight    9C06 1E50 5352 3D00 4B75 03E8 0E00 5A5B 581F 079D 2EFF 2E08
Lockjaw.808     9C06 1E50 5352 3D00 4B75 03E8 0E00 5A5B 581F 079D 2EFF 2E28
```

Loren **CER:** This 1387 byte virus is one of the few viruses which can be considered a 'real-world' problem.

```
Loren           8907 4343 E2F5 58C3 502E 8B86 CE05 2E89 86DB 0558 C30E 1F8E
```

Murphy.Woodstock **ER:** A 1219 byte variant, detected with the Murphy_2 pattern.

Nina.B **CR:** Very similar to the original Nina virus, and detected with the same pattern.

OK **CN:** A 778 byte Russian virus. Awaiting analysis.

```
OK              2E88 44FF E2F7 5E59 C306 B833 35CD 2126 C607 CF07 C3B4 19CD
```

Phx **CER:** A 965 byte virus. Awaiting analysis.

```
Phx             BA1A CA80 FC4B 747D 3D02 3D74 433D 74B9 7436 80FC 4074 05EA
```

Pixel.343 **CN:** Detected with the Pixel.277 pattern.

Pixel.846 **CN:** Very similar to the original 'Buy Pixel' variant, but one byte shorter in length. Detected with the Amstrad pattern.

PS-MPC: As expected, new PS-MPC-generated viruses arrive regularly. Most of them are encrypted, and detected by any scanner that can identify the standard PS-MPC encryption method. The encrypted variants include: Flex (491, CR), 513 (513, EN), Sorlec.535 (535, EN), Skeleton (552, CEN), Sorlec.553 (553, ER), 564 (564, CEN), 570 (570, CEN), Alien.571 (571, CER), 574 (574, CER) 616 (616, CEN), Shock (620, CER), Alien.625 (625, CER), Soup (645, EN), Arcv-3.657 (657, CN), Dos3 (661, CEN) Z10.662 (662, CEN), Z10.704 (704, EN), Ecu (711, ER), Napoleon (729, CEN), Arcv-9.745 (745, CN), Toys (773, CEN), Crumble (778, CEN), Arcv-10 (827, CER), Nirvana (835, EN), Grease (856, EN) and Payraise.897 (897, CER). In addition, several non-encrypted variants have also appeared, but they can be easily detected with a search pattern. They include 331 (331,CN), 349 (349, EN), T-Rex (410, CN), Abx.420 (420, CN), Iron_hoof.459 (459, EN) Iron_hoof.462 (462, EN), 478 (478, CEN), Nuke5 (478, CER), 481 (481, CEN) Page (570, CEN), 597 (597, CEN), Abx.1341 (1341, CN), Abx.1520 (1520, CN) and Abx.2010 (2010, CN).

```
PS-MPC Gen1     A5A4 C686 ???? ??B4 1A8D 96?? ??CD 21B4 47B2 008D B6?? 02CD
PS-MPC Gen2     A5A5 A5A5 C686 ???? ??B4 1A8D 96?? ??CD 21B4 47B2 008D B6
Deranged        A5A5 A5A5 8D96 E302 E838 018D B6A3 02B4 47B2 00CD 218D 9670
PS 349          A5A5 A5A5 B41A 8D96 5D02 CD21 8D96 9401 B44E B907 00CD 2172
PS 481          A5A5 A5A5 C686 5003 008D 96D5 02E8 6B00 8D96 DB02 E864 0080
PS 478          A5A5 A5A5 C686 0C03 02B4 1A8D 96E1 02CD 218D 96D5 02E8 6500
PS Nuke5        5053 5152 5657 1E06 3D00 4B74 03E9 1C01 2E89 16DE 012E 8C1E
PS T-Rex        9C50 5351 5256 571E 063D 004B 740E 071F 5F5E 5A59 5B58 9DEA
PS Abx.1341     A5A4 C686 6F06 FFB4 1A8D 9644 06CD 21B8 2435 CD21 899E 4006
PS Abx.1520     A5A4 C686 5F07 03B4 1A8D 9634 07CD 21B4 4790 B200 8DB6 F406
PS Abx.2010     A5A4 C686 4909 03B4 1A8D 961E 09CD 21B4 4790 B200 8DB6 DE08
```

Shiny **CER:** This 921 byte virus strongly resembles the PS-MPC-generated viruses, but it is not entirely clear whether it should be grouped with them or not.

```
Shiny           80FC 1174 3480 FC12 742F 3D00 4B74 CB2E FE0E 9E03 804E 07
```

Storm.1172, Storm.1218 **CR:** 1172 and 1218 byte variants, detected with the Storm pattern.

Sylvia.1321 **CN:** This variant is 11 bytes shorter than the original one, but detected with the same pattern.

Timid.302 **CN:** Yet another variant of this 'Little Black Book' virus. Detected with the Timid.305 pattern.

Unexe **CN:** A 425 byte virus. Awaiting analysis.

```
Unexe           FFE0 0E07 56BF 0001 81C6 3101 B906 00F3 A45E 5681 C637 01BF
```

VCS.Paranoimia **CN:** This 1077 byte variant is detected with the VCS pattern.

Xph **CER:** Two variants, 1029 and 1100 bytes long. Awaiting full analysis.

```
Xph.1029        3D00 4B74 0580 FC3D 756E 9C50 5351 521E 0657 5655 8BFA 4774
Xph.1100        3D00 4B74 0580 FC3D 7555 2EC6 0670 0401 8BFA 4774 4480 3D00
```

Xtac **CER:** A 1564 byte virus. Awaiting analysis.

```
Xtac            3DC8 BA75 04B8 0214 CF3D 004B 7405 EA?? ???? ??50 5351 5256
```

VCL.Ziploc **P:** This 710 byte 'companion' virus could be detected with a set of search strings, like other encrypted VCL-generated viruses, but any scanner able to handle VCL-encryption should be able to detect it. The virus contains a text string indicating that it destroys ZIP files.

WW.217.D **CN:** A minor variant of this virus, which was originally reported as just 217. There are now four known variants of this virus, all 217 bytes long and detected with the 217 pattern.

Youth.640.B **CR:** 640 bytes. Detected with the Youth pattern.

Zherkov.2435 **CER:** A 2435 byte long variant of this Russian virus. Awaiting analysis.

```
Zherkov.2435    E800 005E 2E8A 44ED 3C00 7423 8BFE 83C7 2C90 B9BF 0851 57BB
```

INSIGHT

Old Editors Never Die...

Mark Hamilton

If it is possible to trace *Virus Bulletin*'s success to one man, it must surely be he who produced the inaugural edition in 1989: Edward Wilding, *VB's* founding editor. During his four years editing the journal, his position restricted him from expressing his own views of the industry. Now, for the first time, he goes 'on the record' with personal opinions.

Into the Hot Seat

After studying Politics and History at *Leicester Polytechnic*, Wilding began his career as a typesetter with the *Oxford Times*. Once he had become more experienced in newspaper production techniques, his next move was to *Elsevier Science Publishers.*

'Initially, I was in marketing,' he said. 'I didn't really click there. It was all right but not quite my future - so, gradually, I moved over to the *Computer Fraud and Security Bulletin.*'

During his time with *CFSB* he visited *Compsec, Elsevier*'s annual computer security conference. Here, he began to make the contacts which later helped him set up *VB*.

'At *Compsec* I met and got to know several afficionados of computer security. Viruses were a new thing, and Harold Highland had come over to present a paper. I had a long chat with him and the thought occurred to me that a bulletin on the subject of computer viruses would be a go-er. However, I didn't actually instigate it; that was done by *Sophos*, who asked, "Do you want to edit this thing?".' *VB* was finally born over a few pints in the *Eagle and Child,* a favourite haunt, with *Sophos'* Karen Richardson and Jan Hruska.

'I ummed and ahhed for quite a long time. I wondered if the virus issue were here to stay; if the magazine could employ me for a year (I stayed for four!). I asked myself: are viruses a real threat? Are they here to stay? Is *VB* a saleable commodity? There was a lot of hype on the subject; would we be tarred with the same brush, seen to be creating panic?

'I started at *VB* in July 1989, a time when there were only a pocketful of specialists. I had the impression that only about three people in the world knew anything about the subject, which was completely erroneous. There were hundreds of extremely knowledgeable people about, but I relied very much on a small team and was, perhaps, over-reliant on such individuals as the technical editor.

'The first issue went out in July; we had one subscriber, and I worked over a weekend laying the magazine out. Copy had come in dribs and drabs; it was unimpressive by any standards, even feeble. When it was completed and printed, I packed and dispatched it myself. In comparison with today's *Virus Bulletin*, it was amateurish, laughable. At the same time, it had to be: we were breaking new territory.'

The first editorial contained the passage: 'Rather like Hitler's V-1 "flying bomb", no-one knows when or where a computer virus will strike. They attack indiscriminately. Virus writers ... must know that their programs, once unleashed, soon become uncontrollable. It is, perhaps, the saddest indictment of these people that they are prepared to hurt anybody and everybody.' Such purple prose was to typify *VB*'s style for the next four years.

Wilding at *VB '93*, being 'hard and uncompromising' with *Central Point's* Tori Case.

The Popp Scoop

Had Joseph Popp not sent out his AIDS Information Disk that first December, *Virus Bulletin* might have had more difficulty establishing itself. As it was, Wilding had a 'scoop', being editor of the only journal with full and accurate details of the Trojan. The AIDS Incident, as it became known, came at a good time for the magazine and brought *VB* into the limelight.

'The *Bulletin* had been born: this was its baptism by fire,' Wilding recounts. 'The AIDS disk went out in December, and the January edition was the only publication in the world with detailed analysis of what was happening. It wasn't just superficialities; we had obtained detailed technical analysis, legal advice: information on the possible offence as it stood in Britain. We had an editorial which discussed it in detail; we put out a major alert to hundreds of companies within a day of its arrival. These were faxed throughout Britain and copies handed to employees on arrival at work.

'Everyone cooperated. Jim [Bates] was hard at work getting the thing stripped down, and always kept us informed. He said, "Look, you should publish this in the *Bulletin:* it will

be taken seriously there". Popp didn't do the magazine the damage he did to a lot of PCs. He caused inconvenience, but served ultimately to justify the magazine's existence. That was December, an exciting time, but it involved burning a lot of midnight oil.'

The AIDS disk was not a virus, despite messages on *CIX* and elsewhere suggesting otherwise, but it was an issue Wilding felt the *Bulletin* should report. 'There was a lot of duff information going about, and Jim was the only person who actually knew what the bloody hell the thing was - he had taken it apart. Nobody else knew; that's the truth of it. I read *CIX* at the time and found there were many misconceptions. People automatically assumed that because the disk did something horrible, it must be a virus. *VB* tracked the story right up to Popp's extradition and his subsequent return to the US. We followed it from start to finish; it established the credibility of the magazine.'

Hard but Fair

Wilding had a reputation for being hard and uncompromising with manufacturers as well as with his writers. Looking back on his tenure as editor, he firmly believes that the manufacturers 'by and large, do an honest day's work for an honest day's pay.'

If uncompromising, he is unrepentant: 'Okay, I had a reputation for being hard on manufacturers. It's only because of some of their ludicrous advertising claims, or because some are so arrogant about their abilities. If they set themselves up like that, they can expect to be put under the microscope, and I think the *Bulletin* did consistently that.'

Writers and reviewers were briefed to screen products in depth, and to produce fair and unbiased assessments. He believes that, because *Virus Bulletin* conducts such detailed reviews, 'a product that got a mediocre review in *VB* could actually get a very good review in other magazines'.

Wilding recalls once receiving a forty-page fax from an anti-virus company, claiming unfair treatment by the *Bulletin's* reviewer. 'It was ludicrous in the extreme,' he told me. 'I would have liked to have had a shredder attached to the fax machine!' But his real ire was directed towards the self-proclaimed 'experts' whom he has variously described as 'quack doctors, snake oil salesmen and charlatans'.

Conference Call

In 1991, Wilding brought together some of the world's leading experts on computer viruses, including Yisrael Radai, Steve White and Ken van Wyk, for the first *Virus Bulletin Conference* in Jersey. It was a huge success, and he repeated the formula, equally successfully, the following year in Edinburgh. Wilding sees *VB*, with its magazine, its book, and its conferences, as an established centre of excellence and remains convinced that *VB* is the 'only organisation in the world that can pull these things off'.

Widened Horizons

His perspectives have changed and broadened since vacating the editor's chair at the end of last year and becoming a consultant with *Network Security Management*, a company which investigates, amongst other things, large-scale computer fraud. He believes the commercial sector must accept viruses as a business hazard, and that it can and should look after itself by implementing suitable anti- and counter-virus strategies: 'If you're going to deal with viruses, work on the basis that they're here, you've got no crisis. Just clean them up as quickly and as efficiently as possible - it's a business problem, like any other.'

"In comparison with today's Bulletin, the first edition was amateurish ... at the same time, it had to be: we were breaking new territory."

He admits that he might view the virus issue too much from the industry standpoint: 'I remember receiving a call from one woman who had had her data wiped out by a virus. Of course she was a single user - it is always the poor little sods who get hit the hardest. That sort of thing had a real effect.' He also recalls, with a certain sadness, the number of people who lost their jobs as a result of loading the AIDS Trojan onto their work machines.

Wilding sees the biggest social danger as computer pornography: 'Computer porn - not the girlie-magazine type stuff, but the really nasty, vicious stuff that's going on - is far more damaging to society than a virus can ever be. A virus will only mess about with machines, but violent and degrading images can mess around in people's heads. Once you start getting children seeing the material you can find...,' he shook his head in disbelief.

The Man Himself

Much of what he does at *Network* is confidential, but, according to biographical notes in the company's corporate brochure, he is a consultant 'specialising in computer forensics, the submission of computer-related evidence, and the use of analysis software to assist asset-tracing and recovery operations'. He is clearly proud of his involvement with the in-house software systems, claiming that the company has the most advanced commercial investigative facilities in the UK, possibly in the world.

Computers and related crime are not, however, his whole life; he has disparate interests. He used to jump out of planes for pleasure, but now has more sedentary pursuits: beer, cigarettes, women and writing short stories ('but not science fiction!'). It is said that truth can be stranger than fiction - and Wilding's experiences in the anti-virus arena certainly strengthen this assertion.

VIRUS ANALYSIS 1

Satan Bug - Unleashed!

Jim Bates

The task of disassembling virus code is at best repetitive and at worst completely demoralising. There are occasional highlights when the efforts of the virus authors generate a little amusement at the sheer stupidity of their designs, but the usual feeling is one of tight-lipped disgust.

With most viruses, it is possible to get some slight feel for the attitudes and competence (or lack of it) shown by the writer, and there are even some stylistic patterns which smell of collusion or plagiarism. This month's virus is called (by the writer) Satan Bug, and is reportedly in the wild in the United States. One worrying aspect of the virus is that, from both style and design, the odour of some inside knowledge of the anti-virus industry is almost overpowering.

Multiple Encryption

As I have remarked before, most virus writers have a single idea which they milk dry. In this case, the idea which is thrashed to death is that of encryption and polymorphism - concepts which the virus author has taken to such extreme lengths that processing is noticeably slower when the virus is memory-resident. Not only is the code encrypted and randomised during infection, but the actual memory image is similarly encrypted and randomised during each intercepted DOS call. During a single access, the encryption/decryption routines may be called over 500 times! It is this obsessive attention to the alteration of the memory image (among other things) which leads me to suspect the involvement of someone within the anti-virus industry.

Detailed analysis of the code reveals that these features are not the only way in which the virus author has attempted to make life more difficult for the anti-virus software manufacturers. The design of the decryption routine will cause problems for less complex scanners. In addition, the virus' 'Are you there?' call will only respond if interrogated by the virus code itself.

The virus contains no trigger routine or payload, its only action being to replicate. Due to programming errors, Satan Bug will cause file corruption and system malfunction under certain circumstances.

Installation

When an infected program is first executed, processing is immediately transferred to the virus code. This begins with an extended series of 'junk' instructions within which are dispersed just a few instructions essential to the decryption of subsequent code. In the sample examined, the junk code consisted solely of single byte instructions and extended to over 600 bytes before the true virus code was reached. Such a large block of randomly generated junk code will in itself be a sufficient indication to heuristic scanners that something is amiss. It is interesting to note that the decryption point is located before the end of the actual decryption loop, so that the first part of the loop must be executed before its true extent is known. This will certainly disrupt the operation of scanners which rely solely upon structure matching for detection purposes.

Once the virus code has been decrypted, the virus resets its segment registers so that the code appears to be located at a standard offset. This is a sure sign of a programmer who flunked classes after only one or two lessons, and does not know how to manage segment manipulation.

Once this initial location is settled, the virus checks whether the host file was an EXE or COM type and repairs the header accordingly. The next process issues an 'Are you there?' call to determine whether the code is already resident and, if so, processing is returned to the host file.

Interpreting Intent

If the virus is not already resident, further checks are instigated before final installation takes place. The code first examines the environment to locate the COMSPEC variable and checks to see if it is the standard COMMAND.COM file. Unusually (there's that smell again), if it is not COMMAND.COM the virus does *not* become resident and processing is returned to the host file. Otherwise, the current memory block is checked to see if it is the last in the chain. If it is not, no further tests are made and again the virus does not become resident. This too is noteworthy in that only the current block is checked.

If all these checks succeed, the virus steals a total of around 10K from conventional memory and relocates itself at the top of memory before hooking the DOS Interrupt vector and returning control to the host program.

At this point, the virus memory image is not encrypted but any subsequent calls to the DOS service routines will change this as various sections of the virus code are invoked.

Operation

When the virus is memory-resident, the virus intercepts only four function calls: 3Dh (Open file), 4Bh (Load and Execute), 6Ch (Extended Open) and the virus' own 'Are you there?' call, F9h.

On each of the first three function requests, the virus will attempt to infect the target file if it has the extension COM or EXE, and is greater than 1023 bytes in length. COM files are identified simply by checking that the extension is COM,

while EXE files are identified by the presence of the 'MZ' header at the beginning of the file. COMMAND.COM will become infected when the virus is resident, and a special provision is made to abort the environment test during installation, if it is the host file.

If a target file is marked as Read-only, the virus saves its attributes (for later repair) and clears them to allow write access during the infection phase. A special exception is made during the Load and Execute intercept to avoid infection during a spawn (subfunction 3).

"This will certainly disrupt the operation of scanners which rely solely upon structure matching for detection purposes."

Once the target file has been located, the date field is checked before it is opened to see if it has had a value of 200 added to the year field - this is the virus' infection marker.

Who's There?

The infection process is the usual one of appending the virus code to the file. Interestingly, instead of arranging for either the initial instructions (COM files) or the entry pointer (EXE files) to be modified to point into the virus code, Satan Bug will trace the program path and insert a jump later in the file.

As I have already mentioned, the virus code is encrypted and a sequence of junk code instructions are generated to lead into the decryption routine. By limiting the junk code to single byte opcodes, the virus writer has unwittingly provided an easy way of identification for this virus and, with a reasonable scanner, no real difficulty in identification should be encountered. There is no attempt at stealth, so even all but the most archaic of integrity checkers should be able to detect the presence of the virus.

Although the virus remains in memory in unencrypted form immediately after installation, the first intercepted function request will invoke a complex series of code generators and encryption routines, such that the memory-resident code is changed. This happens each time an interception is made and causes the virus code to change its appearance constantly. Even the 'Are you there?' call will cause changes to be made, but a more important point is that the call is supplemented by a regressive check on 166 bytes of the calling code. This ensures that it is the virus making the call and not some anti-virus program attempting to emulate it - still more evidence pointing to a possible 'inside job'.

There are several bugs in the code, as well as faults in the design, which will cause system malfunction under specific circumstances. The usual effect will be for the system to hang for no apparent reason, but there may occasionally be truncation or corruption of some program files.

There is a single text message within the virus code ('Satan Bug - Little Loc'), obviously so that investigators will name the virus author's handiwork correctly. This message is not displayed during virus operation.

Targeting

Just prior to attempting infection, the virus makes two final checks on the target file. The first of these checks an 8 byte block of code against offset 6 of the targeted file and, if there is no match, infection proceeds with the normal modification of the program entry point.

If there is a match, the initial instructions will remain the same and the virus calculates an offset before inserting a jump into the appended virus code. The relevant 8 bytes are as follows:

22h, 19h, 35h, 93h, 59h, 57h, 54h, and 80h

The second check is against code taken from the end of the target file. This involves searching the last 75 bytes of the file for a specific sequence of bytes and, if found, they are overwritten with zeros. The sequence in question begins:

F1h, FDh, C5h, AAh, FFh, F0h

The purpose of these checks is unclear, but is likely to be a deliberate targeting of anti-virus programs or self-checking routines. Any readers identifying their own code are asked to let me know the relevant details and I will happily pass my notes and disassembly on to them.

Satan Bug

Aliases:	None known.
Type:	Parasitic appending.
Infection:	COM and EXE files.
Self-recognition on Files:	200 is added to the year field of infected files.
Self-recognition in Memory:	Via an 'Are you there?' call. The memory-resident handler checks caller's ID before returning.
Hex Pattern:	Highly polymorphic. No simple search pattern is possible.
Intercepts:	DOS INT 21h - subfunctions 3Dh (Open File), 4Bh (Load and Execute), 6Ch (Extended Open) and F9 (Virus' 'Are you there?' call).
Trigger:	None.
Removal:	Delete and replace infected files under clean system conditions.

VIRUS ANALYSIS 2

Carbuncle - A Nasty Infection

Eugene Kaspersky

One of the least common methods of infection employed by virus writers is that of the companion virus. This infection technique is possibly the least destructive available, as none of the code which makes up the 'host' executable is altered. This is because a companion virus infects a file by creating an additional program which is executed instead of the intended file.

The genus 'companion viruses' can be firther subdivided, into several subgroups, depending on exactly how the viruses function. The most common type of companion virus operates by searching for files with an EXE extension (generally using the DOS FindFirst and FindNext functions). This type is never memory-resident, and utilises a 'single shot' approach. However, certain companion viruses [*for example, Batman, VB, March 93, pp. 12-13. Ed.*] *are* capable of becoming memory-resident, and generally intercept Int 21h. Whether or not they are memory-resident, companion viruses almost always work by creating a file with the same name as the infected executable file, but with a COM extension.

A Faithful Comrade...

When a user attempts to execute a file, he does not usually specify the file's extension - for example, when attempting to run a file WORDPROC.EXE the user could simply type WORDPROC, and DOS would search for files which match this filename and have an executable extension.

There are usually only three types of file which DOS considers to be executable, and these have the extensions COM, EXE or BAT. In order to take account of the case where there are several files all with the same name in a particular directory, DOS has a built-in hierarchy of 'executability': DOS will execute a COM file rather than an EXE file, and an EXE file rather than a batch file.

As an interesting aside, in DOS versions 3.30 or earlier, the command interpreter will search for a COM file first, even if a different executable extension has been specified. This means that it is possible to type:

```
C:\> PROGRAM.EXE <enter>
```

but for DOS to execute the file PROGRAM.COM (if it exists). Fortunately, DOS 4.00 or above operates marginally more intelligently, and if the user specifies a particular file extension, DOS will search for those files which match the given specification exactly.

Following the Path

The 'Standard' companion virus takes advantage of this hierarchy. However, a refinement to this technique has been developed which has all the benefits of the standard technique, but fewer of the drawbacks.

When a 'Path' companion infects a file, it renames the 'host' file with a non-executable extension. Then, a file with the same name and extension, containing the virus code is created in its place. When the user executes the infected application, the virus code, and not the host file, is executed. It becomes memory-resident, carries out whatever operations it is programmed to run, and loads the original application.

Path and Standard companion viruses differ, in that:

- Path companions do not rely on the fact that DOS has an order in which it will execute files
- Standard companion viruses make no alterations whatsoever to the host file, whereas Path companions change at least the name of the infected file.

It is much more difficult to spot the presence of a Path companion than a Standard one. Companion COM files are always placed in the same directory as the host file, although sometimes the virus will mark these files as hidden. The presence of two files, both with an executable extension, is enough to arouse suspicion. Unfortunately, it is possible for a Path companion to hide the files by marking them with the Volume attribute, thereby making them invisible to DOS - unless searching on a sector-by-sector basis.

This is the story of all companion viruses apart from one. The virus writers have not been idle: Carbuncle is a completely new type of companion virus.

Inside the Carbuncle

Carbuncle is a non-memory-resident virus, consisting essentially of a COM file 622 bytes long. Like so many viruses encountered nowadays, Carbuncle is the result of the experiments by the so-called computer underground. The sample I received even had the assembler source code attached! The documentation from that source stated:

```
The PC CARBUNCLE VIRUS - a companion virus
for Crypt Newsletter 14

The PC Carbuncle is a 'toy' virus which will
search out every .EXE file in the current
directory, rename it with a .CRP [for Crypt]
extent and create a batchfile. The batchfile
calls the PC Carbuncle [which has copied
itself to a hidden file in the directory],
renames the host file to its NORMAL extent,
executes it, hides it as a .CRPfile once again
```

```
and issues a few error messages. The host
files function normally. Ocasionally [sic], the
PC Carbuncle will copy itself to a few of the
host .CRPfiles, destroying them. The majority
of the host files in the PC Carbuncle-
controlled directory will continue to
function, in any case. If the user discovers
the .CRP and .BAT files and is smart enough to
delete the batchfiles and rename the .CRP
hosts to their normal .EXE extents, the
.CRPfiles which have been infected by the
virus will re-establish the infection in the
directory.

—Urnst Kouch, Crypt Newsletter 14
```

More detailed analysis reveals further information about the operation of the virus. When it is executed, the system time is checked. If this meets the trigger conditions, the trigger routine is called (see below), otherwise control is passed on to the infection routine. This routine creates a file with the name CARBUNCL.COM, and assigns the attributes Read-only and hidden to it. If this file is already present, it is overwritten, although if it is already marked as Read-only, the routine fails.

The virus then searches for any files with the extension EXE, using the DOS FindFirst/FindNext function, and infects them. Infection is carried out by renaming the target file's extension to CRP, and creating a companion batch file. As a result of the infection, there will now be two files with the same file name, but with the extensions BAT and CRP.

The companion batch file contains six lines of DOS commands. If the file FILENAME.EXE was infected, the companion FILENAME.BAT contains these lines:

```
@ECHO OFF
CARBUNCL
RENAME FILENAME.CRP FILENAME.EXE
FILENAME.EXE
RENAME FILENAME.EXE FILENAME.CRP
CARBUNCL
```

When the user tries to execute an infected file, the batch file is executed instead. The first line of the batch file ensures that the later commands are not echoed back to the screen. Next, the main body of the virus (stored in the file CARBUNCL.COM) is executed, and more files on the disk are infected. The host executable is then named back to its former executable extension, executed, and renamed to the extension CRP for further use. The last action of this batch file is to execute the main body of the virus.

As a result of this mode of infection, all executable files within one directory are replaced by batch files of the same name. Only one occurrence of the virus body needs to be copied into a directory in order for all files in that directory to become infected. This 'one shot approach' is reminiscent of the link virus, DIR-II.

One advantage of this infection strategy is that it obviates the necessity for the virus to be able to recognise files which are already infected, as the virus only searches for files with the extension EXE. Once a file has been infected, the host executable will have been renamed to the execution CRP, and will therefore not be re-infected.

Pulling the Trigger

The trigger routine is very simple: if the system time has a seconds field value of less than 17, the virus searches for five files with the CRP extension and overwrites them with the body of the virus code. As a result, these files are not recoverable, and should be deleted. If they are executed, they will simply cause the virus to spread.

Fortunately, the virus is more of theoretical interest than an actual threat to users - at least in its current form. The virus is not difficult to detect, and contains the text strings '*.crp CARBUNCL.COM BAT*.exe CRP', which are used when searching for uninfected files. In addition, the virus uses the following text string when creating the companion batch file:

```
@ECHO OFF
CARBUNCL
RENAME
```

It also contains the 'copyright' string 'PC CARBUNCLE: Crypt Newsletter 14'

Conclusions

Carbuncle provides virus authors with yet another way to infect files. The continual war between the computer underground and the anti-virus software manufacturers continues with no sign of lessening - at least for as long as users continue to run DOS.

Carbuncle

Type:	Not memory-resident, Companion.
Infection:	EXE files only.
Length:	622 bytes.
Self-Recognition on Files:	None necessary.
Self-Recognition in Memory:	None necessary - the virus does not become memory-resident.
Hex Pattern:	`BA62 02B9 0000 B43C CD21 8BD8 B96E 02BA 0001 B440 CD21 B43E` or the text string PC CARBUNCLE: Crypt Newsletter 14
Trigger:	Overwriting of CRP files.
Removal:	Delete BAT files and the file CARBUNCL.COM, and rename CRP files to the extension EXE.

VIRUS ANALYSIS 3

Quox

Tim Twaits

The Quox virus (also known as Stealth 2 Boot) was first discovered over seven months ago, but until recently remained a very rare specimen. A recent sighting, however, has prompted further investigation of its capabilities.

Operation

Quox is a simple Master Boot Sector infector which does not contain any deliberate side effects; its only function is replication. The virus code occupies a single disk sector of 512 bytes which replaces either the Master Boot Sector on fixed disks, or the DOS Boot Sector on floppy disks.

When a system is booted from an infected disk, the virus installs itself in the top 1K of base memory and hooks the interrupt 13h vector (the BIOS disk services), before loading and executing the original boot sector. The location of the original boot sector depends on the type of disk: this is usually the last sector on the disk for floppy disks, and the last sector on the first cylinder for fixed disks.

The algorithm used to determine the location at which the original Master Boot Sector is stored does not always function correctly. The calculation made by the virus relies on the manner in which the DOS FDISK program partitions the disk. While most versions of DOS conform to the same conventions, this is not universally true, as certain OEM manufacturers (eg *Amstrad* DOS v3.2) make this calculation in a proprietary manner. In these cases, the copy of the boot sector may be written to an arbitrary sector on the disk. In the event of the calculated location lying outside the physical bounds of the disk, the infection routine is aborted.

Interrupt 13h

The virus intercepts disk read and write operations which use the BIOS disk services. Any attempt to access the boot sector of an uninfected disk causes the disk to be infected: even inserting a floppy into an infected machine and typing DIR will cause the disk to become infected. In a similar manner, when an infected floppy is used to boot a clean system, the fixed disk will become infected during the boot process when accessed by DOS. Once active, the virus effectively hides any changes to the boot sectors, and a request to read the boot record is redirected so that the original contents are returned to the caller.

General Failure Errors

The virus uses the same code to infect both Master and DOS Boot Sectors. In an attempt to ensure that infected disks will still function correctly, the virus does not change the data in those areas of the boot sector used for the BIOS Parameter Block and the Partition table. Therefore, if an uninfected system attempts to read an infected disk, the parameters DOS needs to access that disk will be present. This reduces the space available for the virus to 392 bytes.

In order to gain extra room for code, the author has assumed that the first few bytes of the BIOS Parameter Block are superfluous. This assumption works - except with 1.4Mbyte 3.5-inch disks, where the modification may cause the disk to become unreadable under DOS. The disk can still be read while the virus is active but, on a clean machine, attempts to access the disk will fail, giving the message 'General failure error'. This failure is due to an undocumented quirk of the MS-DOS operating system.

Removal

Disinfection must be undertaken in a clean DOS environment (that is, having booted from a write-protected clean system diskette). The virus can be removed from a fixed disk using the FDISK /MBR command available in DOS 3.31 and above, or by using a disk editor to restore the copy of the boot sector saved by the virus. It can be removed from a floppy disk by formatting the disk or by copying a valid boot sector from an uninfected disk of the same type. This latter method can be used to recover data from disks rendered unreadable by the virus.

Quox	
Aliases:	Stealth 2 Boot
Type:	Master Boot Sector infector.
Self-recognition on Disk:	Checks Boot Sector for a sequence of hexadecimal bytes.
Self-recognition in Memory:	Checks top of memory for a sequence of hexadecimal bytes.

Hex Pattern:	(at start of boot sector)
	`B902 00F3 A5A1 1304`
	`8BD0 B106 D3E0 BE00`

Intercepts:	The virus hooks Int 13h for boot sector infection and stealth.
Trigger:	None.
Removal:	Under clean system conditions use the DOS command FDISK /MBR, or identify and replace original Master Boot Sector.

TUTORIAL

The Shape Shifters

Whenever one casts an eye over any article in the popular press on computer viruses, one will inevitably come across the term 'polymorphic'. What are polymorphic viruses? How do they work? How can they be detected? This article will take an historical look at polymorphic viruses; how they developed, and where they might lead.

In the Beginning

In the beginning, most viruses were very simple, and the first scanners operated by searching for part of the virus code. Every different program stored on the hard disk consists of a unique series of bytes. Therefore, it is possible to search for a particular program by searching files on the disk for this 'signature'. This pattern would be suitable for detection purposes if it were chosen in such a way as to be unlikely to appear in other pieces of code (eg, the code used for the virus' own 'Are you there?' call). Early virus scanners were slow and inefficient; they searched blindly through every byte of a file looking for the virus 'signature'.

Falling Down

The first sign that things might not always be this simple came when Cascade appeared. This virus encrypted all of its code, except for a short decryption routine located at the start of the virus. The encryption routine used by the virus is shown below:

```
0100 E94700       JMP   014A

<HOST FILE CODE.....>

0149 01           DB    01
014A FA           CLI
014B 8BEC         MOV   BP, SP
014D E80000       CALL  0150
0150 5B           POP   BX
0151 81EB3101     SUB   BX, 0131
0155 2E           CS:
0156 F6872A0101   TEST
                  BYTE PTR [BX+012A], 01
015B 740F         JZ    016C
015D 8DB74D01     LEA   SI, [BX+014D]
0161 BC8206       MOV   SP, 0682
0164 3134         XOR   [SI], SI
0166 3124         XOR   [SI], SP
0168 46           INC   SI
0169 4C           DEC   SP
016A 75F8         JNZ   0164
```

Cascade decryption routine.

The first instruction is a jump to the appended virus code, which in turn copies the stack pointer before calculating its own relative offset in memory (using the familiar CALL Next_Address, which pushes the current value of IP onto the stack, followed by a POP instruction). The next instruction tests a marker stored in the file to see whether or not the file is encrypted. This allows the virus author to develop the code in an unencrypted form.

The main body of the decryption routine is stored in the instructions from 015Dh to 016Ah. Here a loop is set up, where consecutive bytes of the virus code are decrypted by XOR-ing them with the two keys SI and SP. This continues until the entire virus has been decrypted.

However, the essential elements of this code are only important for their functionality; there are many different instruction sequences which would decrypt the virus. Take the example of the changes made to the decryption key by the INC SI instruction. The position of this instruction is not critical as long as it comes *after* the two XORs and *before* the end of the loop. Thus, the virus could swap the position of the INC SI instruction with the DEC SP instruction with impunity: although the hex fingerprint of the code has altered, its function has not.

If the virus author wished to be clever, he could have substituted the command sequence POP BX with POP CX, MOV BX, CX, or with any number of alternative instruction sequences. It was only a matter of time until a virus writer would utilise such techniques, whereupon every infection of the virus would appear different. Thus the word 'polymorphic' was coined.

Mark Washburn and 1260

Probably the 'breakthrough' (if one can use such terms) for the virus authors was the work carried out by Mark Washburn. Claiming to be a genuine 'virus researcher', Washburn developed a series of viruses which showed beyond doubt that polymorphism worked.

Washburn wrote the V2P? series of viruses, starting with the slightly polymorphic V2P1 (or 1260) virus and graduating to the much more complex V2P6. With the development of his ideas, Washburn secured himself a place in the history books as the father of polymorphic viruses.

Washburn's first polymorphic virus, 1260, used a very simple algorithm to vary its appearance. Two typical 1260 decryption routines are shown in figure 1.

Understanding the purpose of each instruction is not necessary in order to comprehend Washburn's strategy. 1260 uses the same eight instructions, placed in a random order, to generate the decryption routine, but pads these instructions out with 'Do nothing instructions'. The important instructions in the 1260 detection routine are highlighted in the figure. In essence, the 1260 virus XOR-ed each byte of

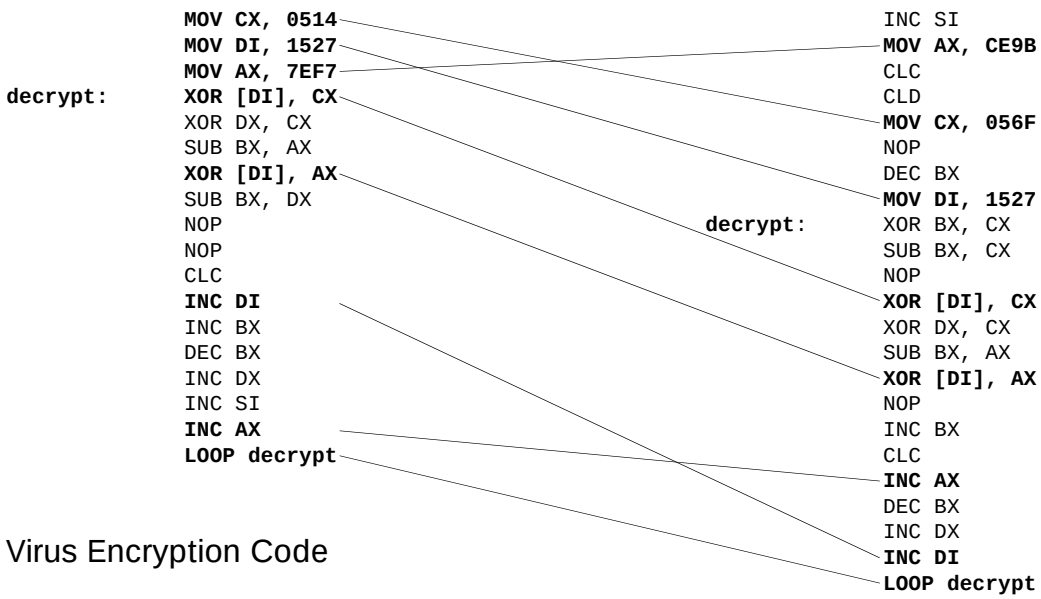


Figure 1: 1260 Virus Encryption Routine: This diagram shows two examples of the 1260 virus' decryption routine. Note that both files contain eight of the same instructions - these set up the encryption loop. The other instructions are chosen at random, in such a way that they do not affect the contents of the registers used for the decryption routine. This makes the virus impossible to find using a simple hexadecimal pattern search.

the main virus body with two decryption keys, somewhat like Cascade. In this case, the important instructions are padded by 'junk' operations like DEC BX or CLC, which do not affect the functionality of the decryption routine. An explanation of the routine is given in the figure caption.

Clearly, looking for such a virus with a hex pattern search is out of the question. However, it is reasonably easy to see that this virus could be detected with a simple search algorithm.

The Mutation Engine

By far the best known polymorphic viruses are those which use the Mutation Engine. This program is (contrary to popular belief) not itself a virus: it is an object module, which was designed to be added to a virus in order to make it polymorphic.

The MtE uses all the tricks developed by Washburn, and more. An experienced eye can pick out an MtE infected file just by glancing at a DEBUG listing, but analysis reveals that each individual infection appears to be completely different. With the number of possible mutations of the code running into billions, the MtE presented a problem of some magnitude to scanner manufacturers when it first appeared. Not only was it a polymorphic engine, rather than a virus, but the code had been extensively circulated among the computer underground. The MtE even included an example virus which showed naïve users how it could be applied.

The coding of the MtE is much more complex than any of its predecessors. Two short snippets of MtE-produced code are illustrated in figure 2, and it is clear that, although the code 'looks' suspicious, it is very difficult to detect accurately and reliably. Faced with such an insurmountable problem, how could the anti-virus software vendors fight back?

Countermeasures

While virus writers were busy improving their handiwork, development was also continuing within the industry. Scanner developers had long since realised that it was impossible to continue using a simple hex pattern to search for viruses: it was slow, and could not keep up with the rising complexity of the way viruses worked. It was time for a change, but with an ever-variable target, onto what properties could the virus hunters latch?

One of the principal disadvantages of using a simple hex pattern search was that it did not utilise much of the information which the developers knew about a virus. Consider the algorithm which a simple parasitic virus uses to infect a COM file. A JMP instruction is inserted at the start of the host file, and code appended at its end. A simple hex pattern throughout the entire file would reveal the presence of the virus, but a much more efficient algorithm would be simply to scan the beginning and the end of the file: that is, *the areas in which one knows the virus must be if it has infected the file*. The next generation of virus scanners took advantage of this Achilles' heel: rather than using a simple hex pattern, the scanner would trace through a file, following the executable path.

Jim Bates, author of the *VIS Anti-virus Utilities*, explained that it was possible to search through a file looking at structure, rather than at individual instructions. 'Basically, a

polymorphic virus has two big weaknesses,' he explained. 'Firstly, the virus decryption code must always be unencrypted, and secondly, it must always be in some kind of loop.' Bates has developed code within *VIS* which can examine files for viruses by taking advantage of the structural similarities between different infections of the same polymorphic virus. This information is then hard-coded into the *VIS* product.

Virus Description Languages

This idea can be taken much further by writing an interpreter which removes the need for each polymorphic detection routine to be hard coded. The problem with hard coding is that, although it executes extremely quickly, it is time-consuming to develop and maintain. Consider again the code example for the 1260 virus. Dr Jan Hruska, from *Sophos Plc*, explained that they have developed code within his scanner which 'understands' the instructions which make up machine code. '*VDL* [Virus Description Language] is capable of examining where the initial instructions of a file lead. For example, in the case of the 1260 virus, one knows that the first instruction of an infected file must lead to a certain offset from the file's end.'

Dr Alan Solomon, of *S&S International*, uses a variety of techniques to detect complex polymorphic viruses. 'The *Toolkit* describes viruses using *VIRTRAN*. A typical *VIRTRAN* detection routine contains naming, detection, identification and removal information. For the more complex polymorphics I also use statistical analysis of the code, to ensure that the relative usage of instructions fits the pattern.' The latest weapon added to the Solomon armoury is the addition of his 'Generic Decryption Engine', which is capable of decrypting polymorphic viruses. In such viruses, it is usually only this decryption code itself which varies between infections; the virus code 'inside' the encryption is static. This means that once the virus is decrypted it is trivial to detect - with a very low risk of false positives.

```
PUSH DX
PUSH AX
PUSH SI
PUSH DI
PUSH BP
PUSH BX
PUSH CX
MOV DI, F236
MOV AX, 1819
MOV DX, 0A69
MUL DX
MOV BX, AX
MOV AX, D456
MOV CX, DI
AND CL, 1F
SHR AX, CL
MOV BP, AX
MOV CX, DI
ROR AX, CL
MOV SI, AX
SS:MOV [DI+2349], AX
MOV CX, SI
ROL AX, CL
XOR AX, BP
...
```

Example 1

```
PUSH AX
PUSH DX
PUSH BX
PUSH SI
PUSH BP
PUSH DI
PUSH CX
MOV AX, 0620
MOV CL, 09
SHL AX, CL
ROR AX, 01
MOV BX, AX
MOV AX, 0192
MOV DX, F0C3
MUL DX
SUB AX, BX
MOV BP, AX
MOV AX, A60A
XOR AX, BP
MOV DI, AX
MOV AX, 09E6
MOV CX, BP
ROL AX, CL
MOV SI, AX
...
```

Example 2

Figure 2: The Mutation Engine: the first few bytes of two different MtE-generated decryption routines. Note that, although it is easy to 'see' intuitively that this code is oddly written, it is hard to write a program which will detect it reliably.

The Future

The main problems which the industry faces is that viruses are set to become still more highly polymorphic. Will it always be possible to scan for viruses? Solomon is confident: 'Detecting a polymorphic virus can be tricky, but will always be possible. It is like playing the game of "who can think of the highest number". The person who goes second always wins!'

One thing which is highly likely is that more polymorphic engines will be written. Already there are several like the MtE - most notably the Trident Polymorphic Engine (TPE) and the NuKE Encryption Device (NED) [*See page 16. Ed.*]. As these virus construction tools become more widespread, it is likely that users will start to find more MtE viruses in the wild.

The anti-virus industry seems to be divided about how things should proceed from here. Is it necessary to decrypt the contents of infected files so that a scanner can identify the virus which lurks beneath the encryption, or is it simply enough to inform the user that a particular file is encrypted with the Mutation Engine?

The argument for precise identification is that if one intends to disinfect a file, it is of paramount importance that the disinfection routine has correctly identified the virus - if not, disinfection will almost certainly fail.

With this in mind, a number of vendors have developed code which allows the scanner to 'strip back' infected files and reveal the unencrypted virus - thereby allowing the file to be (where possible) disinfected. This is precisely the rationale behind the Generic Decryption Engine.

The counter-argument is that this procedure is simply too time-consuming, and that users do not care exactly which virus has infected their machine - they simply want it eradicated. This will have a speed advantage when dealing with badly infected disks, but means that disinfection of files infected by viruses which utilise such polymorphic engines will be impossible.

Whether or not such techniques are sufficient to keep the tide of polymorphic viruses at bay, one thing is certain: viruses of the future will be increasingly polymorphic. Are developers ready for the next generation of the 'Shape Shifters'?

FEATURE

The Nuke Encryption Device

Edward J. Beroset
Datawatch Corporation

As computer virus detection capabilities have become more sophisticated over recent years, so have the ways in which viruses attempt to avoid detection. One of the most troublesome techniques employed by virus writers is that of polymorphic encryption, and the most recent tool used to accomplish this is the Nuke Encryption Device (NED). This article follows on from the more general discussion of polymorphism (pp.13-15), and looks in detail at how a polymorphic engine functions.

Like the Trident Polymorphic Engine (TPE) and the Dark Avenger Mutation Engine (MtE) before it, NED is distributed as a linkable object module with sample code illustrating its intended use as a virus encryptor. The object module is 1524 bytes long, but the actual code size when linked is 1356 bytes. Although NED is not itself a virus, the intentions of the author (he claims to be the same person who wrote the Virus Creation Laboratory) are clear. Like the VCL, NED is a rather sloppy piece of work, full of unreferenced variables, duplicate variables, poorly written code, and plain bugs.

Inside the Engine

Polymorphism is defined by *Webster's Dictionary* as 'the quality or state of being able to assume different forms', and most of the code in NED is devoted to the purpose of assuring that the decryptor is polymorphic. Some of the tricks NED uses to accomplish this are:

- inserting 'garbage' between the decryptor loop and the start of the virus code
- using 'dummy' instructions (e.g. NOP, ADD AX, 0000, etc.) to vary the length of the decryptor code
- using different operations for encryption - ADD, SUB, and XOR
- using either byte or word operands
- using either immediate values or preloaded registers for the encryption value

Calling NED

The calling sequence for NED requires that registers be loaded with a pointer to a buffer area, the start address and length of the code to be encrypted, the offset of the virus to be in memory when it actually runs, and the various flags which control how NED operates. Dummy instruction insertion, use of move variations (eg code other than simple MOVs that accomplish the same effect), ADD/SUB substitution, garbage code insertion, and data and code segment equivalence assumption are selected by individual bits in the SI register. NED then creates the decryptor and garbage code (if selected), and encrypts the code into the specified buffer. It returns the combined length of the decryptor, garbage code, and encrypted code in the AX register.

The Generation Game

The encryptor takes advantage of the fairly consistent way in which *Intel* processors' instructions are encoded. All instruction encodings are subsets of one general format: zero to four prefix bytes, one to two opcode bytes, zero to two MODRM (MODe, Register/Memory) operand specifier bytes, zero to four address displacement bytes, and zero to four bytes for an immediate constant. Of these components, the opcode and MODRM bytes are the most important to the functioning of NED.

Generally, opcode bytes define the type of operation performed, while the MODRM bytes control addressing modes and register use. In some cases, both the opcode and MODRM byte specify the type of operation performed.

To illustrate this, the instruction ADD AX, 5528h may be encoded as 81C0 2855. By altering a single bit in the MODRM byte, the instruction becomes ADD CX, 5528h. Replacing the low three bits of the MODRM byte with all possible combinations from 000b to 111b enables the registers AX, CX, DX, BX, SP, BP, SI, and DI to be referenced. Replacing the high three bits in the MODRM byte in a similar fashion cycles the opcodes through ADD, OR, ADC, SBB, AND, SUB, XOR, and CMP.

Clearly, a program incorporating this information could create machine instructions based on a 'Chinese menu' approach - take one option from column A and one option from column B and combine in any manner. Extrapolating this concept yields a program capable of generating a coherent decryptor loop.

NED has three main functional components: a decryptor generator, an encryptor, and a random number generator. The random number generator and encryptor parts combined represent about ten percent of the code, while the remaining ninety percent is comprised of the decryptor generator.

When NED is called, it first makes a few random decisions which affect how the decryptor loop is generated. This process chooses whether to encrypt by the byte or the word, which register is to be used as the loop counter, whether to use an immediate value or a register for the encryption constant, and (if the latter) which register it should be. The only other decision made in this routine is whether to initialise the loop counter or the memory pointer first.

Figure 1. A Simple NED Generated Decryptor

```
0100 MOV    CX,5DFE        ; load the loop counter in two steps
0103 XOR    CX,5EFF        ; which leave 0301h in CX
0107 MOV    BP,0116        ; load the loop pointer
010A XOR    WORD [BP],E9F0 ; do decryption with immediate value
010F ADD    BP,0002        ; increment the pointer
0113 DEC    CX             ; decrement our loop counter
0114 JNZ    010A           ; and keep going if there's more 0116
                           ; encrypted data starts here
```

Figure 2. A More Complex NED Generated Decryptor

```
0100 NOP                   ; dummy instruction
0101 JLE    0103           ; dummy instruction
0103 ADD    AX,0000        ; dummy instruction
0106 MOV    AX,66B3        ; load loop counter
0109 SUB    BX,0000        ; dummy instruction
010C SUB    BX,0000        ; dummy instruction
010F ADD    AX,9F4F        ; adjust loop counter to desired value
0113 MOV    SI,0164        ; load pointer
0116 MOV    BL,CD          ; load decryption constant
0118 MOV    BL,5D          ; in two steps
011A ADD    [SI],BL        ; do the actual decryption
011C NOP                   ; dummy instruction
011D INC    DI             ; dummy instruction
011E DEC    DI             ; dummy instruction
011F NOP                   ; dummy instruction
0120 INC    SI             ; increment pointer
0121 INC    AX             ; dummy instruction
0122 DEC    AX             ; dummy instruction
0123 ADD    BX,0000        ; dummy instructions
0126 DEC    AX             ; end the loop by decrementing counter
0127 JNZ    011A           ; NED always uses JNZ to end the loop;
<here through 0163h is filled with garbage bytes>
0164                       ; the actual virus code would begin here
```

A subroutine which generates dummy instructions is called after these decisions are made and also after each decryptor loop component is created. As with all code generating subroutines in NED, this dummy instruction subroutine writes the instructions it generates to the next available location in the buffer and then updates the buffer pointer. Its purpose is to insert one or more valid instructions which have no practical effect on the subsequent code. The different tricks employed by NED include moving a register into itself, exchanging a register with itself, incrementing and then decrementing the same register, complementing the carry flag twice, conditional or unconditional jumps to the next instruction, and the ubiquitous NOP instruction.

Two subroutines generate the code necessary to initialize the decryptor. The first generates the code responsible for loading the loop counter register with the initial value based on the size of the code and the encryptor. Any of the eight general-purpose sixteen bit registers except SP may be used as the loop counter. The second routine generates code which loads the pointer register with an initial value that points to the beginning of the encrypted data.

If the original 'coin toss' decision was to use a register instead of an immediate value, the next subroutine called will generate code which loads some constant value into either a byte or word sized register. In the case of byte-sized registers, NED is capable of using any of the eight available on an *Intel 80xx* processor (ie, AL, AH, BH, BL, CH, CL, DH, DL). The loaded constant is the number successively applied to each of the words or bytes in the encrypted code.

Looping the Loop

The next subroutine creates the 'heart' of the decryption routine. This routine generates an instruction which applies the above mentioned constant to the contents of the memory referenced by the pointer. NED is not a very bright animal and only knows three tricks - encryption using ADD, SUB, or XOR.

If the calling program has set the appropriate flag, the code also generates a CS segment override to address the data that immediately follows the decryptor. The intent of this clearly was to allow the infection of EXE as well as COM files, but a bug in the code prevents reliable infection of EXE files approximately one time in sixteen.

To finish the decryptor loop, another subroutine generates code which decrements the loop counter and checks to see if it has reached its terminal value. NED always decrements the loop counter and the terminal value is always zero. If the loop counter register happens to be CX, the routine has a 50% chance of using the LOOP instruction; otherwise it is treated like any other register. If the loop counter is not CX, it will be decremented by DEC instruction(s), or an appropriate ADD or SUB. This routine also generates the final command in the loop. This is always a JNZ instruction.

The last code-generating routine simply generates a short relative JMP instruction, followed by a variable number of 'garbage' bytes ranging from zero to one hundred.

Once the decryptor has been written to the memory buffer, the only remaining task is to copy and encrypt the target code. This is done with self-modifying code. The routine as written is a simple LODSB; XOR AL, BL; STOSB; LOOP construct. NED modifies the LODS/STOS instructions to word size if required. The XOR operation is substituted for the inverse of the operation used by the decryptor, and is also adjusted to use AX, BX for decryption routines which operate one word at a time.

NED's random number generation routine uses the system clock as a raw source, performing a few perfunctory mathematical transformations on the clock tick counter in an attempt to make the output more random. A statistical analysis of over eleven billion iterations shows a reasonably uniform distribution.

Closing Thoughts

In summary, although NED is the most recent of the polymorphic encryptors employed by virus writers, it is neither the most sophisticated nor the most capable. The bugs and the relatively large size of this module make it unlikely that it will be widely used, but there is already at least one virus which is known to use the NuKE Encryption Device. One final thought is that although NED is the latest polymorphic encryptor, it is not likely to be the last.

PRODUCT REVIEW 1

Oyster - A Pearl of a Product?

Megan Palfrey

BEST's Oyster 'computer immunisation system' claims to offer permanent protection against all viruses, known and unknown. *Virus Bulletin* readers will know that such tall claims are often reason enough for one to be wary of a product, but could it be possible that *BEST* has found the holy grail of the computer security world?

BEST's approach to the virus problem is, as the title suggests, file immunisation: that is, adding a protective 'shell' around a file, capable of 'healing' itself should the file become infected. Such techniques have formerly been prone to problems, and have certainly not offered a complete anti-virus strategy. Therefore, with my interest well and truly stimulated, I opened the package and began.

Component Parts

The review copy of *Oyster* was comprised of one write-enabled 3.5-inch diskette, an 181-page manual, a fast reference guide, six self-adhesive perspex badges with '*Oyster* Protected' inscribed upon them, and various other pieces of assorted 'bumph'.

The first unpleasant surprise I had upon reading the documentation was that *Oyster* is copy-protected. Normally (and quite rightly), *VB* does not review copy-protected software, but as several *Virus Bulletin* readers have asked for a review of the product, an exception was made.

Copy-protection goes against good security practice, as has often been stated in *VB*. The foundation of a good data security policy is that, in the event of a computer disaster, one can always start again from a backup and/or master disks. If software is copy-protected, this is impossible. For a company distributing security software to take this approach shows a lack of understanding of the issues involved: the best protection available for computers is a regular backup. *BEST* has since stated that it will change this policy - potential customers should check before purchasing.

The package is made up of a virus scanner and an immunisation program which claims to allow infected programs to 'heal themselves'. Each element has its own acronym: files are protected by COP (Coded *Oyster* Protection), which 'heals' infected files; the DATE feature (Direct Allow To Execute), which allows only *Oyster*-protected files to execute, and GOS (the Generic Oyster Scanner), which is able to 'scan and capture unknown viruses.' The continual use of these acronyms rapidly became confusing, and I was repeatedly forced to refer to the manual for clarification.

Documentation

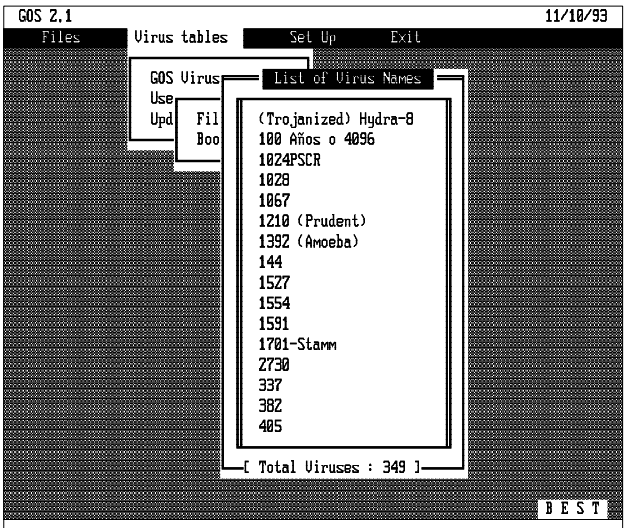
The documentation opens with an introduction to 'the *Oyster* way of life', and explains that '*Oyster* is the first product developed in the world that has found a universal solution to the virus problem.' After this extremely confident opening, the manual goes on to give a brief introduction to computer viruses and anti-virus software, before explaining the product's methodology and operation.

BEST is a Chilean company, and the original release of *Oyster* was in Spanish. Unfortunately, the documentation suffers from the fact that it is not a particularly well-written translation, and contains many irritating errors. Although there are appreciable difficulties translating such technical material, *BEST* would do well to spend more time tidying up the manual, in order to make it more readable.

The second problem with the manual is its love of acronyms (see above) - it is confusing and unnecessary. Why not call a spade a spade? However, these are small niggles compared to the manual's main flaw: it has no index. How a user is supposed to search for specific information in a publication of this size is not clear; such a shortfall must be rectified.

Installation

The installation routine is simple: the user simply places the *Oyster* diskette into the disk drive and follows the on-screen instructions. As *Oyster* works by immunising files against further attack, a vital part of the procedure is to scan the target disk for viruses. At no point (either in the manual or on-screen) is the user advised to clean-boot the installation machine. Still worse, the installation routine writes back to the master diskette (presumably as part of the copy protec-



With over three thousand viruses now known, *BEST's* scanner, *GOS*, is lagging well behind the current development curve.

tion scheme). Should a virus penetrate *Oyster's* own self-infection check, the user might unwittingly infect several machines. This should be changed.

The installation proceeded relatively smoothly, with the product explaining the changes it was making, and waiting for a keystroke to continue. There appears to be a feature somewhere in this routine: if no key is pressed, the routine times out and proceeds anyway. This needs to be fixed.

Fortunately, these quibbles aside, installing *Oyster* proceeded without flaw. Now, according to the documentation, my system was protected!

Functionality

Using Debug to examine a protected file revealed that it now had a JMP instruction at the start of the file and some 6K of code tacked to the end (the resultant file looked as though it had been infected with a parasitic virus!).

Was the newly-added code checksumming the contents of the host file? Tests showed that it was not, as I could alter files protected by *Oyster* and execute them without being alerted to this fact. This was a particularly worrying sign, as there are certain viruses which infect executables by overwriting code halfway through a file (eg 8888). *Oyster* provides little protection against such a virus.

The TSR component of *Oyster* can be configured so that only protected files are allowed to execute. As one of my tests, I overwrote the start of a protected file with the two bytes CD20h (the old CP/M 'terminate' instruction). Not only was the immunisation bypassed on the file, but *Oyster's* TSR still allowed the file to execute. With these unsettling results in mind, I started my tests, using a specially selected virus collection.

Under Fire

The acid test of any product is not whether the reviewer agrees with the methodology employed, but how the product functions. Will it prevent infection on a user's machine? As *Oyster* is a file immunisation product, there is only one way to test it: install the product, infect protected files, and see how it behaves.

The first virus against which *Oyster* was tested was Cascade. An infected file was copied onto the hard drive, and the standard *Oyster* file protect procedure of scan and immunise was carried out. The *Oyster* scanner, GOS, is not very accurate (see below) and, rather alarmingly, was unable to detect the Cascade infection.

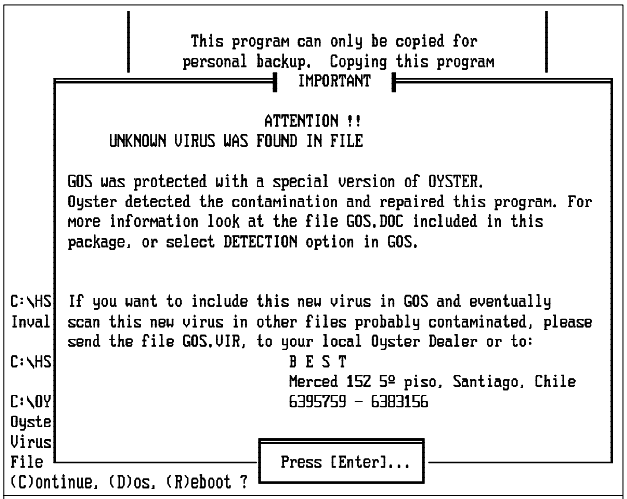
When the file was later executed, Cascade became memory-resident and active. From this point onward, whenever an *Oyster*-protected COM file was run, the virus infected it. However, when control was passed back to the host file, *Oyster* was capable of detecting the changes made, and returned the file to the state it was in directly before becoming infected - right down to the last bit!

This was a pleasing result, but there were some drawbacks. Most importantly, the machine now contained a file which was infected inside the extra code added by *Oyster*. Many virus scanners would now be unable to detect the virus, and the user would continually re-infect his machine. The manual does not deal with this eventuality.

The next sample used for testing purposes was a file infected with the 4K virus. This is a stealth virus, and is extremely difficult to detect once memory-resident. Could *Oyster* cope with this much tougher challenge?

The 4K tests started off well enough - the scanner identified the file as infected, warning that the test file contained the '100 Años o 4096' virus. However, as the average user would not scan *every* floppy brought near his machine, it is important to test whether or not the virus could infect files on the hard drive.

When an *Oyster*-protected file was run with 4K resident in memory, *Oyster* displayed the message 'Phase II virus detected in file GOAT14CO.COM. Run FASE2 program to remove the virus. (C) Continue (D) Dos'.



Whenever a virus infects one of the *Oyster* executables, *Oyster* captures the virus for later analysis.

Referring to the manual, I was instructed to select the 'Return to DOS' option, and run the FASE2 program. Upon execution this file was immediately infected by 4K, making it unusable. After several abortive attempts to use the rescue diskette to repair the problem, and with COMMAND.COM on the host machine infected, I was flummoxed. Was *Oyster* simply unable to cope?

Further tests showed that the problems were caused by 4K hanging the machine before the *Oyster* code was run. If the date of the PC was altered so that 4K did not crash, *Oyster* functioned correctly, and disinfected all files. The possibility of the virus malfunctioning does not seem to be dealt with either in the manual or in the programs stored on the rescue disk. Relying on the virus not to cause system instability is not good enough - the rescue disk should be able to cope with this eventuality.

Further tests showed that this failure was atypical: *Oyster* could cope well with other viruses (Jerusalem, SBC, 1575 etc) and in each case returned infected files to their original condition. These results are quite impressive, and show that under certain circumstances *Oyster* can function very well indeed. The problems encountered with 4K are still cause for concern however. Regardless of the cause, *BEST* needs to explain to users what to do in the event of failure.

As a final test, I tried infecting *Oyster*-protected files with one of the 'Trivial' family of overwriting viruses. The result was as I expected: the infected file was allowed to execute, and *Oyster* was unaware of the virus' actions.

Boot Protection

Boot sector protection was more effective. I attempted to change one of the error messages on the Master Boot Sector and was rewarded by a message from *Oyster*, stating that it had detected the presence of an 'altered boot sector' and would reboot the machine. Well done, *Oyster*.

The next test was to infect the hard drive with four different boot sector viruses: Spanish Telecom, Form, Quox and Swiss Army. In each case, *Oyster* spotted the infection successfully, replaced the infected boot sector and forced a reboot. The manual gives no explanation of how *Oyster* avoids being 'stealthed', but the most likely route is that it takes a 'snapshot' of important parameters at installation. However it functions, it is unobtrusive and efficient.

At one point during testing, *Oyster* seemed to stop protecting the boot sector, and I could infect and alter it at will. *Oyster* still claimed that the drive *was* protected, but this was clearly not the case. I could not get to the bottom of this problem, and was eventually forced to restart all tests.

Scanner Evaluation

Oyster's developers explained before the review began that it is not principally a scanning product, and that the scanning engine needs improving. Nevertheless, for an immunisation product to be effective, it is of paramount importance that the system is virus free before one tampers with files.

Oyster's scanning component, GOS, claims it does not work like traditional virus scanners, but 'uses *Oyster* technology to scan known viruses and capture unknown viruses'. As the manual is bereft of technical detail, I simply ran the scanner against the virus test-set.

The test results were disappointing: out of 72 infected files, *Oyster* identified only 41, missing such well known viruses as Cascade.1701, Flip and Yankee. Boot sector detection was rather better: *Oyster* missed only the comparatively new (although now in the wild) Quox virus.

The only way I could imagine the 'capture unknown viruses' might work was for the product to detect changes to protected files. Altering a file protected by *Oyster* and running the scanner on it showed that this was not the case. A more careful examination of the literature showed that the product simply waits for one of its own executable files to become infected. Since shipping the software, *BEST* has signed a deal to include the *Norman Data Defense Systems* scanner with the package. As this scanner has never been evaluated by *VB*, I cannot comment on its efficacy.

Conclusions

The makers of *Oyster* have made a brave attempt to use a different, less update-intensive method of virus protection, and this is laudable. Sadly, *BEST* has chosen to use a technique which is not very well-known, and has failed to explain it adequately. Using only the information provided in the documentation, *Oyster* did not live up to its extravagant claims of being the ultimate preventative package. Enhancing the programs on the rescue diskette and the contents of the manual would be a dramatic improvement.

Some of the features of the product are very impressive, and for certain machines (such as those in a very high risk environment) it may prove to be a valuable purchase. The newly announced bundling with the *Norman* scanner should improve matters quite considerably, but the product as sent to *VB* falls short of the mark. *Oyster* could (and should) be improved, as lurking beneath the murk lies some very interesting technology struggling to make itself seen.

Technical Details

Product: *Oyster*

Developer: *Business Engineering and Software Tools*, Merced 152, 4° Piso, Santiago, Chile. Tel. +56 (2) 639 5759. Fax. +56 (2) 639 8406.

Vendor: *Pacific Associates Ltd.*, Lapwing 400, Frimley Business Park, Camberly, Surrey, GU16 5SG. Tel. +44 (276) 62252. Fax. +44 (276) 62250.

Availability: IBM PC XT or Compatible with 640K of RAM, running MS-DOS version 3.3 or greater.

Version Evaluated: 2.1

Serial Number: 021CP0043

Price: £111 for a single PC. £5250 for a 100-user licence.

Hardware Used: 25MHz 80386SX *Opus Technologies* Desktop machine, with 4MB RAM, one 3.5-inch (1.44MB) floppy disk drive, one 5.25-inch (1.2MB) floppy disk drive, and a 100 MB hard drive, running *MS-DOS* 5.*0*.

The Test-Set

Viruses used for testing purposes: 1575, 2100 (C+E), 4K (C+E), 777, AntiCAD (C+E), Captain Trips (C+E), Cascade 1701, Cascade 1704, Dark Avenger (C+E), Darth Vader-200, Darth Vader-344, Darth Vader-409, Datalock (C+E), Dir-II, Dos Hunter, Eddie, Eddie 2 (C+E), Exebug Trojan 1, Exebug Trojan 2, Father (C+E), Flip (C+E), Hallochen, Invader (C+E), Jerusalem (C+E), Keypress (C+E), Liberty, Liberty-E, Macho (C+E), Maltese Amoeba, Mystic, Nomenklatura (C+E), Nothing, PCVrsDs (C+E), Penza, Pitch, Powerpump, SBC, Slow (C+E), Spanish Telecom 1, Spanish Telecom 2, Spanz, Syslock, Tequila, Todor, Tremor, Trivial-37.B, V2P6, Vacsina, Vienna, Virdem, W13_A, W13_B, Warrier (C+E), Whale, Yankee.

Boot Sector Infectors: Aircop, Beijing, Brain, Disk Killer, Form, Italian, Joshi, Korea, New Zealand 2, Quox, Spanish Telecom, Swiss Army.

PRODUCT REVIEW 2

Norton Strikes Again?

Dr Keith Jackson

The third major revision of *Norton AntiVirus* has just been released, and correspondingly, this is the third time that *VB* has reviewed the product (v1.0 was reviewed in January 1991, v2.0 in March 1992). *NAV3.0* is the offspring of *Symantec's* acquisition of the *Certus Novi* product: has this new blood improved *NAV's* pedigree?

Documentation

The only manual which comes with the product is an A5 book. This is well-written, thoroughly indexed, contains a decent glossary, and is easy to use. The sections which discuss the various types of virus, and how to deal with them, are very interesting (if somewhat brief), but I must confess I found the rest of the manual rather boring. Much of the information contained in the manual is of the type 'Select Print if you would like to print the entries'. Maybe I am the wrong person to judge this aspect of the documentation; new users may well appreciate its kindly style.

The documentation states that quarterly upgrades are available for a 'low yearly fee'. Note that this is somewhat less frequent than rival products - though *Symantec* claims that monthly updates are available if required.

Installation

Norton AntiVirus v3.0 was supplied on two 1.44 Mbyte (3.5-inch) floppy disks. This immediately excludes those users whose PC lacks such a drive. This aspect of the product has gone downhill as new versions have been introduced; v1.0 was supplied on both 5.25-inch and 3.5-inch floppy disks, and v2.0 was supplied on two 720 Kbyte, 3.5-inch floppy disks. In addition, the software has increased enormously in size: the hideously bloated version 3.0 requires 3.45 Mbytes of hard disk space!

I would have been pleased to see the problem of supplying various types of floppy disk discussed somewhere in the manual. It isn't. If ever a product was required on dual media, it is an anti-virus scanner, as it may be asked to inspect various types of PC within a single company.

The installation program proved very easy to use. It asks the user to choose a screen type, and then scans memory and all available disk drives. The user is asked whether he requires a full or a custom installation; then copying commences. After this is completed, automated changes to the startup files AUTOEXEC.BAT and CONFIG.SYS are offered. Finally, users are encouraged to create a 'rescue' disk, which contains information about the partition table and boot sectors of the hard disk.

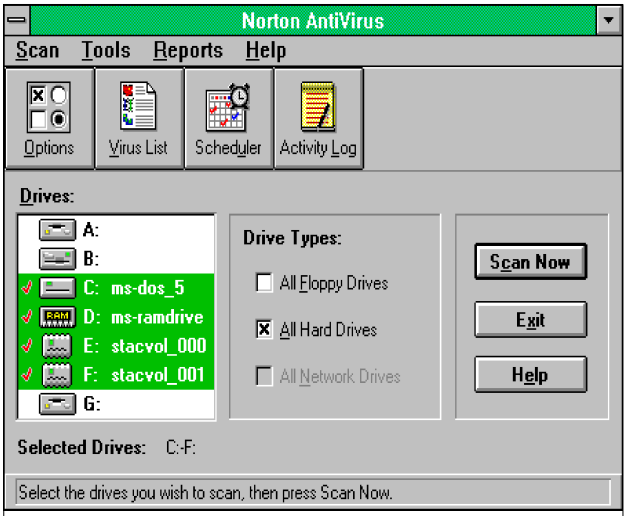
If the 'full installation' option is selected, all required *Windows* component parts are installed. Curiously, a *Windows* group is not created, although this is an explicit option as far as custom installation is concerned. Beyond this small detail I find it hard to fault the installation program.

Components

Once installed, *Norton AntiVirus* offers two distinct units: a memory-resident component (referred to as 'Automatic Protection') which can be tailored to monitor computer activity, and the main executable program which provides all features which are not memory-resident. A *Windows* scheduler is incorporated so that virus scans can be carried out at predetermined times.

Looking back through previous *VB* reviews of *Norton AntiVirus*, there has been a tendency to use silly names for certain aspects of the software. In v1.0, *Symantec* insisted on using the phrase 'virus definitions', when what it meant was the virus pattern/signature that their scanner used. The two main components of *Norton AntiVirus* used to be called Virus Intercept and Virus Clinic, both of which have (thank goodness) been dropped. The new buzzword is 'Virus Sensor technology', which is splashed all over the packaging. However, when it comes to explaining what this actually means (if anything), this phrase is missing from both the Index and the Table of Contents. Treat it as hype.

The discussion of phrases used by *Norton AntiVirus* is an introduction to the way that v3.0 uses the term 'inoculation' in a manner which I find downright misleading. I think I am correct in stating that other anti-virus software developers use this word to mean adding code to an executable program, so that the program can itself recognise that it has



The *Norton* user interface is well designed and allows quick and easy access to the plethora of functions provided by the product.

been altered. *Norton AntiVirus* uses the word 'inoculate' when it really means checksumming: the manual states quite clearly that 'Inoculation doesn't change a file or boot record'.

To illustrate how the total number of viruses known to *Norton AntiVirus* has increased, v1.0 knew of 115 viruses (142 variants), v2.0 of 341 (1006 variants), and v3.0 of 2350 (the distinction between variants seems to have been dropped). Information on these viruses is contained in the 'Virus List'. This is an excellent component which can display a description for each virus and includes comprehensive searching and printing facilities. The searches can be constrained as desired to encompass all, common, program, boot sector or stealth viruses.

Scanner Operation

The *Norton AntiVirus* scanner is very easy to use. When execution commences, a menu of all available drives is displayed, with hard drives highlighted. This default configuration can be changed to include any combination of disk drives, and options are available to scan a subdirectory or individual files. A raft of other options is available, all of which permit scanning to be carried out in any desired manner, including 'inoculation' of files to test whether or not they have been altered. *Norton AntiVirus* can also inspect files which are compressed in the ZIP format.

"This clearly illustrates that the scanner is not really doing a thorough scan - most likely in an attempt to avoid false positives"

When a scan commences, memory is scanned, followed by the Master Boot Sector, any Partition Boot Sectors, and selected files. Upon completion the results are displayed; the user is led through a list of problems found, and an activity log is written to disk. As with most other components, the content of these two log files can be tailored to suit requirements by configuring various options. One useful feature is that when a particular setup has been finalised, it can be password-protected against alteration. This completed, a user who does not know the password can still perform scans, but only in the prescribed manner. One curious feature is that scanning pauses if the left mouse button is held down - I have no idea why.

All in all, I cannot fault the way in which the scanner interface operates. It is clear, and easy to use. There are two user interfaces, the DOS and *Windows* versions, and the operation of each is remarkably similar. Indeed, apart from the imposed *Windows* graphical style, it would be hard for users to tell them apart. This is obviously a design decision and it works very well. The on-line help is very useful for both operating systems, but the *Windows* version is more extensive. This may well be more a comment on the *Windows* style of help than anything else.

Scanner Performance

Testing *Norton AntiVirus* against the viruses listed in the *Technical Details* section proved to be revealing. When I first ran the scanner it failed to detect 7 of the 223 test samples: 1260, Casper, Maltese Amoeba, Power Pump, V2P6, WinVir_14 and the boot sector version of Spanish Telecom. Not bad, but not outstanding.

For security reasons, I usually store my virus samples with a non-executable extension and use a scanner's 'do all files option'. In the light of these results, I decided to rename the infected files to an executable extension and redo the test. *Norton* was now capable of detecting four more viruses: 1260, Casper, Maltese Amoeba and V2P6.

When initially tested against the Mutation Engine samples, *Norton AntiVirus* detected just 6% of the 1024 test samples. Again, when the extension of the Mutation Engine samples was altered to COM, the detection rate increased to 100%. This clearly illustrates that the scanner is not really doing a thorough scan - most likely in an attempt to avoid false positives. What is the 'scan all files' option doing? This 'feature' needs to be discussed in the manual.

Scanning speed is more difficult to quantify, so comparative results with other scanners are included. In its default mode, the DOS version of *Norton AntiVirus* scanned the entire contents of my hard disk (800 files, 11.3 Mbytes of which were executable) in 33 seconds. If program files only were tested, 324 files were scanned and the scan time dropped to 18 seconds. By way of comparison, *Dr Solomon's Anti-Virus Toolkit* scanned the same hard disk in 17 seconds, and *Sweep* from *Sophos* took 20 seconds in 'Quick' mode, and 62 seconds for a 'Full' scan.

On the same hard disk, the *Windows* version of *Norton AntiVirus* took 35 seconds to scan all files, and 19 seconds for program files only. These figures are extremely close to those reported for the DOS version which, given the usual slowdown introduced by *Windows*, is very creditable indeed. All in all, *Norton AntiVirus* compares well with rival packages where scanning speed is concerned.

While a disk is being scanned, a horizontal bar indicating the percentage completed creeps across the screen. In certain circumstances (in both *Windows* and DOS), this horizontal bar had only reached about half its range when the software realised that it had completed its execution, and immediately zoomed up to 100%. I first complained about this 'feature' when I reviewed version 1.0; it is still there in v3.0, and it is still irritating. What is the point of a progress indicator which is incorrect?

Detection of Alterations

The 'inoculation' features of *Norton AntiVirus* (checksumming in more usual parlance) do not seem to be technically explained anywhere in the documentation. The best I could find was that 'critical information' (whatever that may be) is recorded about a file or boot record for future reference. Such

a trivial explanation is at best useless, and at worst, purposely hides the fact that only parts of each file 'inoculated' are checksummed. My own tests show that inoculation can detect the appearance of a new file, a change in file size, and bit level changes made at the start of the file. However, it does not detect any alteration made to the date/time stamp attached to a file, or single-bit changes made later in the body of a file. I do not know the point at which a file's contents cease to be monitored: technical information such as this is not supplied.

Memory-resident Component

The 'Automatic Protection' part of *Norton AntiVirus* (memory-resident is the more usual technical term) can be set up in many different ways, by choosing different options from within the scanner and then rebooting the computer. The file used to perform memory-resident tests (NAVTSR.EXE) can be loaded either as a device driver (from CONFIG.SYS), or as an executable program (from AUTOEXEC.BAT or the command line).

In its default state, the memory-resident component occupies 3.5 Kbytes of RAM, but this can soar to as much as 45 Kbytes if all possible options are selected. The documentation makes no mention of the execution overhead imposed by the memory-resident software. I moaned about this when reviewing v1.0, and again when reviewing v2.0, but inclusion of basic technical detail seems to go against the grain, and nothing has changed.

A performance overhead is inevitable, and users should be given factual information in order to make up their own minds as to whether or not the extra delay is justified. My own tests show that the time taken to copy 45 files (1.8 Mbytes) from one drive to another increased from 28 to 49 seconds when 'Automatic Protection' was introduced. This represents an overhead of 75%!

Other Features

The scheduler included with the *Windows* version of *Norton AntiVirus* is excellent, and permits scans to be set up on a one-time, hourly, daily, weekday, weekly, monthly or annual basis. Any number and any combination of scheduled scans can be specified, so that the way in which scanning is performed can be fine-tuned - an excellent utility.

Norton AntiVirus also includes features that permit files to be 'repaired' after they have been infected by a virus. Infected files should be replaced rather than repaired, so the efficacy of this feature has not been tested. If you are tempted to use it, the manual states that it only fails in 'rare instances.' Make of that what you will.

Conclusions

I have previously found *Norton AntiVirus* program design and ease of use excellent. It remains so. The *Windows* version is one of the most visually stunning anti-virus software packages I have seen (on a par with *Dr Solomon's* artwork). However, the worm turns here, and it has to be said that all this is merely garnish on the underlying functionality. It really does look as if the marketing men have taken over - for instance, when a scan under *Windows* is undertaken, the coloured logo of three arrows chasing each other rotates. Is this really necessary? Is it what the user wants? Is it what the user needs? My own view of the answers to these three questions is No, Yes, and No (people seem to like pretty flashing pictures).

Notwithstanding my diatribe against some of the 'more artistic' features of the user interface, it has to be said that the scanning features are approaching those contained in the best anti-virus scanner packages, both in terms of speed of scanning and accuracy of detection. *Norton AntiVirus* also offers the same scanning speed under either DOS or *Windows* - a feat few packages can manage. Virus detection has a few foibles but basically works well. These excellent features are marred by a lack of technical detail in the documentation (if this would frighten users, why not introduce a second volume?), and the use of marketing jargon in a misleading manner.

Symantec must be very confident that it is not going to have many disgruntled customers if it feels confident enough to offer a 30-day, no quibble, money-back offer. More software companies should put their money where their mouths are and include similar offers with their software products. I hope (and believe) that *Symantec* will profit from this: *NAV3.0* seems to be much improved on its parents, and *Symantec's* purchase of *Certus* may well pay dividends.

Technical Details

Product: *Norton AntiVirus*

Developer: *Symantec Corporation*, 175 W. Broadway, Eugene, OR 97401, USA, Tel: +1 (800) 444 7234, Fax: +1 (503) 334 3474, BBS: +1 (503) 484 6699 (or 6669).

Vendor(s): Most computer software retailers.

UK Support: *Symantec UK Ltd.*, Sygnus Court, Market Street, Maidenhead, Berkshire, SL6 4AD, UK, Tel: +44 (628) 592222

Availability: IBM PC, AT, PS/2 or 100% compatible running MS-DOS v3.0 or higher and/or *Microsoft Windows v3.0* or higher, with a 1.44 Mbyte (3.5-inch) floppy disk drive, 512 Kbytes of RAM and 4 Mbytes of hard disk space available.

Version Evaluated: 3.00

Serial Number: None visible

Price: £149 for single user copy. 50 user licence £3725.

Hardware Used: A 25MHz 486, with one 3.5-inch (720K) floppy disk drive, one 5.25-inch (1.2 Mbyte) floppy disk drive, a 120 Mbyte hard disk (split into three drives using *Stacker*), running under MS-DOS v5.0 and *Windows* v3.1

Viruses used for testing purposes: This suite of 143 unique viruses (according to the virus naming convention employed by *VB*), spread across 228 individual virus samples, is the current standard test-set. A specific test is also made against 1024 viruses generated by the Mutation Engine (which are particularly difficult to detect with certainty).

Full details of the test-set used are printed in *Virus Bulletin*, August 1993, p.19.

SUBSCRIPTION RATES

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, Oxfordshire, 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

END NOTES AND NEWS

***Symantec* is now offering users a *NetWare* version** of its *Norton AntiVirus* product, which runs under both *NetWare* 3.xx and 4, and provides file scanning on-access as well as on-demand. Prices start at £699 per server. Tel. +44 (628) 592222.

Patricia Hoffman's VSUM ratings for August: 1. *McAfee Associates ViruScan* V108, 96.4%, 2. *Command Software's F-Prot Professional* 2.09f, 95.9%, 3. *Sophos Sweep* 2.53, 92.8%, 4. *Dr Solomon's AVTK*, 91.7%, 5. *Safetynet's VirusNet* 2.08a, 90.8%. **NLMS:** *McAfee NetShield 1.52a* V108, 95.2%, 2. *Sophos Sweep NLM* 2.53, 92.9%, 3. *Dr Solomon's AVTK NLM* 6.54, 88.0%, 4. *Command Software's Net-Prot* 1.00s, 69.9%, 5. *Cheyenne's InocuLAN* 2.0/2.18g, 65.8%.

Anti-virus through the ether! Got a virus? Nobody else can help? Why not turn on your television and download a copy of *Dr Solomon's Anti-Virus Toolkit*? In one of the first deals of its kind, S&S International has signed an agreement with the Italian state television channel *RAI*, enabling anti-virus protection through teletext transmission. In order to receive the transmissions, a copy of *S&S's Security Kit* is needed, along with a low-cost expansion board for the PC. Tel. +44 (442) 877877.

Fifth Generation is to **cease production of its two anti-virus products,** *Untouchable* and *Search and Destroy*, following the company's purchase by *Symantec*. The two products will be discontinued during the first week in November, but existing customers will continue to be supported by *Symantec* for an as yet undecided period. At that time, they will be offered an upgrade to *Norton AntiVirus 3.0* at the normal upgrade price.

A former *Princeton University* doctoral student has been **fined $500 for hacking into the school's computer system**, according to a report in the *New York Times*. The student, Luo-Qi, pleaded guilty after being guaranteed that he would not be sent to jail - the maximum penalty for the charge was a $1000 fine and six months imprisonment. He has since been expelled from the university.

The *DTI*, in association with the *British Standards Institute*, and a number of other companies, has released a 'Code of Practice for Information Security Management', designed to help organisations secure their IT resources. It is hoped that the code will be developed into a British Standard by the middle of 1994.

Which book provides **an instant one-shot reference on computer viruses** and anti-virus software? It can only be the *Survivor's Guide to Computer Viruses*, available from *Virus Bulletin*, priced £19.95. For further information contact Victoria Lammer. Tel. +44 (235) 555139.

Triangle Software Division, of *Datawatch Corporation*, has announced the **upgrade of its anti-virus product, *Virex* for the PC**. The program now sports a new installation routine and enhanced TSR capabilities, and can be purchased for $49.95. Tel +1 (919) 490 1277.

Oxford University has had another outbreak of the Spanish Telefonica virus, more than two and a half years after the virus first appeared at that institution. Two departments are reported to be affected. The university has also had several incidences of the EXEBUG virus, which is particularly worrying, because the virus prevents users clean-booting their systems on certain machines.

Sophos' next two Computer Virus Workshops are scheduled for 24th-25th November and 26th-27th January. Prices are £295 for one day, and £545 for two. Contact Karen Richardson on +44 (235) 559933.

A Cautionary Tale: Fridrik Skulason has recently been sent a very interesting request for help ... from a virus author. A Brazilian computer science student wrote a new boot sector virus, purportedly to show his friends how a virus worked. It subsequently escaped into the wild. Unfortunately for the virus author, he had included his own name and details in the virus. Understandably, he is not popular among Brazilian computer users, and is desperate for a disinfection routine to be included in the next release of *F-Prot*. Potential virus writers be warned!