October 1991 ISSN 0956-9979

VIRUS BULLETIN

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

CONTENTS

CONFERENCE EDITION

The First International Virus Bulletin Conference

In a break with tradition, this month *VB* tentatively introduces some photographs into its editorial. These pictures were selected from the hundreds of snapshots taken at the *First International Virus Bulletin Conference* held last month on the Channel Island of Jersey. On September 12-13th 1991 some 150 delegates and twenty speakers from four continents assembled at the *Hotel de France* in St. Helier, Jersey. Before expanding upon the themes of the conference itself it is the editor's beholden duty to say thank you to:

- the delegates: a formidable and eminent audience.
- the organisers: the supremely efficient team of Petra Duffield, Karen Richardson, Lynne Whitehead and Sarah Hood.
- the speakers: who prepared their presentations with great care and, in many cases, had to travel thousands of miles to attend.

Conference Themes

The single loudest appeal from delegates at this conference (nearly all of whom were from commerce, industry, government or military organisations) is that the anti-virus community (if such exists at all) must start to see the wood for the trees, i.e. a wider perspective on this problem is required. To paraphrase Steve White of *IBM*: 'We all of us know how to protect one computer from rogue software, the question is how do you protect a *whole user community*?' Anti-virus software developers, in addition to providing diagnostic tools must start formulating complete, even bespoke, strategies for their customers and provide training and consultancy. It is now evident that corporate end-users of defensive software (or hardware) are increasingly demanding an augmented service from suppliers. (Incidentally, many software developers were present including representatives from *Central Point*, *Symantec*, *Software Concepts Design*, *Sophos*, *Cybec Pty* and *BRM Systems*.)

There was evident criticism of the research community for failing to identify and explain the essential technical trends which will inevitably affect long-term defensive strategies. Explaining the redundancy of certain worn-out and ineffectual technologies to management is extremely difficult. Outlining the limitations of obsolescent techniques (as opposed to

The 'A' Team. (Left to right) Jim Bates (*Bates Associates*, UK), Ross Greenberg (*Software Concepts Design*, USA), Richard Kusnierz (*Network Security Management*, UK), Dr. Jan Hruska (*Sophos*, UK), Fridrik Skulason (*Technical Editor, Virus Bulletin*, Iceland), Detective Constable Noel Bonczoszek (*City & Metropolitan Police, UK*), Joe Norman (*SGS-Thomson*, UK), Steve White (*IBM T. J. Watson Research Center*, USA), Prof. Eugene Spafford (*Purdue University*, USA), Edward Wilding (*Virus Bulletin*, UK), Vesselin Bontchev (*University of Hamburg*, Germany), David Ferbrache (*Defence Research Agency*, UK), Dr. Simon Oxley (*National Power*, UK), John Norstad (*Northwestern University*, USA), Scott Emery (*Digital Equipment Corporation Inc*., USA), Squadron Leader Martin Smith MBE (*Touche Ross Management Consultants*, UK), Ken van Wyk (*CERT*, USA).

obsolete methods) is even more difficult. Software 'solutions' are often rushed into effect without sufficient care and planning, only to be discarded at a later date (often following considerable financial outlay) due to their unsuitability.

These general criticisms will start to be redressed by *VB* in the coming months. The general message appears to be to keep the journal practical and balanced (between the technical and managerial) and, at all costs, avoid the more futile academic exercises to which the subject of computer viruses so often gives rise.

A not entirely unexpected message from the conference is that the corporate technician or manager is not interested in such ethereal concerns as bit changes in memory or minor code variations or modifications. This is unfortunate because the research community is currently fascinated by such things. Interestingly Detective Constable Noel Bonczoscek of *New Scotland Yard* intimated that without such precise identification methods, his job of collating and presenting evidence would become impossible. A conflict of interest is readily apparent.

After-dinner speaker and associates half-way into disassembling the notorious two-byte virus. Back row (left to right): Petra Duffield (*Virus Bulletin*), Karen Richardson (*Sophos*), Wing Commander Amanda Butcher (*Ministry of Defence*, UK). Front row: Lynne Whitehead (*Oxford University*), Squadron Leader Martin Smith (*Touche Ross*), Julie Hollins (*WH Smith News*).

Some Presentations in Precis

IBM provided the most intensively research-based presentation of the two days. Studies have been undertaken at the *T. J. Watson Research Center* in New York State into the spread of different virus samples worldwide. In common with the findings of *Virus Bulletin* (*VB*, September 91, p. 14) *IBM's* statistics show that a few viruses account for the most incidents - the New Zealand (Stoned) virus accounting for approximately 28 percent of all incidents. The 'promiscuous software society' alluded to by the 'virus industry' is proving to be a myth - software sharing is invariably localised and limited. Those theoreticians who talk of epidemics, universal contagion and the end of personal computing, take note!

Central reporting of incidents, diagnostic software and immediate response (the essential components of a defensive strategy) are proven as the most effective anti-virus approach. *IBM's* research staff are currently automating the development of virus-specific detection software - results from these experiments put *VB's* attempts to provide reliable search patterns to shame. *IBM* minimises false positive indications with the use of a vast library of user supplied software running into gigabytes. To quote Steve White: 'In a company with 250,000 PCs, a single false positive can mean three days solid tied to the telephone.' White warned against the term 'exponential' - nothing that his team has observed in the virus field comes close to being exponential.

The most original (and complex) paper was provided by Yisrael Radai of the *Hebrew University of Jerusalem*. Radai contends that cryptographic checksumming employing DES or an ISO standard algorithm is effectively 'overkill' for managing the computer virus threat. The CRC algorithm is just as effective as well as being easier to implement and far faster in its execution.

According to Radai, the confusion on this point arises from the cryptographic obsession with confidentiality; CRC is more vulnerable to cryptographic attack than DES but this point is irrelevant when choosing an integrity checking algorithm to counter indiscriminate computer virus infection.

Joe Norman of *SGS-Thomson* described the corporate anti-virus strategy which he has devised. He insisted that detection software must be compatible with the nomenclature and terminology adopted by *VB* so that information can easily be cross-referenced.

Describing an initial integrity check at one site where 4,000 hard disks and diskettes were scanned, he reported that some 2 percent were found to be infected.

A video covering computer virus prevention had been adopted for educating employees - this had proved effective but has also been costly in terms of time (5,000 users x 1 hour for each employee's induction works out at about 2.5 man years in total).

Norman cautioned security managers against Draconian disciplinary measures. He would rather have a virus incident reported than have some non-technical end-user attempt to disinfect the machine and subsequently compound the damage. (*This new 'softly softly' policy is a total reversal of the 'hang 'em and flog 'em' tactics of yester-year*.)

Dr. Simon Oxley of *National Power* reiterated this theme: 'We don't want to drive this problem underground. It's common to be over-zealous and publish policies which threaten instant dismissal for anyone found infecting a PC with a virus. A better approach is to encourage rapid and full notification of suspected virus problems without the threat of retribution. Incidents can then be diagnosed and dealt with correctly.' Severe disciplinary measures should be reserved for instances where there was flagrant disregard for procedures or the deliberate introduction of a virus.

Oxley also alluded to the economics of virus protection:

> *'A quick back-of-an-envelope calculation can be done for a company with around 1000 PCs. An initial reaction might be to equip all these PCs with a commercial anti-virus package. This might cost £50,000 at £50 per PC. The package could be invoked on every PC boot to carry out a check or a scan lasting maybe one minute. If each PC is booted once a day we are spending 16 hours (two man-days) every day checking for viruses, at an average cost in lost time of perhaps £200. During the first year of operation, this mechanism will therefore cost £100,000. In addition to this we have the cost of training users and support staff in the use of these packages and this too could be considerable*.'

Vesselin Bontchev provided an insight into the Bulgarian 'virus factory'. A disillusioned army of programmers trained by the communist regime to break software copy-protection schemes had turned its attentions to virus writing. The low-level programming methods (sometimes described as 'on-the-metal' programming) involved in copy-protection were readily adaptable to the development of virus code.

Some 80 Bulgarian viruses are causing disruption within Bulgaria itself. According to John McAfee approximately 10 percent of all infections in the USA are caused by Bulgarian viruses.

Jim Bates discussed the process of virus disassembly - not an easy task within the time frame of forty-five minutes. The vital point (and one to which many virus writers seem entirely oblivious) is that *any functioning computer program can be reverse engineered back to its original (human intelligible) instructions which can then be analysed to determine its actual functioning*. (Ross Greenberg re-emphasised this point 'All viruses can be disassembled. If a CPU can do it - and it must in order to run the virus - then a human can do it, too, albeit slower and usually with a good deal more foul language.')

Bates covered the essential tools and steps necessary to the task. Less obvious requirements were copious quantities of coffee and cigarettes, insomniac colleagues who could assist with technical enquiries at three o'clock in the morning and a wife (or husband) with the patience of a saint.

The major threat at the moment was the dissemination of source code: 'If object code is a bullet then virus source code is a loaded gun!' Having immersed himself in virus disassembly for nearly four years Bates concluded: 'The more experience you gain, the more you realise just how much you *don't* know'.

Gala Dinner. Delegates Esther Armbrust (*BASF AG*) and David Henretty (*Apricot Computers*) demonstrate static analysis and dynamic decompression utilities.

Gala Dinner. (*Left to right*) Vesselin Bontchev, Helen White, Steve White, John Norstad and Gene Spafford assembled (but executing less quickly than normal).

Ross Greenberg, discussing MS-DOS anti-virus tools and techniques, described his dismay each time he reads the dreaded entry 'no search pattern is possible' in the *VB Table of Known IBM PC Viruses*. 'It means that anti-virus researchers have to stop attacking each other in public forums and actually get to work.' Encryption and 'armour' were obstacles, but never proved insurmountable. 'The best news is that it's not always necessary to disassemble the full virus in order to detect it, disable it, inoculate against it, or even disinfect a file.' Greenberg concluded that the public will continue to misuse the defensive tools at its disposal.

John Norstad, author of the widely used *Disinfectant* anti-viral utility provided an introduction and overview to the Macintosh virus problem. Macintosh users are far fewer than those of IBM PC compatibles (there are approximately three million Macs in use compared to some fifty million PCs), which means that the user community is relatively closer and more united. Certainly there is none of the bickering, infighting and political intrigue currently prevalent in the PC anti-virus industry. Norstad described an extraordinary situation on the Macintosh whereby the nVIR virus interbreeds and spawns different generations of offspring. Watching this process in action led to an 'uneasy sense of voyeurism'.

Ken van Wyk of the *Computer Emergency Response Team* (*CERT*) addressed network security, specifically referring to the Unix environment. Van Wyk is responsible for issuing security advisories for *Internet* users (some half a million hosts combine to make the *Internet* the largest network in the world). There are political considerations inherent to such a sensitive role - tact and diplomacy are essential when dealing with system vendors and users as diverse as the military and academia. One vital consideration when threatened by system intrusion is to keep the catalogue of known and existing vulnerabilities off-line!

Professor Gene Spafford (*Purdue University*, USA) successfully demolished the common misconception that Unix as an operating system is insecure - users had grown to expect openness and convenience. As with all computer systems, 'user friendly' can mean 'attacker-friendly'. Configuration controls are available under Unix but implementing them was liable to trigger a wave of protest among users familiar with an unrestrictive environment.

David Ferbrache of the UK's *Defence Research Agency* demonstrated that traditional Orange Book methods were wholly inadequate to countering the virus problem. The US *Department of Defense* Orange Book was principally concerned with confidentiality whereas viruses impact upon integrity and availability. Malicious software introduced at an untrusted level is likely to be executed by users with restricted or even full system privileges.

An Informal Initiative

As with any conference, much of the real work was conducted away from the bright lights of the conference hall and in the darker recesses of the bar. Over pints of beer, a number of informal arrangements were agreed between various researchers and agencies. The priority among the anti-virus community is to cut incident response times, increase cooperation, the sharing of binary code, disassemblies and tools. The means and methods to accomplish these objectives are agreed.

Informal cooperation will be the key to the success of this initiative - too many organisations with contrived acronyms have been formed, which once furnished with self-appointed committees, have become stuck in a mire of red tape and soul-searching.

All Fun and No Play...

...makes Jack a dull boy. Many thanks to Petra Duffield and Karen Richardson for arranging the spectacular gala dinner, to Jim Bates for his extempore saxophone accompaniment to the dance band, to Gene Spafford for his helium-induced Donald Duck impersonations, Martin Beney for providing the best photograph of the conference (regrettably not clear enough for publication), and to the *Hotel de France* for supplying its beautiful schooner *'Meriliisa'*, aboard which speakers and organisers assembled for some post-conference recovery.

Finally, *VB* looks forward to renewing acquaintances with all who attended this year's event, at the *Second International Virus Bulletin Conference* in 1992.

KNOWN IBM PC VIRUSES (UPDATE)

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

Note: The standard policy starting with this edition is to publish **24 byte patterns** for each entry as a means to reducing the incidence of false positive indicators. Some of the entries shown in this month's table are shorter than 24 bytes due to their having been analysed prior to this policy being adopted.

Type Codes

C = COM files **E** = EXE files **D** = Infects DOS Boot Sector (logical sector 0 on disk)
M = Infects Master Boot Sector (Track 0, Head 0, Sector 1) **N** = Nor memory-resident after infection
R = Memory-resident after infection **P** = Companion virus

Seen Viruses

191 - CN: A very simple virus with no side effects.

```
191             8BD7 B902 00B4 3FCD 2181 3D07 0874 DF33 D233 C9B8 0242 CD21
```

656 - CN: Triggers on 14th of any month except January or any day in April. Overwrites first 80 sectors of drive C:.

```
656             ACB9 0070 F2AE B904 00AC AE75 EEE2 FA5E 0789 7C17 89F7 83C7
```

CZ2986 - CER: A network-specific virus received from Pavel Baudis, *ALWIL Software*, Czechoslovakia. The virus is reported to be in the wild in Eastern Europe although this has not been confirmed. This is a parasitic (appending) virus based on Old Yankee. Infects COM and EXE files via load and execute function requests. Infective length varies from between 2971 and 2986 bytes. There is no trigger routine. The virus examines LOAD and EXECUTE requests for execution of LOGIN.EXE and subsequently collects User ID and password information requested by that program. Virus compares these with an internal table and maintains storage of the most recent 15 User ID/Password combinations in encrypted form. The inference is that any infected program may be collected and interrogated by an informed user who would then know the relevant network IDs and passwords.

```
CZ2986          9074 13EB 3090 BF6F 09E8 3300 AA3C 6F90
```

Horse 8 - CER: No search pattern is possible, awaiting analysis.

Liberty boot: This amended pattern detects both file and boot sector infections caused by the Liberty virus. (*VB*, Oct 91)

```
Liberty boot    B931 2833 D2CD 130D BB5C 0653 CB2E 803E BC06 0A74 4633 C08E
```

Liberty 1186 - CR: Awaiting analysis.

```
Liberty         1186 A02E 01CD 2183 FBFF 7431 B403 33DB CD10 890E 1601 B401
```

Rage - CR: Encrypted virus which overwrites sectors 0 through 225 of hard drives C: to Z: on the 13th of the month. Issues an 'are you there call' to locate VIREXPC.COM in memory and, if present, restores control to the host program. (*VB*, Oct 91)

```
Rage            8A24 518A C8D2 C459 8824 FEC0 46E2 F1
```

Semtex - CR: Infects every COM file opened or executed. The name of the virus suggests that it is destructive. Awaiting analysis.

```
Semtex          8B3E 8400 268B 1686 008E C226 813D 9C50 7519 BAFF FFFC 8D36
```

Sov1 - CN: Awaiting analysis.

```
Sov1            F3A4 E8D4 01E8 8C01 7303 E8C0 01E8 1900 E8DA 0107 1FCB 2A2E
```

Sov2 - CN: Awaiting analysis.

```
Sov2            E80D 02E8 9801 3C00 740D E8B4 013C 0074 06E8 D801 EB04 90E8
```

Spanish Telecom 2 - MCER: A new variant of the Spanish Telecom virus submitted by *RG Software Systems*, USA.

```
Span Telecom 2  8A0E EB00 BE70 0003 F18A 4C02 8A74 03C3
```

Virus 9 - CN: Infects all COM files in current directory and recursively to the root directory. Infected files contain virus code at the end of the file. There are no side effects.

```
Virus 9         3ECD 21B4 4FCD 2172 02EB B0B4 3BBA 7501 CD21 7202 EB9C CD20
```

SCANNER UPDATE

Mark Hamilton

The Stamina Test

Stamina is perhaps the single most necessary quality in producing virus-specific detection software. If an aura of complacency crept in among the scanner manufacturers after last month's favourable Scanner Update then the following exercise will come as a rude awakening to many.

When comparing virus scanners, I have concentrated on testing each product's ability to detect a large number of different viruses. The library used for these tests contains, where available, a single COM and a single EXE infection for each virus. This strategy works well for most viruses, but does not adequately test the scanners' ability to detect self-modifying specimens.

For this category, a new strategy has been devised.

With the recent and ongoing problems caused by Spanish Telecom and Tequila, coupled with Washburn's continuing development of his V2Pn series, it is important that the anti-virus product vendor community demonstrates its commitment by developing and incorporating algorithms to detect such viruses in their respective products.

As many of these viruses can not be detected using simple hexadecimal search patterns, a number of anti-virus developers employ hard-coded 'identities'. Several of the more recent scanners use 'smart' detection engines which means that the identities for encrypting viruses can be included in their search database; since the base product doesn't need to change, such vendors should be able to react quickly and include the detection routines for these viruses more readily than those stuck with the more antiquated engines.

One exception to this trend is Whale, for which simple hexadecimal search patterns were obtained for all thirty variants by *Sophos*, following cooperation between Peter Lammer and Jim Bates. *Sophos* made these patterns available to other anti-virus product vendors.

Detecting such viruses imposes several demands on the scanner writer: principally, the dedication to sit at a PC for long periods of time to disassemble these brutes, annotate the resulting disassembly and determine a means to incorporate detection algorithms into the product. This is taxing in terms of time and resources.

Combating encrypting and, to a lesser extent armoured viruses, is the ultimate demonstration of stamina. Such viruses will inevitably become more numerous over the next year and scanner technology will have to adapt accordingly.

Testing

I chose a total of twelve viruses and generated a number of infections of each.

These twelve strains are split into three logical groups:

Group 1: These viruses are currently in the wild. All scanners should be able to detect **all** the infections.

Flip	20 COM and 20 EXE infections
Spanish Telecom 1	5 COM infections
Spanish Telecom 2	4 COM infections
Suomi	20 COM infections
Tequila	50 EXE infections

I deliberately chose to include both variants of Spanish Telecom as each has a different geographic impact. Currently (September 1991), Spanish Telecom 1 is prevalent in Europe, while Spanish Telecom 2 is reportedly causing infections in the United States. **British readers should note that reliable detection of Spanish Telecom 1 is absolutely vital due to the spread of this destructive virus within the UK**.

Group 2: These viruses all belong to the same Bulgarian family and samples have been available for quite some time. They are reportedly 'at large' in parts of Eastern Europe and the Soviet Republics. Mechanisms for detecting these viruses are desirable in any scanning program.

1226	20 COM infections
Evil	20 COM infections
Phoenix	20 COM infections
Proud	20 COM infections

Group 3: This group is generally regarded as 'experimental'. Like the Group 2 viruses, samples have been available for quite some time but unlike either Group 1 or Group 2 viruses, there have been no reports of these in the wild. Nevertheless, the ability of a scanner to detect the Group 3 viruses does increase one's confidence in the product. Admittedly, the Group 3 exercise is somewhat academic.

V2P1 (1260)	20 COM infections
V2P6	50 COM infections
Whale	11 COM infections

The number of infections, as can be seen, is generally 20 or 50. Except, that is, in the cases of Spanish Telecom and Whale. The nine Spanish Telecom 1 and 2 infections proved

more than enough to trip up the majority of products. In the case of Whale, I was only able to generate 11 unique infections.

Each set of infective samples was placed on separate floppy disks. One of the prime requirements of this test-suite was that for each virus, each of the infections was unique. This was achieved by infecting a series of identical sacrificial goats and, using a 32-bit CRC checksumming utility, comparing checksums - infections with identical checksums were discarded.

All the test files were infected by the virus on an infected machine, to duplicate real-world conditions. All the scanning was undertaken under 'clean' conditions i.e. the machine was booted from a clean system diskette.

Since in these tests I am measuring each scanner's reliability, it must be able to detect all the infective samples of each virus to pass. As far as the 'reliability score' is concerned, a scanner is deemed capable of detecting the particular virus if it successfully detects all of that virus' progeny.

Therefore, rather than provide statistics detailing the number of viruses detected by the various scanners, I simply provide 'Pass / Fail' tables and a narrative account of each scanner's performance. Readers can then consult the tables to determine whether the scanner(s) under consideration can detect a particular virus.

As is current practice with these comparative evaluations, the latest version of each product is used (to my knowledge as of 20th September 1991); however, I must emphasise that I should be grateful if vendors could ensure that I am in possession of their latest versions as they are released.

With the exception of *PC Enhancements*, all the products which are regularly tested for the Scanner Update column, were included in this reliability test.

PC Enhancements' PC-Eye scanner was deliberately not included, because the company makes no claims that its product can detect any of these viruses. Having said that, I do understand that the company is to re-think its strategy and include detection mechanisms for encrypting viruses presently. Of the viruses featured in this test, *PC Eye* detected Spanish Telecom 1 accurately and reported a bad date/time stamp in all samples of V2P6.

These 'pass/fail' tables (see pp. 9-11) indicate *reliable* versus *unreliable* detection but do not indicate the extent of failure where it occurs. More detailed statistics are available upon request from *Virus Bulletin.*

In a supplementary test, incidents of the Cascade virus (12 COM infections) the 2100 virus (2 COM, 3 EXE), the 4K virus (3 COM, 4 EXE) and the Jerusalem virus (27 COM, 35 EXE) were all detected with absolute reliability by all of the scanners featured in this update.

The Results

Central Point Anti-Virus - Version 1.0

The product which is tested here including the recent signature update available on *Compuserve* fared very poorly against these encrypting viruses. It failed to detect all generated samples of 1226, Evil, Phoenix, Proud, Spanish Telecom and Tequila. It only managed to find 14 (of 50) V2P6 samples and 8 (of 11) Whales.

FPROT - Fridrik Skulason - Version 1.16

I was only able to test version 1.16 of this product, which was issued in June 1991 although version 2.0 is now available. For some inexplicable reason, the later version refused to load and locked my PC. Nevertheless, version 1.16 did exceptionally well detecting all the samples, except those of Spanish Telecom 2 where it missed all four infections.

FindVirus - Dr Solomon's Anti-Virus Toolkit - Version 5.11

This version of the *Toolkit* is somewhat elderly, but no later version has been made available for tests. [*Apologies to Dr. Solomon and Mark Hamilton. Editor's oversight.*] Somewhat alarmingly it missed all samples of the Spanish Telecom virus. Less significantly it failed to detect one sample of V2P6. It gained a clean bill of health on all the other samples. [*Surprised by this product's failure to detect Spanish Telecom, the editor tested Version 5.15 against four different generations of Spanish Telecom 1 - all COM files were passed as clean*.]

HTScan - Harry Thijsen - Version 1.15

This scanner identified all the samples of 1226, Evil, Phoenix and Proud as 'P-1', which might cause some confusion. Accurate identification of viruses is not one of this program's strong points.

It failed totally to detect any samples of Spanish Telecom 2, V2P1 and V2P6, but found everything else.

Norton AntiVirus - Version 1.5

This scanner failed to detect reliably three strains: it found none of the Tequila or Spanish Telecom samples (all of which are in the critical Group 1 test-set). Of less significance, *NAV* missed one of the V2P6 samples.

ScanV80 - McAfee Associates

McAfee Associates' product proved to be one of the most reliable. It did not fail to detect a single generation of any of the encrypted viruses included in the test-set. The product gains a 100% accuracy rating in this test.

Sweep - Sophos - Version 2.29

Sweep failed to detect Spanish Telecom 2 which has been reported in the wild recently in the United States. *Sweep* detected all the other samples. The Proud virus was identified as Phoenix, while V2P1 was mistakenly identified as V2P6.

TBScan - ESaSS - Version 2.8

In common with *HTScan*, *TBScan* identified 1226, Evil, Phoenix and Proud as 'P-1'. This is not surprising as both scanners use the same virus data file prepared by Jan Terpstra. *TBScan*, however, failed to detect 9 of the 50 samples of Tequila and it didn't find any of the V2P1 and Spanish Telecom 2 samples.

TBScan appears to have a very rudimentary engine flagging all fifty V2P6 samples as 'suspicious'.

VIRx - Ross Greenberg - Version 1.7

This version is far more up-to-date than Ross Greenberg's commercial product distributed by *Microcom*, but even this scanner failed to find all the infections. Of those where it found some infections, *VIRx* failed to find 17 samples of 1226, 10 samples of Evil, 16 samples of Phoenix, one sample of V2P6 and one of the Whales. It totally failed to detect any samples of Proud and Suomi. It's all very well producing a scanner that can scan inside executables compressed with *DIET* and *PKLITE* - useful gimmickery - but of little benefit to users if the detection of encrypting viruses is overlooked.

	Flip	Suomi	Tequila	Spanish Telecom 1	Spanish Telecom 2
CPAV	Pass	Pass	Fail	Fail	Fail
F-FCHK	Pass	Pass	Pass	Pass	Fail
FindVirus	Pass	Pass	Pass	Fail	Fail
HTScan	Pass	Pass	Pass	Pass	Fail
Norton AV	Pass	Pass	Fail	Fail	Fail
ScanV80	Pass	Pass	Pass	Pass	Pass
Sweep	Pass	Pass	Pass	Pass	Fail
TBScan	Pass	Pass	Fail	Pass	Fail
VIRx	Pass	Fail	Pass	Pass	Pass
Virscan	Pass	Pass	Pass	Fail	Fail
Viscan	Pass	Pass	Pass	Pass	Pass
Vi-Spy	Pass	Pass	Fail	Pass	Fail
VPCScan	Fail	Fail	Fail	Fail	Fail

Table 1. All of the encrypting viruses in this table have been identified in the wild in the United Kingdom with the single exception of Spanish Telecom 2 which has been identified in the United States. For the purposes of this review the reliable detection of Tequila and Spanish Telecom 1 is considered mandatory.

	1226	Evil	Phoenix	Proud
CPAV	Fail	Fail	Fail	Fail
F-FCHK	Pass	Pass	Pass	Pass
FindVirus	Pass	Pass	Pass	Pass
HTScan	Pass	Pass	Pass	Pass
Norton AV	Pass	Pass	Pass	Pass
ScanV80	Pass	Pass	Pass	Pass
Sweep	Pass	Pass	Pass	Pass
TBScan	Pass	Pass	Pass	Pass
VIRx	Fail	Fail	Fail	Fail
Virscan	Fail	Fail	Fail	Fail
Viscan	Pass	Pass	Pass	Pass
Vi-Spy	Pass	Pass	Pass	Pass
VPCScan	Fail	Fail	Fail	Fail

Table 2. Encrypted viruses shown in this table are reported to be in the wild in the Eastern Bloc and Soviet Republics.

All of these samples emanate from Bulgaria. Penetration into Western Europe and elsewhere is possible.

Considering the limited geographical spread of these specimens, their reliable detection should be considered desirable but not essential.

Interestingly, *IBM's VIRSCAN* does not detect these samples, which may indicate that they have definitely not been encountered in the United States.

Virscan - IBM - Version 2.1.2

As far as this group of viruses is concerned, *IBM's Virscan* did not perform too well. It consistently detected the widespread Tequila virus but failed reliably to detect the equally ubiquitous Spanish Telecom 1 virus.

It managed to find one of the four Spanish Telecom 2 infections but none of the 1226, Evil, Phoenix, Proud or V2P6 infections. The *IBM T. J. Watson Research Center* monitors 'real world' virus outbreaks - it is possible that the Bulgarian encrypting viruses have not been seen outside Bulgaria.

Viscan - Total Control - Version 3.25

This software correctly detected and identified all of the encrypted viruses and generations. The developer informs me that he has only needed to hard-code a recognition identity for V2P6; all the other viruses are recognised by the main scanning engine.

Viscan (which is marketed by *Total Control* but developed and maintained by *Bates Associates*) and *McAfee Associates' Scan* are the two scanners which pass this particular test without any failures.

Vi-Spy - RG Software - Version 7.0

A note in this software's documentation warns that it does not detect Tequila in program files (whereas it will detect it in boot sectors and in memory), so it was hardly surprising that it didn't find any of that virus' parasitic progeny.

Vi-Spy also missed one of the four Spanish Telecom 2 infections, but detected and correctly identified all of the other encrypted samples included in this test.

VPCScan - Microcom - Version 1.2

As mentioned above, this software lags behind the public domain *VIRx* scanner from the same author. *VPCScan's* performance on these encrypting viruses was disastrous. It only managed to detect V2P1 reliably, missed one of the Whales and failed to detect any of the other viruses with any degree of reliability .

The product detected two of twenty 1226 infections, ten of twenty Evil infections, four of twenty Phoenix infections, and no Proud, Flip, Suomi, Tequila V2P6 or Spanish Telecom infections.

	V2P1	V2P6	Whale
CPAV	Pass	Fail	Fail
F-FCHK	Pass	Pass	Pass
FindVirus	Pass	Fail	Pass
HTScan	Fail	Fail	Pass
Norton AV	Pass	Fail	Pass
ScanV80	Pass	Pass	Pass
Sweep	Pass	Pass	Pass
TBScan	Fail	Pass	Pass
VIRx	Pass	Fail	Fail
Virscan	Pass	Fail	Pass
Viscan	Pass	Pass	Pass
Vi-Spy	Pass	Pass	Pass
VPCScan	Pass	Fail	Fail

Table 3. Detection capabilities for encrypted viruses considered to be experimental or 'laboratory' examples.

While not presenting a current threat, manufacturers which employ reliable detection routines for these samples demonstrate their commitment to providing comprehensive protection.

The Whale virus in particular is of purely academic interest. Apart from one instance whereby the virus was seeded onto a machine, Whale has never been identified in the wild. Its chances of survival in the wild are slim - it usually causes infected machines to hang or reduces processing speeds so much as to arouse suspicion.

TOOLS & TECHNIQUES

Jim Bates

Scanner Tactics and the Reliable Identification of Virus Code

The increase in the number of viruses is likely to cause a shift in emphasis from specific to generic detection. However, there will *always* be a need for accurate identification of virus code. It is therefore time for an explanation of some of the problems and techniques involved in virus identification.

There are essentially two ways to identify computer viruses - by direct recognition of the code or informed observation of what functions the code actually (or potentially) performs. I shall refer to direct recognition as 'virus-specific' detection and informed observation as 'virus-generic' detection. This is a slight over-simplification since there is an overlap between the methods, but it will suffice for our purposes.

Let us first define what a computer *user* requires for protection against computer viruses:

> *Virus code should be instantly and unequivocally identified as such by a process that does not allow the virus code to execute. Thus any unknown software could be verified as 'clean' prior to being used.*

For the moment, I shall avoid the complications introduced by stealth viruses by assuming that the computing environment is clean and reliable. Starting with the ideal, the simplified instructions for such a detection method could be stated thus:

```
Find a file
CheckFile:
  Examine the contents of the file
  If it contains a virus go to Alarm
  Otherwise Find another file and go to CheckFile
Alarm:
  Tell the user I found a virus in the file
Stop
```

I call this process 'pre-emptive recognition' because one of its most important assets is that it will locate virus code before the code can be executed. Ignoring the fact that the simple listing above has no way of exiting when it runs out of files to check, the key statement is 'If it contains a virus ...'.

The assumption is that I know what I'm looking for, and this immediately excludes unknown viruses. If there was only one virus program which could possibly exist within a PC then the above instructions would be simple to satisfy - either the target file did contain the virus, or it didn't and the whole of the virus code could be compared against the contents of the file to determine this. Even if there were more than one virus, the principle might still apply as long as speed and memory were not vital considerations.

Unfortunately, once the number of viruses reaches double figures, both speed and memory become important limiting factors. The scanning program no longer has room or time to maintain and manipulate the whole code of each virus for comparison. So the limits are pushed back by selecting only a small portion of the code in each virus. As the number of viruses increases even more, the limits begin to loom large.

Checksumming Search Data

An alternative method of recognition involves taking a checksum of the relevant sequence and using that for comparison purposes. This method slightly reduces the storage requirements of the virus-specific information and it is capable of some expansion when dealing with simple encrypting viruses but as we shall see later its effectiveness is limited to detecting viruses which have at least a proportion of static code.

Immutable Mathematics

The time taken to search a file for any one of a number of code sequences (or checksums) must increase as the number of sequences increases. Similarly, the amount of space occupied by the sequences must also increase, whether they are maintained on disk or in memory. There are immutable mathematics behind these limits with which all virus-specific scanners must contend.

Search Instructions

I will now examine the actual scanning process.

First I must have a unique sequence of bytes which will only occur within the virus for which I am searching. This is impossible to guarantee unless we use all of the virus code. However, in practice a sequence of around 15 to 30 bytes will usually suffice without an unacceptable risk of false identification.

So, once we have our sequence, a simple search instruction might say:

```
Does the sequence - 1, 2, 3, 4, 5, 6, 7 - exist in
the file?
```

Selecting a section of virus code which will uniquely identify that virus might preclude such a simple sequence and instead require a broken sequence which would appear like this:

```
Does the sequence - 1, 2, 3, X, 5, X, 7 - exist in
the file?
```

Here, the letter X signifies any value. This makes the search process much more powerful because a unique sequence can now be selected even if it includes bytes which may change from infection to infection. The principle of specifying 'wildcard' values in this way can be extended to include a multiple wildcard value like this:

```
Does the sequence - 1, 2, 3, YX, 7 - exist in the
file?
```

where Y can specify a fixed number of wildcard bytes to include, or even a maximum number of wildcards. This is still a simple sequence matching technique but as the wildcard principle is expanded the search becomes a structure matching exercise.

The addition of wildcards increases the risk of false identification. Greater skill is required in selecting the recognition sequence. This becomes more difficult when dealing with self-encrypting viruses where each copy of the virus is encrypted with either a different method (like Whale and V2P6) or a different key (like Cascade and 4K). In these cases, the search for a recognisable sequence of instructions is usually limited to the code which doesn't change (the decryption stub).

With Cascade and similar viruses, only one decryption routine is used and this can be recognised easily. With Whale, the problem is recognising which one of thirty different decryption routines was selected at random by the virus for a particular infection. With V2P1, Flip, V2P6 and others, the decryption routine itself is modified in varying degrees by the viruses during each infection and in some cases further complicated by the random addition of non-essential instruction bytes.

Simple sequence recognition in these latter cases is impossible unless all combinations of instructions are considered. However, expanding the concept of sequence recognition into structure recognition adds a new dimension which makes identification of even V2P6 possible with a minimum amount of both sequence and structure information. This expansion is easier to understand if we write out our search instructions in plain English:

```
Find the true code entry point and note it
Search the succeeding Z bytes for a LOOP instruction
If the LOOP is found
  Note its target offset
  Search between code entry and LOOP for required Y
```

This fragment illustrates the principle behind structure matching. Noting that the target virus begins with a variable length decryption loop, our research will have told us the maximum length of the loop (the Z value in the instructions above) and also some of the code and data values which may be expected within the loop (the Y values). In the above case, Y may be anything from a simple sequence to a list of absolute and/or calculated values in byte or nibble form. The values being searched for will not necessarily be in a fixed sequence and may even be verified as part of an exclusive list.

It is precisely this requirement which limits the use of checksumming to maintain virus-specific search data.

The process of locating the true program entry point may be more or less difficult depending upon the complexity of the virus but will always succumb to detailed research.

Program Flow Analysis and Token Decryption

Some alternative methods of virus recognition include program flow analysis and attempting decryption based on analysis of the stub routine (although there are as yet no viruses which really need this level of inspection).

At one time, program flow analysis looked promising but the arrival of multiply infected files and viruses which use self modifying code has highlighted its limitations. Token decryption might also be a worthwhile avenue to explore but time will be needed to see whether an algorithm can be developed with a sufficiently universal application.

Categorisation

Viruses can be categorised in a way that helps the efficiency of the scanning process. For example, categorisation can ensure that time is not wasted searching COM files for EXE infectors and vice versa. Similarly, time is wasted searching the beginning of files for a virus which attaches its code to the end. In theory, optimising the scanning algorithm may produce a slight decrease in security. In practice, a useful gain in speed is attained with no loss of security.

Once a virus is disassembled its unmodified presence in a file can always be recognised by a capable scanner. The suggestion that some scanners are 'smart' or 'intelligent' is just marketing hype, the most important question that a user asks about a scanner is quite simple - *does it accurately identify known viruses?* Only if the answer to this question is *yes*, do subsequent concerns about speed and ease of use become of interest.

Packed Files

I use the word 'unmodified' above deliberately, since there remain two major problems in pre-emptive recognition. The first problem relates to packed files. These are individual program files packed with a utility which greatly reduces their size by a process of encryption and reduction. These should not be confused with the well known ZIP, ARC and other forms of file compression. A packed program appears just the same as an ordinary program and will automatically unpack in memory during execution. Any virus attached to the original program once packed in this way becomes much more difficult to detect.

There is no technical problem for a virus packed in this way since after the unpacking routine has executed, it passes processing control into the file exactly as if it had been loaded in an unpacked form. Thus if the file is infected, the virus code will execute as expected before passing control back to the host program.

The packed file problem is purely one of quantity. If there was only one type of packed file, with only one unpacking algorithm, it could easily be incorporated into a scanner such that the relevant files could be expanded in memory and then scanned in the usual way. However, there are in common use at least *eleven* different packers with at least *thirteen* different algorithms between them. More are appearing and incorporating all of them into a scanner imposes a heavy overhead in terms of execution time and memory requirements.

Fortunately, packed files are only a problem as individual sources for virus code since once it begins to infect on its own account, the virus becomes recognisable by one or more of the methods already mentioned. It is theoretically possible for a virus to recognise a packed file, unpack it, infect it and then repack it. However, this process would pose immensely complex problems for stealth viruses trying to conceal evidence of their existence in the changed file size and content. It should also be noted that a packed file which contains a virus internally (i.e. within the packing), is also likely to become infected externally as a normal program file.

Stealth Viruses

Stealth viruses cause a near insurmountable problem in pre-emptive recognition. Stealth viruses compromise the integrity of the operating system to such an extent that it can no longer be trusted to return accurate information. Thus at the lowest level of stealth, if DOS is asked for the length of a particular file and there is a stealth virus resident, the virus will hold the request while it checks to see if the target file is infected. If it is, the returned length will be modified to remove the infective length of the virus code. Higher levels of stealth become more sophisticated and are capable of 'hiding' the sections of a file which contain the virus code. **The simple answer is to reboot the machine from a known clean system disk and thereby restore its integrity**. A more complex alternative is to attempt to recognise the existence of stealth code in memory and either unhook it, or force a reboot to remove it.

Resident Scanning

The discussion so far has assumed that we're using the *point and shoot* type of scanner which requires the user to implement it at regular intervals.

A more recent development has been the introduction of resident scanning engines which will automatically scan program files as an integral part of the Load and Execute process. While the idea of a resident scanner is a good one in theory, the practicality introduces a host of new problems.

Firstly, these program are immediately heir to all the usual problems of residence - co-existence with existing facilities,

clashes over the use of system services, permanent use of valuable system resources, co-existence with network and multitasking handlers and so on. Also, the problems of stealth viruses are much greater with resident scanners since it is impracticable to insist on a clean reboot before every scan.

Once these difficulties are addressed, the principles are essentially similar. During the Load and Execute cycle, the requested program file is scanned in the usual way and the process is aborted if recognisable code is found. However, major problems can occur when the initial scan process does not detect a virus, particularly if the virus in question has stealth capabilities. Non-recognition may occur because the virus is in a packed file, is a new variant or is just not yet known to the scanner. **This is the most serious limitation of resident scanning: once a virus gets into the system, the monitoring and protection capabilities of the scanner may no longer be reliable but the user will not be aware that his defences have been breached.**

It is possible to make resident scanners aware to some degree of the current condition of the operating system services but implementing a full security check at every Load and Execute or disk access request would once again impose immensely heavy overheads on the system services. Even a simple check of the current values stored in the Interrupt Table would take time and still miss several viruses which hook their services using different methods.

Within these resident programs, there may often be additional code designed to protect the user in other ways from some of the more general effects of viruses. For example, attempts to reformat the hard disk may be aborted, as may attempts to write to the Master Boot Sector. There may be code to prevent the system from writing to program files, or a routine to strengthen the effectiveness of the write-protect attribute. Whatever is added at this level needs extremely careful consideration since it will effect all machine operations and may prevent legitimate functions from being performed. Some resident scanners insist on monitoring and scanning all disk read requests and this imposes a heavy overhead on disk access time without appreciably improving security since virus code is rarely invoked via a plain read function.

Postload Infection Detection

The discussion so far has concerned detecting virus code before it is executed. It is reasonable to implement some additional checks which assume that a virus may be resident and active in memory. Research has revealed a very efficient way of recognising virtually all known resident viruses during their operation. Normally, it is not considered wise to publish exact details of the more sophisticated detection methods as they can then become targets for the virus writers. However, in this case, targeting would be extremely difficult and it is therefore in the best interests of all users and vendors for such information to be revealed.

For want of a better description, we will term this detection method ‘looking up the skirts’ of a virus and it works like this:

Assuming that a monitoring program has accurately trapped any hooks that the virus may have installed into the system, virus system requests may be monitored as they pass into the interrupt system. [Note: it is not sufficient merely to trap requests via the Interrupt Table because some viruses such as INT13 access the operating system at a lower level. *Tech Ed.*] Each request for appropriate functions (determined by accurate research) is held while the calling code is examined for recognisable virus code. Because any interrupt request must give a return address (usually placed on the stack), this can be used by the monitoring software to locate and examine the requesting code.

Since the virus code is operative at that time it cannot be encrypted! Although there is one virus which encrypts and decrypts its code ‘on the fly’, even here the current code must be *functioning*. Hence it matters not what disguise the virus wears before it installs itself, the examining routine sees through it and can ‘watch’ the code actually at work. The key to this method is a secure method of ensuring that the monitoring software is *always* between the virus and the system. This is by no means as easy as it sounds but it can be done and the detection method does work with a surprisingly small overhead requirement.

Subversion

No discussion of scanning methods would be complete without a mention of the subversion techniques used by some viruses to get around the scanner’s security. Any program which attempts to collect and analyse the contents of files or disk sectors can become a target for subversion. Even those programs which are stored in ROM firmware need to communicate with the operating system and it is these linkage points between DOS and the programs which form the main target. I remain convinced that in the present basic design of the PC there can be no such thing as a 100 percent secure anti-virus system. *PCs are designed to run programs, viruses are programs and will therefore run on PCs.*

Conclusions

I have yet to meet a virus which has been able to avoid detection by the methods mentioned above.

As expected, the virus writers are running out of ideas. We are seeing fewer new techniques and fewer ‘clever’ tricks even though the numbers are increasing. There are only so many ways that virus code can achieve the spread of infection that is essential to its survival. As each loophole is identified and closed off, the security provided to users is increased. I remain optimistic that the level of protection available from the better virus detection packages will continue to be maintained well into the future.

DISCUSSION PAPER

Jonathan D. Lettvin
Lotus Development Corporation

A Call to Adopt the Lotus Virus Numbering Standard

Virus Scanning and Integrity Validation Checking

The majority of customer virus infections we encounter are the result of old or well known viruses. In general, a good scanning program, properly run, is sufficient to identify and locate these intruders.

However, the number of new viruses being produced will soon exceed the ability of professional virus investigators to extract, publish, and employ scan sequences. Integrity management is the future in virus response. Integrity Validation Checking (IVC) is our initial method for simple and fast integrity management. Our methods have detected instances of disk media failure, FAT damage, DOS failure, tampering, and viruses.

IVC (Integrity and Virus) Numbers

Lotus has developed a suite of programs for virus scanning and integrity breach detection for internal use. Also, our latest software products perform Integrity Validation Checking for *Lotus* product specific files. To support these programs and products we have developed a standard numbering strategy for viruses that we feel helps in their identification. These 'IVC numbers' never change.

When one of our programs detects a virus, it will display an IVC number. When we receive a report of an IVC number, we immediately retrieve up-to-date information about the virus from our virus database. In addition, *Lotus* Customer Support agents dealing with integrity problems have a simpler process to learn when *Lotus* products generate these IVC numbers.

Clients for the IVC Standard

One of the best sources *Lotus* has found for relatively standardized information on viruses is the *Virus Bulletin*. We add new IVC numbers and information from each issue of the *Virus Bulletin* to our database. To support our effort to standardize virus numbers, we are submitting the full IVC database and indexing strategy to the *Virus Bulletin* for integration into their database. We hope that other industry leaders in both hardware and software will support this effort by adopting this standard.

Lotus hopes that professional scanner companies will integrate these virus numbers into their products. Industry leaders adopting this standard one-to-one correspondence between the virus number and the actual virus itself will be offering professional virus crisis managers a valuable tool. Customers will receive more consistent quality and speedier service from their support personnel.

No Contest Against the Crackers

We do not intend to present a challenge to crack our methods. IVC is designed to be resistant to accidental damage, not proof against deliberate sabotage. Talented programmers have always been able to overcome any new measures taken to prevent sabotage. IVC is not intended as a barrier to any future virus although the effects of many new viruses will be detected. Our goal is to improve the reliability of diskettes delivered to *Lotus* customers. Virus detection is an ancillary activity provided as a courtesy to our customers with minimal performance cost.

Any program targeted as a host for a parasitic program can be subverted. We do not claim to prevent malicious or 'directed attacks.' Our experience with copy protection has taught us to respect the skills of dedicated independent programmers.

'S', 'I', 'V'

The IVC numbers in the first *Lotus* products were composite numbers. We now believe the use of a composite number was a mistake. The use of composite numbers resulted in no harm but it made support of our customers more difficult than anticipated. We have now decided that the IVC numbers refer to one of three types of integrity breaches by a separate indicator. IVC numbers from *Lotus* products will differ depending on whether that product was developed before or after the date we decided to simplify the numbers.

In the first few products, an IVC number between 1 and 999 referred to a System error. 'IVC0005: ' is translated as 'Access Denied'. An IVC number between 1000 and 1999 referred to an Integrity breach. 'IVC1002: ' is translated as 'File has Changed Size'. An IVC number above 2000 referred to a positive identification of a virus. 'IVC6007: ' is translated as 'The file has been Attacked by virus 7, the Cascade (01) Virus; and .COM files are at risk'. In this specific case, the addition of 6000 to the virus number indicated the .COM file portion of the message.

We will now be using the letters 'S' for System errors, 'I' for product Integrity problems, and 'V' for known or suspected Virus attack. 'IVC S0002:' will translate to system error 2, which on DOS and OS/2 means 'File Not Found'. 'IVC I0003:' will now translate to Integrity Error 'Initial instructions of the program have changed'. 'IVC V0028: ' will now translate to virus 28, which on DOS and OS/2 means 'Original Jerusalem virus'.

When we compare the indicator 'IVC V00028:' with the name 'Jerusalem', we find a clarity in the former and an ambiguity in the latter.

Why Not Virus Names?

What is wrong with virus names? We have been charmed by the creative naming of viruses. Often, the best name given a virus is a mnemonic for some major characteristic feature of the virus. It's tempting to think this works well when the number of viruses is small. In practice we find that a victim of an attack or an unseasoned investigator will prematurely associate a virus name with the attack. Sometimes, this premature association leads to confusion as the body of evidence accumulates toward a conflicting report.

"When we compare the indicator 'IVC V00028:' with the name 'Jerusalem', we find a clarity in the former and an ambiguity in the latter."

In addition, we have had some misgivings about the coordination of naming practices among the various authorities. We have seen the same name given to different viruses by different investigators. We have seen different names given to the same virus by different investigators. In each case, the published attributes associated with the virus were sufficiently in variance to raise our doubts that the same virus was being reported.

Also, for the virus author, the romance of having his/her virus 'named' is muted if a number is used instead. The political value of having investigators name the 'Jerusalem' virus as they did must have pleased the virus author, given what we know about that virus. The original of that virus is numbered 28 in our catalog.

What About Integrity Breaches?

Lotus uses IVC numbers for more than just viruses. We see viruses as part of a larger problem. Along with virus numbers we have numbers to represent specific types of integrity breaches. In practice, non-virus integrity breaches cause more problems for customers than viruses. The charter of IVC is to detect changes in *Lotus* product files after delivery to the customer. This allows us to give better service to our customers. The most stringent quality control cannot prevent the occasional marginal floppy disk media. Also, machine failure and mixed DOS version use can produce FAT damage and root directory damage. Customers will be alerted by integrated IVC code where it is possible to detect the problem.

How Are Numbers Chosen?

When *Lotus* first started numbering viruses we considered using the number to encode classification information about each virus. Classification sounded good. However, we found that the volume of new viruses coming in suggested a much easier approach. Just number viruses as we acquired methods to detect them. The sequence we have adopted is semi-chronological as higher numbers imply a later vintage.

Occasionally the *Virus Bulletin* will change a virus identification string or method. False positives are one of the reasons the Bulletin makes these changes. The first time we saw a change we had to make a decision whether or not to change the virus number. We find it most comfortable to retain the association of a number with a given virus even through such a change. Perhaps a revision number could be associated with the change although we have not implemented such revision numbers to date.

Sometimes customers will sufficiently analyze virus attacks on their own to acquire scan sequences for new viruses. When this happens, certain virus numbers can be employed in their local customized sequence database as temporary virus numbers while the sequences and viruses are submitted to the keepers of the virus numbers. We suggest that numbers from 99000 to 99999 are reserved for local use by such customers.

Additional Advantages

When I personally investigate a virus attack and see IVC generate a virus number, I feel a sense of reliable satisfaction that a cleanup can be done. If I am told by a customer that they have the 'Cascade' virus, I do not feel that same sense of reliability until I have investigated for myself or received from the customer one of the numbers 7, 8, 9, 10, 101, or 259. Only then am I prepared to proceed, after secondary confirmation, with assisting in a cleanup.

When in possession of a full complement of up-to-date viruses, an investigator can choose which virus to disassemble based on an in-progress attack without having to visit the site of the attack to get a live sample. This can save time in a possibly time-critical operation. Upon receiving a report from a client regarding a virus number for which the investigator does not yet have a full report, any source that has adopted the standard can give reliable information about the virus.

Possible Disadvantages

Publishers of virus information must agree to share the numbers without necessarily sharing the signatures. The only potential problem here occurs when the publishers have

different viruses which share a common section of code and then publish different signatures for different viruses while thinking they are the same virus.

One publisher (such as the *Virus Bulletin*), or a board of publishers must be the keepers of the numbers. This will end the era of independent and exclusive virus identification.

Conclusion

The practice of scanning for virus signatures will soon be obsolete. The time interval over which scanning will be useful as a method of virus detection will be determined by the ability of publications such as *Virus Bulletin* - whatever their media - to deliver accurate, useful, and easy-to-integrate information. Numbering viruses extends this opportunity a little longer as methods are developed for better integrity management and disaster recovery planning. We expect *Virus Bulletin* to maintain its leadership role in publishing signatures and promoting new safeguards well into the future.

While scanning is still viable as a method, we can use virus numbers to communicate confidently with our industry coordinators about our findings and continue to deliver and receive the very best in service that can be provided.

Jonathan D. Lettvin acts as Principal Investigator at Lotus Development Corporation and is the developer of the IVC system adopted by Lotus. Mr Lettvin also helped develop spreadsheets and utilities for Lotus and Access Technologies, biometric access systems for Ecco Industries Inc., interpreters for Catalytix Corporation, compilers for AT&T Bell Laboratories, and automatic authoring systems for MIT.

VB Classification

The *Lotus* IVC system is one of a number of classification schemes currently being evaluated for standardisation in *VB*. No adoption of any system is envisaged before next year as there are a number of issues which still need to be resolved.

Any classification scheme adopted by *VB* will certainly maintain names as a primary means to identification. As witnessed at countless conferences a numeric system is not helpful to general discussion because people remember names more readily than numbers. However, the inclusion of a standard number in addition to a name is under consideration.

Exact identification of virus specimens is a vexed issue; providing a 24-byte search pattern and an identifying number falls short of this requirement. At the inaugural meeting of *New Scotland Yard's Computer Virus Strategy Group* (see *VB*, April 1991, p. 2) two suggestions were proposed. The first

Virus Classification

Boot sector?	B
Floppy Boot?	f
Hard Disk Master Boot?	M
Hard Disk Partition Boot?	p
Infects COM files?	C
Infects COMMAND.COM?	c
Infects EXE files?	E
Companion virus?	Cm
Overwrites?	O
Appends?	A
Prepends?	P
Memory-resident?	R
Self-encrypting?	K
Redirects DOS calls?	Sd
Redirects BIOS calls?	Sb
Indiscriminate corruption?	X
Destructive trigger?	D
Network Aware?	N
Specific Targets?	T
Minimum Infective length?	????

Some examples:

Faust = CERAX1184 - COM/EXE/Res/Append/Indiscriminate

Beijing = BmR512 - Master Boot/Res

Spanish Telecom = CRAD3700 - COM/Res/Append

Nomenklatura = CcERAX1024 - COM(inc)/EXE/Res/Append/Indiscriminate

Aircop = BfR512 - Floppy boot/Res

Anthrax = BmCERA1000 - multipartite/COM/EXE/Res/Append

Figure 1. Proposed virus classification scheme from *Bates Associates*. The researcher logs each entry sequentually from the top of the list downwards ignoring irrelevant entries. This method is inexact when compared to checksumming but, unlike the latter method, can contend with self-modifying encrypted viruses. This scheme also facilitates the comparison of virus samples - matching listings may indicate identical specimens.

was to provide a checksum value using an agreed algorithm for each virus specimen. The second method proposed was a classification scheme whereby the properties of the virus are described in a defined sequence (see *Figure 1*.)

Both methods have their drawbacks. A checksum identifier, while providing an exact identification for the overwhelming majority of virus samples, will only identify non-encrypting, non-mutating virus code. Obviously a checksum can be taken of the decryption stub in the case of viruses such as Cascade but the advent of self-modifying encryption (V2P1 through V2P7) and the recent widespread availability of the so called Mutation Engine has rendered this approach obsolescent. You cannot checksum a program which exists in some 4.5 billion guises! Agreeing on the choice of the checksum algorithm used and coordinating its distribution to all relevant parties presents an organisational challenge to even the most diplomatic and efficient mediator.

The second method devised by Jim Bates (*Figure 1*.), while being somewhat easier to administer, is obviously less precise. Whereas checksumming (where it is possible) will reveal a single bit variation between samples, the most that this system can do is provide approximations or indicators. Similarities between samples are quickly revealed, but exact identification can only be confirmed by traditional byte-by-byte comparison between samples.

Why then do companies such as *Lotus* and *IBM* (to mention just two of the many large corporates addressing this issue) concern themselves with virus naming conventions and identification? Principally, recovery is simplified if the exact nature and behaviour of the offending virus is known. In the case of a serious contamination spread across numerous machines and media, software disinfection routines (where possible) can be developed, tested and distributed quickly.

Equally important is the need to notify customers who have fallen victim to a PC virus infection of the exact side-effects, trigger conditions and other diagnostic information about the offending virus to assist disinfection. Management do not need to know the technical intricacies of the sample involved but they do want to know if it triggers at midnight on a 'mission-critical' machine or set of machines. Furthermore they want accurate information forwarded to the technicians as quickly as possible.

Police and law enforcement agencies obviously require exact identification of each and every virus specimen which is reported to them. Forensics demand full and annotated disassemblies. Without this information no investigation can commence or prosecution be secured.

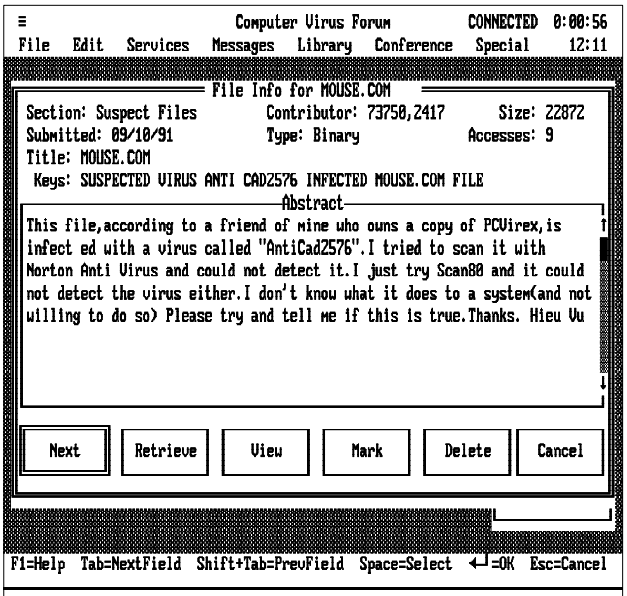
Many large corporates are seeking standardisation of names and terms among anti-virus software vendors. The multiplicity of naming conventions now in existence is causing enormous confusion among computer users. The question is whether the anti-virus community can unite on this issue.

BBS FORUMS

The Need for Vigilance

The need for BBS SySops to remain vigilant was re-emphasised recently when live virus code was discovered on the *Computer Virus Forum* administered by *McAfee Associates* on *Compuserve*. There are six library areas in this conference whereby various files are made available for download. The file in question, an infected version of MOUSE.COM, was found in the library called 'Suspect Files'. As is shown on the screen dump below, the contributor of this file believes it to be infected with Anticad 2576 (it is in fact a 3088 byte variant of the Taiwanese Plastique family). The file was uploaded on 10th September by user 'Hieu Vu' (73758,2417).

Scotland Yard's *Computer Crimes Unit*, which had received an independent complaint from a British subscriber to *Compuserve*, contacted the company's Bristol offices. This resulted in the 'Suspect Files' library area being immediately closed down by the *Compuserve* administrator responsible for this forum. *McAfee Associates* state that the 'Suspect Files' area is a 'one-way' channel for uploads only and that this particular file had become unlocked by accident. Nine copies of MOUSE.COM were downloaded. Companies which harvest malicious code via BBS forums must ensure that suspect files are **never** made available for download. In this instance there was no access control to this infected file whatsoever.



Breach of security. Infected file MOUSE.COM freely available for download on *McAfee Associates' Computer Virus Forum*. Nine accesses are indicated. A cautionary reminder to SysOps to remain vigilant.

VIRUS ANALYSIS 1

James Beckett

The Liberty Virus

A sample of this virus was received by *VB* in early June 1991 from a bank in Hong Kong which had suffered widespread infection and subsequent reinfection on a local area network. In this instance the source of the virus was traced to a visiting software salesman - a widely acknowledged source of virus infiltration.

In his accompanying letter, the systems manager at the bank referred to the offending virus as 'Mystic' after a pattern to be found near the start of the file. However, the sample was detected by the pattern for the Liberty virus published in the January 1991 issue of *VB*. Liberty had hitherto not been analysed, and Mystic was assumed to be a 'hacked' variant. After analysis and comparison, however, it is apparent that they are one and the same. (*VB* will continue to refer to this virus primarily as Liberty, with the addition of the alias Mystic. Other researchers have also called it 'Magic'.)

Infective Path

The infection (and trigger) methods of Liberty are convoluted and unusual. In the January issue it was reported as being parasitic only, infecting both COM and EXE files. Boot sector infection must now be added to the list as the virus switches its attention to diskette boot sectors on discovering that the targeted disk is full.

The virus side-effects are initiated in such a way as to ensure widespread dissemination before activation and a maximised potential for data loss.

On program files Liberty/Mystic has an infective length of 2857 bytes. *VB* published an initial report that 'a 2867 byte mutation is known' but this is due to the fact that targeted files are padded by up to 16 bytes before the virus code is appended.

Patterns and Symptoms

The method for appending the virus code to the target file, including the modification of the original program to jump to the virus code first, is well known. Simpler viruses (such as Tiny, *VB*, Sept 90, p.17) simply overwrite the start of the program, destroying it (and also limiting the virus' own spread). Liberty is unusual in that it overwrites about a hundred bytes at the start of COM files with initial code and a message, saving the original data in encrypted form in the main virus to restore later.

This message provides the name of the sample, as well as its presumed author - SAMUEL.

```
- M Y S T I C - COPYRIGHT (C) 1990-2000 by
SSASMSUSESL
```

The word 'Liberty' is used as the virus' own self-recognition signature to prevent multiple reinfection of a COM file. In EXE files it uses a different signature - an out-of-range value FFFF in the program size field in the EXE header. This can cause some versions of DOS to refuse to run the program resulting in error messages such as 'Insufficient memory'.

Installation and Propagation

When run from a parasitic program, Liberty/Mystic relocates itself to high memory and traps the DOS Load and Execute function (INT 21H fn 4BH), infecting all COM and EXE files run. It will not infect a COM file smaller than 1280 bytes or larger than 61 Kilobytes, or an EXE file larger than 704 Kilobytes. Residence in memory is flagged by a value in the Interrupt Table to avoid multiple instances of the virus becoming active.

This appeared for some time to be the only route of infection. After my analysis, confirmation was received from other sources that diskette boot sector infections had been discovered, but under conditions unknown to my sources.

Track 40

These conditions are in fact already known. When it fails to infect a file due to the disk being full, Liberty tries to infect the boot sector of the next diskette accessed, trapping INT 13H for one invocation. For its boot-infection, the virus formats track 40 to an unusual format in order to store its main code section. The original boot sector is stored in track 40 with its 3rd byte changed to an FFH, and the virus boot code commences at logical sector 0. As usual, the original boot sector is executed after the virus has installed itself.

Track 40 is a common hiding place for virus code on 360 Kbyte diskettes. However, as has been noted before in *VB*, this can cause corruption on 80-track 720 kbyte and 1.44 Mbyte diskettes. As Liberty only writes this track when it finds that a disk is full, the likelihood of corruption is high.

The track is formatted with nine 512 byte sectors numbered 31H, 32H, ... 38H, 3FH, the virus being stored in the first eight and the boot sector in the last. When reading tracks, DOS only looks for sector numbers in the range 1 to 9, and disk editors often impose similar restrictions, making recovery difficult without special software.

On booting from an infected diskette, the virus traps INT 13H immediately and indefinitely, infecting any locatable disk(s). It also sets up the INT 21H intercept to continue infecting files.

A count is kept of the number of times an infected diskette is booted from, and when this reaches 10, the virus copies back into place the original boot-sector (with its 1-byte change), effectively disinfecting the diskette of this part of the virus. Such a diskette will no longer install the virus if it is booted.

Side-effects

For each boot from an infected diskette before this copying-back process, the overtly visible side-effects of the virus are initiated: For about 2 seconds, at ever-decreasing intervals starting at 5 minutes (this reducing by some 20 seconds each time), several resources are intercepted: all writes to the display using INT 10H functions 09 and 0A, reads/writes to COM ports (INT 14H) and writes to the printer (INT 17H), are replaced with the letters of the text 'MAGIC!!' in sequence. The effects differ depending on exactly how a program (or DOS) chooses to write characters: a third INT 10H function (0E, 'write characters in teletype mode') is not trapped, and is used by some software.

After a total of about 45 minutes, the virus flashes a bar containing 'MAGIC!!' at the top of the display for about a minute (this being the delay time at this point) and then attempts to start up ROM BASIC. On all test machines, this resulted in the computer crashing.

Destructive Effects

If the virus is transiently trapping INT 13H after installation from a program rather than through the bootstrap virus' continuous trap, and if it finds that the diskette boot sector it tries to infect is one which it has previously copied back in its 'disinfection' routine, track 0 is reformatted similarly to track 40 and the track 0 data replaced. This makes the disk unreadable by DOS, and even by some popular disk utilities including Norton. The data, however, is still intact and potentially recoverable.

Style and Programming

The programming is a curious mixture of precision and extreme carelessness, suggesting that the virus (or its derivative code) may have been written by an experienced programmer, then modified by someone less competent. The final overwriting of sector 0 on diskettes is probably an unintentional side effect. Destroying disk data can be achieved in a much less tortuous way - it can be done in only four assembly instructions. However, this side-effect renders the diskette unreadable by DOS. Incidentally, many DOS FORMAT programs will refuse even to reformat the modified disk!

Care is taken to compensate for any modifications which the virus makes, for example to the EXE file header. When flashing the message to screen, the original screen data is fastidiously replaced. Returns from functions are invariably checked for error codes.

Conversely there are several examples of poor programming, including superfluous calling of routines, and failure to save old values of interrupt vectors properly... even restoring one such unsaved value to the wrong place.

Parts of the code are liberally sprinkled with NOP instructions, suggesting modifications to object code, for example using DEBUG, rather than re-assembly of a source code listing. Most of the data strings used in the virus are trivially encrypted to evade casual scrutiny. The code isn't particularly engineered to make disassembly difficult. Files on any medium may be attacked parasitically, but only diskettes have the boot sector modified.

Detection

The following pattern will detect both file and boot sector infections caused by the Liberty virus and should be used in preference to the previous pattern published in *VB*.

```
Liberty   B931 2833 D2CD 130D BB5C 0653 CB2E
          803E BC06 0A74 4633 C08E
```

Disinfection and Recovery

As this virus is multi-partite, disinfection may be required for both executables and diskettes.

Always start clean-up procedures after booting from a known clean write-protected system diskette. Although this virus doesn't take steps to hide itself in memory, if active it will happily infect any program you run while you're trying to disinfect your system.

COM and EXE files should be restored, as always, from copies of the original master software. Diskette boot sector infections in general are less tractable, requiring use of a disk editor - in this case a good deal more skill is required as Liberty protects itself rather unusually on diskette. Because the copied boot sector is modified, in most cases it cannot simply be written back to its rightful position as the 1-byte change induced by the virus causes the machine to halt on reboot. Some cautious editing is required as the virus is designed to trigger if this is not done correctly. However, this is rather academic as a disk editor is unlikely to be able to read the oddly-formatted track used to save the sector anyway! In this case the simple answer is to copy all the files from the disk (replacing any infected executables on the way) and reformatting it.

The real problems come when the virus finally triggers and reformats the whole of track zero on the diskette. Your files are still on the disk - even the boot sector, FAT and root directory are still there, but you can't use them, DOS can't access them, and your favourite disk editor refuses to believe that they're there! You can use DEBUG at a pinch, but unless you're an adept assembly-language programmer you really are better off looking for some professional help.

VIRUS ANALYSIS 2

A. Thomas Busey
Microcom, Inc.

Rage - Monitor Subversion

Rage is yet another virus from a group self-styled as the RABID International Development Corporation which is also responsible for the Violator family of viruses. Rage is not new or innovative, but it contains several features worth examining. Rage appears to be an experimental virus implementing several techniques which may be used in future RABID viruses. Rage is a non-resident COM file infector. The most significant feature of the virus is its attempt to evade detection by the memory-resident component of *Virex-PC* - a feature which will be examined in greater detail later in the article. [*Adversarial* is a generic term to describe viruses which attempt to subvert software or hardware defences. Ed.]

Encryption

This is the first RABID virus which employs encryption. The encryption is not very effective because the decryption routine is neither variable nor hidden. However, it could cause problems in the future. With the appearance of the Dark Avenger's Mutation Engine (source code which makes it easy to add variable encryption mechanisms to any virus), it is probable that many virus writers will be able to render their decryption routines undetectable by normal hex string scanning.

Non-Destructive Trigger

Rage has two trigger routines. The first trigger is based on the system time. The virus uses an INT 21H function 2CH call to get the system time. The virus then ANDs the DL register with 0FH. The DL register stores the current time's hundredths-of-a-second field. If the hundredths-of-a-second field is a multiple of 16 (i.e. 00H, 10H, 20H etc.) the following message is displayed:

```
Pray for death - RABID '91
```

and the virus continues processing normally.

Destructive Trigger

The second routine is based on the system date. If the date is the 13th of any month the virus activates its destructive routine. This routine begins by converting the display to 40 x 25 column mode and displays the following message:

```
Rage - RABID Int'nl Development Corp.
By Data Disruptor - Thanks to Zodiac
```

The program then issues INT 26H calls which perform absolute disk writes, to overwrite sectors 0 through 255 of hard disks drives C: to Z:.

Virex-PC

Rage is interesting in that it specifically recognizes the presence of *Virex-PC* in memory. It does this with an interesting method never seen before. Apparently, the author of this virus has spent considerable time disassembling *Virex-PC*. VIREXPC.COM, the memory-resident portion of *Virex-PC* (versions 1.2 and earlier), uses an 'are you there' call' to see whether it is already resident. The author of Rage found this call and uses it in the virus. Fortunately, the use of this information is not harmful to *Virex-PC* users. If Rage detects that *Virex-PC* is in memory it simply returns control to the host program. The logic behind this is elusive. The author may believe that this will help the virus evade detection. As a result, if *Virex-PC* is active when a Rage-infected file is run, Rage will remain dormant.

How Rage Spreads

When an infected program is run, Rage will infect one COM file in the current directory. If there are no uninfected COM files which meet its specifications in the current directory, it will infect one file in the root directory. Rage will infect any COM file that is at least 2000 bytes but not more than 63,425 bytes in length. The infected file will increase in size by 575 bytes. Rage does not alter the file time and date stamp when it infects a file. All infected files contain the following unencrypted text:

```
Patricia Boon
```

To detect a Rage infection the following hexadecimal search string can be used:

```
0181 C203 018B F28B EA83 C541 9055 EB0D
```

This hexadecimal string represents the code just before the 'Patricia Boon' text. If the decryption routine is preferred for searching, the following string be used:

```
8A24 518A C8D2 C459 8824 FEC0 46E2 F1
```

Editor's Comment

The Rage virus is yet another example of attempted subversion of an anti-virus software product. In this instance, the virus simply seeks to evade detection by issuing an 'are you there' call to VIREXPC.COM. Obviously *far* more hostile effects could be encoded. Software that is in widespread use is liable to be targeted in this way. As a result secure distribution channels and self-checking code is becoming prevalent among manufacturers. The danger presented by adversarial or targeting viruses is a major concern for software developers seeking to protect their products and customers.

VIRUS ANALYSIS 3

1575 - Attack of the Mutant Green Caterpillars

Following the small number of common viruses which contribute to the greater part of reported 'real world' infections comes a long trail of lesser protagonists in the virus battle. Though hugely overshadowed by such names as New Zealand 2, Cascade and Jerusalem, reports of all manner of viruses are increasing and specimens which numerically represent only a few percent of total reported infections are still taking hold and causing problems.

One contender that is moving up the ranks of those jostling for a place behind the 'leaders' and currently being reported out in the field is the Caterpillar virus (aka 1575) This is now overhauling the likes of Italian and Plastique, and on a par with Joshi, Dark Avenger and Nomenklatura.

A COMMAND.COM Infector

Some viruses specifically avoid infecting COMMAND.COM despite the fact that this file is one of the first to be executed, which logically should increase the potential for any virus to propagate.

It has been suggested that the virus writers' general aversion to the COMMAND.COM file is because its infection is altogether too obvious to the user resulting in the early detection of the virus. However, the widespread Cascade virus infects COMMAND.COM with no apparent risk of such premature discovery, so the perceived risk of infecting this file may be over-estimated by the virus writers.

The Caterpillar virus specifically looks for C:\COMMAND.COM on every invocation (the string is encrypted within the code) and actually targets this file for infection - this may explain why the virus has become relatively widespread in a short space of time.

Operation

Caterpillar infects both COM and EXE files. It goes memory-resident by manipulating DOS arena pointers (though it doesn't lower the BIOS Available Memory information) and traps the DOS FCB find-file functions - thus infecting files when the DIR command is used to examine disk contents. This additional disk activity ought to alert users, at least when using diskettes, but makes this virus highly infective if it is not noticed.

Only files in the inspected directory are infected, and a bug in the interrupt routine causes the virus to miss all files that have the full eight characters in their base name. [*On the machine used to compose this article such files comprise about 8 percent of the number of executables on the hard disk*. Ed.] The DOS critical error handler is trapped, so users will not see failed attempts to write to a protected diskette. Infected EXE files have a tendency not to run correctly due to corruption by the virus.

Trigger

The trigger routine of this virus occurs when a file which has been infected for over two months is run, COMMAND.COM is already infected (or doesn't exist in the root directory), and there is another copy of the virus already resident in memory.

In the idiosyncratic manner which we have come to expect from virus authors, the user is subsequently greeted with an animated green caterpillar crawling down the screen, munching through characters and turning the text yellow in its wake.

If uninterrupted, this display lasts some three minutes. Any input which causes the screen to scroll up results in the caterpillar jumping back and continuing from its new position. It cannot be scrolled off the top of the screen and only stops when it finishes its journey.

Technically, the programming is a curious melting pot: neat, legible code interspersed with long-winded meandering, trivial errors and some arcane structuring (or lack thereof). Instead of a simple JMP linking the virus code to the start of a COM program, 12 bytes of instructions are used to set up the code segment for the virus.

"The user is greeted with an animated green caterpillar crawling across the screen, munching through characters and turning the text yellow in its wake."

The next piece of code manipulates the program stack in highly dubious ways and jumps all over the shop but eventually achieves absolutely nothing. Its only function seems to be to foil an automatic debugger - this particular section cannot easily be traced as it uses a part of the Interrupt Table as its stack. Another possibility is that it is related to a launch or dropper program which is not carried with the virus.

Even before becoming resident in memory, the virus checks for the existence of an uninfected C:\COMMAND.COM and infects it. A data string of 'C:\COMMAND.COM' might

arouse suspicion so this string is 'encrypted' (to use the term very loosely) by the addition of 32 (decimal) to each character. Infected files are marked by having the last two bytes as hexadecimal 0C, 0A.

If the virus finds that another copy of itself is already installed in high memory (by searching for the 0CH and 0AH bytes file signature), it checks its own date of infection to ascertain whether it is more than two months old. If it is, the BIOS timer tick (interrupt 1CH) is trapped and the already-installed copy services these interrupts to produce the animated caterpillar. Note that if C:\COMMAND.COM has just been infected the caterpillar display will not initiate.

On going resident, interrupt 21H (the DOS function executor) is trapped to intercept functions 1AH (set DTA address), 11H (FCB find-first) and 12H (FCB find-next). From that point on, all results from a file find are examined after letting DOS go ahead with the operation. Returned filenames are parsed and examined for .COM and .EXE extensions, except, as noted above, this parse routine fails for 8-character base names. The surrounding code confirms that this is not a deliberate 'sparse infection' ploy. The first suitable executable is opened in read/write mode and updated with the virus code, with the current month being planted in the new virus.

Naturally, the virus prefers to let DOS initially process the find functions. As the virus has already trapped these itself, it needs to bypass its own interception to let DOS do the job; the usual way is by a FAR CALL to the old address, but this author chose the tack of providing a 'pass-thru' code. Interrupt 21H is still used within the handler - a potential recursive death - but the only two calls it needs to make don't use the processor's AL register and it places a hex 57 in AL to avoid tying itself in knots. Its own intercept routine catches this signature and passes the request to DOS.

Detection and Removal

The Caterpillar virus has an infective length in COM and EXE files of between 1575 and 1591 bytes, and may be detected by the following hexadecimal pattern:

```
Caterpillar 0E1F A12B 018E D087 ECBE 3C01 BF00
            00B9 1000 FCF2 A4E9 DEFE
```

Caterpillar is not encrypted and makes no attempt to hide itself in memory or on disk, not even hiding the increase in file size.

Infected program files should be deleted and replaced with clean write-protected copies of the master software.

[*The next issue of VB will contain an analysis of the recently discovered and reportedly fast-spreading DIR II virus from Bulgaria which incorporates innovative stealth features.*]

PRODUCT EVALUATION

Dr. Keith Jackson

PC-EYE

PC-EYE is an anti-virus software package that provides facilities to verify checksums calculated for a defined set of files; scan files for virus 'signatures'; monitor program execution; and detect/remove viruses present in memory. These facilities can either be individually invoked, or they can be selected from a menu driven program. Many other small utilities are also used but space constraints prevent their description.

Installation

Some of my reviews have started with a favourable initial impression of a product which gradually diminishes as the product reveals its actual capabilities. This review travels in the opposite direction.

PC-EYE comes with an INSTALL utility so my immediate effort was targeted towards using this program. Immediately after the installation program was executed the following error message appeared:

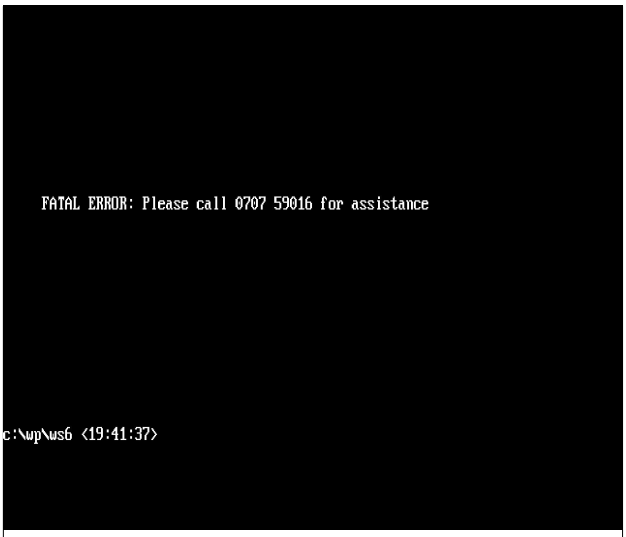
```
FATAL ERROR: Please call 0707 59016 for assistance
```

was prominently displayed. I tried out INSTALL on two other computers and the same error message appeared. Not an auspicious start.

On telephoning for assistance I found that the error message was readily explained. The first question isolated it: 'Do you have the DOS file COMMAND.COM in the root directory of the hard disk'. My answer was no for all of the three computers tried; I hate files cluttering up my root directory and COMMAND.COM can be placed elsewhere so long as DOS is told where to look for this file during the boot process (I won't go into the details of how to use COMSPEC to inform DOS of this).

When I enquired why *PC-EYE* did not just look at where the hard disk boot process had located COMMAND.COM, it was pointed out that this does not work if the computer is booted from a clean floppy disk; a much used and very useful anti-virus tactic. I enquired why all this was not explained in the manual; summarised in one word the answer was 'simplicity'.

Given the complexities that can be involved in this matter, I agree that asking the user to telephone for assistance is simpler than trying to explain what is quite a complex situation. However, a hint in the manual would help matters.



Expect this message if COMMAND.COM is absent from the root directory when installing PC-EYE. The manufacturer will talk you through the problem but what happens if no-one is in the office over a weekend?

The developers of *PC-EYE* claim I'm the first person to have reported this error, which is either an intriguing comment on how most PCs are setup in a standard manner, or a comment on my usage of MS-DOS. Probably the latter.

Documentation

A lack of explanation of the possible error messages detracts from the *PC-EYE* documentation, which is provided as a 69 page manual in an A5 ring binder. This manual is very clear, does not get bogged down in too much detail, and should be understandable by the even the most technically naive user. I'm becoming rather coy about mentioning this point but the manual does not have an index - it should.

Scanning

I tested the scanning portion of *PC-EYE* for both speed and accuracy of detection. A complete scan of the hard disk took 2 minutes 27 seconds, which reduced to 59 seconds for a fast scan (which only scans the first and last 4 kilobytes in each file). For comparison purposes, *Sweep* v2.29 (from *Sophos*) scanned the same hard disk in 5 minutes 51 seconds reducing to 1 minute 14 seconds when a quick scan was requested, and v76 of *McAfee's SCAN* took 9 minutes 1 second for a complete scan, and 3 minutes 19 seconds for a fast scan.

These measured figures place *PC-EYE* in roughly the same position as that shown for the product in the *VB* scanner review (see *VB*, September 91, page 16) i.e. very quick indeed for a fast scan, and still one of the fastest when performing a complete scan. I could not commence a scan from within the menu utility, as any such request locked up the computer. I shall return to this point.

The impressive scanning speed did not seem to impact upon the accuracy with which viruses are detected. From the 183 virus samples listed in Technical Details, *PC-EYE* failed to detect a virus in only four test files (Terror, Turbo 488, and both samples of Voronezh), an accuracy of 97.8%, not dissimilar to the 95.7% quoted in the *VB* scanner review. Within this result, the 1260, Casper, Flip and Virus-101 viruses were only described as having a bad date/time stamp, a message that could be misconstrued by a naive user. In some cases (Dark Avenger, Datacrime II, Hymn, Jerusalem, Keypress, Lovechild, Murphy, Sunday, Typo, Yankee and Vienna), *PC-EYE* reported more than one virus infection within what was a singly infected test file. This presumably occurs when more than one signature is used to test for a given virus.

Fingerprinting

Verifying the fingerprints of all executable files on my hard disk (holding 33.8 megabytes of files) took 2 minutes 24 seconds. It is noticeable that this is very close indeed to the time taken to scan the same disk for viruses, suggesting that the time to carry out either check is taken up more by file manipulation than by actual testing.

Fingerprints can be calculated by using either a simple checksum algorithm, or an encryption algorithm. The user can choose which is most appropriate for his particular environment. Given that I have expounded at length in *VB* about algorithms used for fingerprint calculation, I was pleased to see such a choice being offered. However, the manual gives no indication of the relative merits of the two methods of calculation, nor details the algorithm used.

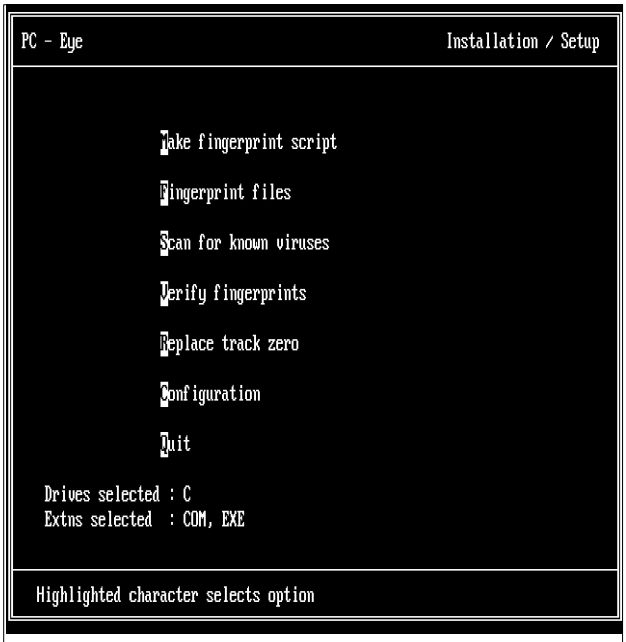
Curiously the fingerprint verification time is the same for both cases (I would have expected CRC calculations to be *much* quicker). I assume that the developers of *PC-EYE* will help in this selection process and guide the user through the various trade-offs which must be made.

I did find a few minor problems with *PC-EYE* fingerprinting:

- When the encryption algorithm was used in verifying fingerprints, the following error message appeared when all files had been verified: 'run-time error R6001 - null pointer assignment'. This looks suspiciously like a Microsoft C error report. I have no idea why this appears.
- If the fingerprint creation process is terminated early (by pressing the Escape key), when verification is subsequently attempted the computer locks up.
- During fingerprint verification the serial number displayed on the screen is 123456. However the serial number written on the *PC-EYE* master disk is 700035. Curious.

Clearing Memory

PC-EYE provides a test utility which ensures that the interrupt vectors required by the operating system have not been altered, tests for a virus already resident in memory, and ensures that memory is cleared. This utility executes very quickly, and provides a valuable test function as it tests ordinary RAM, extended RAM and expanded RAM. This claearance routine provides an extremely useful first line of defence against current stealth viruses.



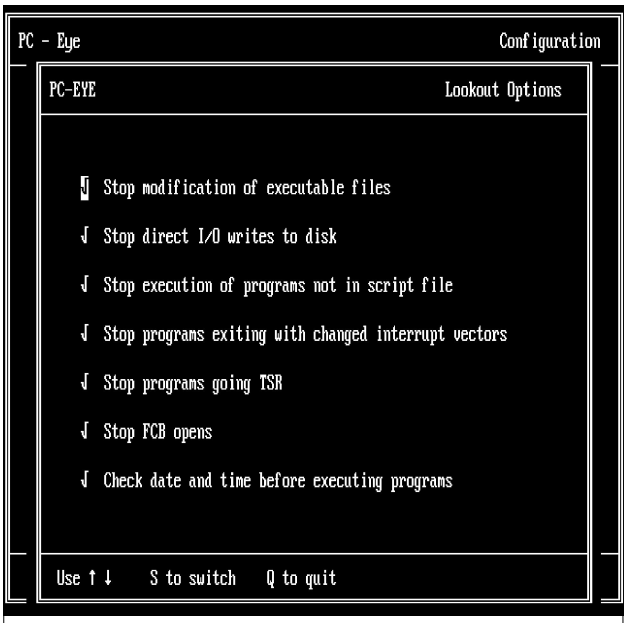
A clear and straightforward options menu offering both checksumming of executables and virus scanning. Here, only COM and EXE extensions are selected for checking although full extension checking can be configured.

Monitoring Program Execution

A utility called *LOOKOUT* is provided to monitor program execution, and disallow what it thinks are illegal actions.

The user can tailor the operation of *LOOKOUT* in many ways. This proved necessary, as predictably *LOOKOUT* was the component of *PC-EYE* that gave me the most problems:

➤ If the option preventing execution of a file unless it is in the *PC-EYE* script file was not disabled, then an error message stating ‘ATTEMPTED ILLEGAL FUNCTION CALL’ was always produced, almost as though *PC-EYE* could not locate its script file. I don’t understand why this occurred, and it may well be my fault.



LOOKOUT options menu. This monitor attempts to trap ‘illegal’ actions such as direct disk I/O writes and infected programs going memory-resident. The problem with all such monitors is defining ‘illegal’

➤ If you type ‘LOOKOUT’ many times mindlessly, then it installs itself repeatedly as a memory-resident program. This is wrong; it should detect that it is already installed, and refuse to install again.

➤ When *LOOKOUT* detects something it thinks should be prevented, it displays an explanatory screen and asks the systems administrator (not the user) to press any key to perform a hard boot. On my test computer, this usually (but not always) resulted in the machine hanging.

➤ Nothing in the documentation explains how much memory *LOOKOUT* requires when resident (probably 19 kilobytes).

Conclusions

In fairness, I should mention that this is not the first time that I have come across *PC-EYE*, I wrote about it approximately a year ago (not for *VB*). So I did start this review with some knowledge of the product.

Although the performance of the *PC-EYE* scanning program was very good (one of the best around), this was marred by the fact that this part of *PC-EYE* could not be executed from the menu interface. As scanning performed flawlessly when executed as a standalone utility, this sounds like a problem caused by lack of memory. However my computer has 541 kilobytes of RAM available before the *PC-EYE* menu interface is executed, more than many other computers.

Even though I have reservations about my own use of a memory-resident monitor such as *LOOKOUT*, I have little doubt about its benefit when used in a controlled environment (such as that offered by the majority of PCs in existence). So long as some time is taken choosing the correct things for *LOOKOUT* to monitor, then it is potentially capable of acting as a very useful filter of virus activity.

To sum up, I like the simplicity of *PC-EYE*. Particularly the simple way that the menu interface offers single sentence options among half a dozen choices, any one of which is selected with the appropriate highlighted letter. As an example of this simplicity, no matter how you view the COMMAND.COM installation problem described earlier in this review, it must be admitted that the solution is probably the simplest one. Rather than run the risk of complicated instructions on how to find COMMAND.COM and copy it to the root, just get users to phone up and talk it through.

However, such tactics do have a downside. I found the error late one Saturday evening (what better time to test anti-virus software!), and had to wait until Monday morning to get past the inevitable answering machine. I can visualise a very distinctive message on this machine:

> *'I'm afraid that there's nobody here, please call back during office hours or leave a message. Oh by the way, if you've found a fatal error while installing PC-EYE: try copying COMMAND.COM to the root directory'.*

This is guaranteed to cause confusion - maybe talking users through the problem is best after all!

LOOKOUT! There's something dastardly in memory! Notice the instruction to summon the system administrator thus relieving users from complex diagnostic and disinfection tasks.

Technical Details

Product: *PC-EYE*

Developer & UK Vendor: *P.C. Enhancements Ltd*., The Acorn Suite, Greenleaf House, Darkes Lane, Potters Bar, Herts. EN6 1AE, UK, Tel: +44 (707) 59016, Fax: +44 (707) 55523.

Availability: IBM PC/XT/AT, PS/2 or compatible, PC-DOS/ MS-DOS v3.00 or higher. 512K of RAM is required, and a hard disk drive.

Version evaluated: 2.1g. This version did not seem to exhibit any of the malfunction mentioned by Mark Hamilton in last month's *VB*.

Serial number: 700035

Price: £157.45 including monthly updates and VAT.

Hardware used: A Toshiba 3100SX laptop with a 40 Mbyte hard disk, 5 Mbytes of RAM, a 16 MHz 80386 processor, and a single 3.5 inch floppy disk drive.

Viruses test-suite: This is a test-suite of 113 unique viruses (according to the virus naming convention employed by *VB*), spread across 182 individual virus samples. It comprises two boot sector viruses (Brain and Italian), and 111 parasitic viruses. There is more than one example of many of the viruses, ranging up to 12 different variants in the case of the Tiny virus.

The actual viruses used for testing are listed below. Where more than one variant of a virus is available, the number of examples of each virus is shown in brackets. For a complete explanation of each virus, and the nomenclature used, please refer to the list of PC viruses published regularly in *VB*

1049, 1260, 1600, 2144 (2), 405, 417, 492, 4K (2), 5120, 516, 600, 696, 707, 800, 8 TUNES, 905, 948, AIDS, AIDS II, Alabama, Ambulance, Amoeba (2), Amstrad (2), Anthrax (2), Anti- Pascal (5), Armagedon, Attention, Bebe, Blood, Burger (3), Cascade (2), Casper, Dark Avenger, Datacrime, Datacrime II (2), December 24th, Destructor, Diamond (2), Dir, Diskjeb, Dot Killer, Durban, Eddie 2, Fellowship, Fish 6 (2), Flash, Flip (2), Fu Manchu (2), Hymn (2), Icelandic (3), Internal, Itavir, Jerusalem (2), Jocker, Jo-Jo, July 13th, Kamikaze, Kemerovo, Kennedy, Keypress (2), Lehigh, Liberty (2), LoveChild, Lozinsky, MIX1 (2), MLTI, Monxla, Murphy (2), Nina, Number of the Beast (5), Oropax, Parity, Perfume, Piter, Polish 217, Pretoria, Prudents, Rat, Shake, Slow, Subliminal, Sunday (2), Suomi, Suriv 1.01, Suriv 2.01, SVC (2), Sverdlov (2), Svir, Sylvia, Taiwan (2), Terror, Tiny (12), Traceback (2), TUQ, Turbo 488, Typo, Vacsina (8), Vcomm (2), VFSI, Victor, Vienna (8), Violator, Virus-101 (2), Virus-90, Voronezh (2), VP, V-1, W13 (2), Whale, Yankee (7), Zero Bug.

BOOK REVIEW

Practical Unix Security

Many Unix insecurities are old, well-known to gurus world-wide, and blindly ported to new versions of system software. However, a significant proportion of so-called 'holes' are the result of poor system setup, and can be plugged with a little foresight, awareness and - most vitally - *information*. In this book, Simson Garfinkel and Gene Spafford cover a broad range of areas where security can be compromised. From basic Unix concepts and history, through a discussion of the paths by which a cracker can gain entry, to coping with an attack on the system, the potential problems are thoroughly outlined and positive advice given on their prevention.

One trick in writing such a book is providing useful advice on prevention, while not giving away information of use to a potential attacker. The authors strike a respectable balance. Throughout, the text gives specific solutions to general problems, with warnings about the common pitfalls to be encountered in day-to-day use. Even the humble shell is a source of many 'features' that can be subverted by an intruder to gain access, but most users aren't aware of how the programs they use daily can be turned against them.

Another Unix feature that is variously frowned-upon, ignored, misunderstood and generally abused is *Set-UserId* (SUID) which allows a program owned by one user to be run by any other user with all the file-access permissions of the owner. The index of SUID files with the *reasons* why they need to be SUID is a nice touch. SUID programs are a prime example for the need to apply the principle of Least Privilege: a program should not be given root privileges if at all possible.

Networking is increasingly used to provide fast, reliable communication paths between widely-separated users, but if uncontrolled this can provide an entrance point for anyone worldwide to have a bash at your system. In addition it means that *The Bad Guys* can do their damage from a safe distance. Part III of the book covers communications security, including UUCP, Internetworking and NFS in some detail, with advice on *sendmail*, FTP and other network services, plus offerings for securing local networks, such as MIT's *Kerberos* authentication system.

Prevention may be better than cure, but on any system more complex than a slide-rule it's unlikely one can be sure to get everything right first time. Knowing what to do if someone does get past all your carefully prepared defences is vital, but many people aren't prepared. (Some people still think they will get away for ever without making backups). When you find something going awry with your system, do you know how to recognise a problem and react to it? Unless the attacker is very fast, the system itself can be your best ally in finding out what's going on - if you know how to ask it nicely.

Part IV provides advice on identifying the threat, containing it, auditing the actions of the intruder and cleaning up afterwards. You should be able to discover the route by which entry was gained and prevent a recurrence - you might even track down the culprit!

'Programmed threats' are an increasing worry. Viruses are currently high-profile, stormin' through the PC population as they are, but there are other potentialities: disgruntled ex-employees leaving logic bombs to delete data; unscrupulous software writers 'protecting' their software; developers putting 'trap doors' in programs for testing and not removing them at distribution; and of course, intruders leaving deliberate trap doors to regain access later. The only defence against programmed threats entails a complete loss of trust - *everything* has to be checked and audited continually.

Naturally there are some things that aren't covered in any detail - some holes are due to system flaws that are very hard to correct or work around, and some problems can not be explained at all without giving the game away completely.

The book is sprinkled with Unix folklore and askance humour (can you guess George and Dan's passwords?) with parallel explanations where BSD and System V diverge. Many examples are given of scripts for the automation of testing and analysis, together with an extensive security checklist.

A great deal of ground is covered in specific examples. Forearmed with the information in this book, administrators should be able to deduce further problems and derive solutions as they arise. The book is not in itself a basic guide, but it should be an excellent companion to any introductory text on Unix, for these often have little to say on security matters. It should be on every Unix administrator's required reading list.

A final word about hackers. The word 'hacker' is now almost universally applied as a reference to *The Bad Guys*, bent on destruction. The demise of its original meaning - as someone who eats, sleeps and breathes computers, always wanting to learn more - is a source of sadness to many of those familiar with (or a part of) the history of computing. To call someone 'a pretty mean hacker' would once have been a supreme compliment. Many now prefer to apply the terms cracker, or simply vandal, to those who set out to disrupt systems.

Title: Practical Unix Security (512 pp.)
Authors: Simson Garfinkel and Gene Spafford
ISBN: 0-93717572-2
Price: US$29.95
Publisher: O'Reilly & Associates, Inc., 632 Petaluma Avenue, Sebastopol, CA 95472, USA. In USA: Tel 800 338 6887, Tel International: +1 707 829 0515.
UUCP: uunet!ora!bookquestions
Internet: bookquestions@ora.com

END-NOTES & NEWS

Sarah Hood holding the fort.

End-notes & News returns next month.

Proceedings of the First International Virus Bulletin Conference are now available. Contact Petra Duffield at the *VB* editorial office (address below) for details.

Delegates wishing to receive hard copy of the slide and acetate sets used at the conference should write or fax their request to *VB*. These will be sent out free of charge.

VIRUS BULLETIN

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

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

Editorial enquiries, subscription enquiries, orders and payments:

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

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

US subscriptions only:

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