ISSN 0956-9979 AUGUST 1997

VIRUS BULLETIN

THE INTERNATIONAL PUBLICATION ON COMPUTER VIRUS PREVENTION, RECOGNITION AND REMOVAL

IN THIS ISSUE:

• **Legal Trends:** Trend Micro is in the news again because of its court cases and claims over its email virus-scanning patent. Our lead news article on p.3 has more details.

• **This might be a virus:** Anti-virus developers are turning again to heuristic scanning methods to reduce resource-intensive disassembly and analysis work but keep their products' detection rates up. Carey Nachenburg looks at some of the issues on p.6.

• **Parisian pursuits:** Our occasional series on the the state of things viral in various countries, focuses on France this month. François Paget's report is on p.10.

CONTENTS

EDITORIAL

A Word in Your Ear

In the News column last month, *VB* reported that *Word* macro viruses had topped the 1000 mark. Although we did not report this detail, if the listing maintained by the University of Hamburg's Virus Test Center (VTC) is used, the date this auspicious event occurred was 20 June. In announcing the June monthly update of the VTC macro malware list, Professor Klaus Brunnstein, head of the centre, reported that June had seen a slow-down in the rate of appearance of both new macro virus families and new variants. June saw only slightly more than half the growth recorded in May. However, a few days ago I saw an indication that July is likely to have been 'catch-up month' for the macro virus writers.

"the number of known macro viruses will more than double in the last six months of 1997"

A recent macro virus detector update, dated 21 July, listed 1201 known viral, dropper and Trojan macros. As one of that program's developers is closely involved in maintaining the VTC macro malware list, I assume July saw the world's macro virus writers return to business as usual. If the last few months' growth-rate (around 200 to 250 per month) remains steady, the number of known macro viruses will more than double in the last six months of 1997.

Fortunately, to date, it has been relatively easy to add detection and (usually) disinfection of new macro viruses to most anti-virus products. This may be changing, however, with the appearance of increasingly complex polymorphism in macro viruses and with likely changes in the macro environment – the latter could be the most important factor. *Microsoft* is under pressure from a significant (though arguably not large) group of third-party or 'aftermarket' developers. This group makes a tidy living creating customized office 'applications'. Using schedulers, automatic data collection tools and the like, and by gluing these to components of the *Microsoft Office* suite via DDE and OLE, and often using some fiendish *Word* and *Excel* macros, these developers provide custom data collection and reporting applications.

This group, amongst others, wants *Microsoft* to provide better protection for their handiwork. Why? Because their macros contain tricks not easily gleaned from the *Visual Basic Programmer's Guide*. These developers see part of their value in this specialist knowledge, and they wish to maintain the advantage that knowing such things brings them. They also know that it is trivial to work around the protection *Microsoft* has thus far provided. The solution many such developers want is robust encryption of macro code to protect their investment. Fortunately, a non-encrypted representation of a macro must be available so the macro can run, and to date anti-virus developers have been able to reverse-engineer the macro language to code mapping. They have not, however, had much timely cooperation from *Microsoft*, if reports I hear from developers are reliable.

Can anything be done to alleviate the situation? Pressure *Microsoft*. Let the lads in Redmond know that as IT administrators, security consultants, CIOs and concerned users, you want the ability to prevent *Word* securely from ever loading and running macros from document files. Templates are a different matter, but most system administrators I've talked with agree there is no need for macros *in document files* within their organizations. Others want to retain their document-based macros, but would be happy to see options to prevent macros usurping internal *Word* commands and to prevent auto-macros from running. Given the apparent structure of much of *Word's* internals, I suspect these are less likely to be delivered, but the former would make a lot of network administrators and support staff very happy. For reasons it doesn't seem prepared to discuss, *Microsoft* seems to have refused to make such changes in *Word*. They had the perfect opportunity with *Word 8* and missed it, but with *Windows 98* looming over the horizon, and with the reasonable presumption of an *Office* upgrade on its heels, it may almost be too late to influence *Microsoft* for the better. If you are concerned about these issues, find and twist an important *Microsoft* ear now!

Having been so harsh on *Microsoft*, it should be pointed out that they do seem willing to talk about these issues with the anti-virus research and developer community, but tend to do so under non-disclosure, which doesn't suit some vendors. A new initiative, involving the *NCSA* and *EICAR*, is being established, and I wish it luck in its dealings with the *Microsoft Office* developers.

NEWS

Integralis Challenges Patent

On 9 July 1997, *Integralis Inc* filed a complaint against *Trend Micro Inc*, claiming *Trend's* earlier move to have a court ruling stop *Integralis Inc* and its UK parent *Integralis Ltd* is based on an invalid patent. The basis of the *Integralis* claim is what is known as a prior-art defence.

Integralis Inc's complaint was filed in the United States District Court in Seattle. It requests a judgment of invalidity and noninfringment of *Trend Micro's* patent, citing interference with *Integralis'* business relationships and unfair business practices. *Integralis* filed the complaint after *Trend Micro* received a US patent for a 'virus detection and removal apparatus for computer networks' and threatened *Integralis* with a charge of patent infringement directed at *Integralis' MIMEsweeper* product.

'One of the reasons we do not believe *Trend's* patent is valid,' explained Victor Woodward, president of *Integralis Inc*, 'is that *Integralis* was shipping *MIMEsweeper* in Europe months before *Trend* applied for its patent. For a patent to be valid, the technology must be novel and unique, and that was not the case here.'

Trend Micro has recently sued *McAfee* and *Symantec* with the same charge of patent infringement. 'This series of lawsuits and barrage of news releases provided the impetus for filing our complaint,' stated Woodward. 'We do not want our customers or business partners to be influenced or intimidated by threats.' If the complaint is successful, the *Trend Micro* patent will be ruled not infringed, invalid and unenforceable. Also, an injunction will be issued prohibiting *Trend Micro Inc* from further claims regarding infringement. Monetary damages may be awarded to *Integralis* if *Trend* is found guilty of unfair business practices ▮

Errata

Several unfortunate test result errors crept into the July DOS scanner comparative review. These errors will be corrected in any subsequent *VB* reprints or other republishing of those test results.

First, the *IBM AntiVirus* and *Cybec VET* speed results for the two passes of the clean file set were transposed. *IBM AntiVirus* generates checksums in the first, relatively slow, scan, but this is compensated for with subsequent scans being much faster if no infection has occurred.

We also reported that *Symantec's Norton AntiVirus* failed to detect two Byway variants and Dir_II. This was due to an oversight in our testing method. Re-testing the product shows that it clearly detects the samples of these viruses in the 'In the Wild File' test-set, making *Norton AntiVirus'* correct ItW File score 99.7% and ItW Overall score 99.8%.

Prevalence Table – June 1997

Virus	Type	Incidents	Reports
CAP	Macro	22	17.9%
NPad	Macro	15	12.2%
Concept	Macro	12	9.8%
ParityBoot	Boot	11	8.9%
AntiEXE	Boot	10	8.1%
AntiCMOS.A	Boot	8	6.5%
Form.A	Boot	5	4.1%
AntiCMOS.B	Boot	3	2.4%
Appder	Macro	2	1.6%
DZT.B	Macro	2	1.6%
Empire.Monkey.A	Boot	2	1.6%
Empire.Monkey.B	Boot	2	1.6%
Laroux	Macro	2	1.6%
ShowOff	Macro	2	1.6%
V-Sign	Boot	2	1.6%
Wazzu	Macro	2	1.6%
Wazzu.X	Macro	2	1.6%
Boring	Macro	1	0.8%
Concept.AL	Macro	1	0.8%
Date.B	Macro	1	0.8%
DMV.B	Macro	1	0.8%
EXEBug	Multi	1	0.8%
Flip	Multi	1	0.8%
Johnny	Macro	1	0.8%
Junkie	Multi	1	0.8%
Kampana	File	1	0.8%
Kompu.A	Macro	1	0.8%
MDMA	Macro	1	0.8%
Natas	Multi	1	0.8%
NYB	Boot	1	0.8%
Quandary	Boot	1	0.8%
Sampo	Boot	1	0.8%
ShareFun	Macro	1	0.8%
Stoned	Boot	1	0.8%
Viruz.729	File	1	0.8%
WelcomB	Boot	1	0.8%
Total		123	100%

A similar oversight, affecting results on all test-sets but the ItW Boot one, meant the results for *DrWeb* from *Dialogue-Science* were also incorrect. Results for *DrWeb* should read:

ItW Boot	94.4%	ItW File	99.2%
ItW Overall	97.2%	Standard	97.8%
Polymorphic	100.0%	Macro	99.5%

DrWeb's clean file scan time was also misreported; however, not having access to the exact test machine, the considerably longer time cannot be accurately retested ▮

IBM PC VIRUSES (UPDATE)

The following is a list of updates and amendments to the *Virus Bulletin Table of Known IBM PC Viruses* as of 15 July 1997. Each entry consists of the virus name, its aliases (if any) and the virus type. This is followed by a short description (if available) and a 24-byte hexadecimal search pattern to detect the presence of the virus with a disk utility or a dedicated scanner which contains a user-updatable pattern library.

Type Codes

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

Andry.565

CR: An appending, 565-byte virus containing the texts 'ViRuZ by Andry Christian' (this message appears twice; at the beginning and at the end of infected files). The virus infects only files starting with E9h (near jump) and recognizes infected programs by the word 6956h ('Vi') at offset 0006h.

```
Andry.565          FB3D BEBA 7504 B8DC ACCF 80FC 4B74 03E9 6A01 5053 5152 5657
```

Awake.1099

CR: An encrypted, appending, 1099-byte virus containing the text, displayed on 8 December, 'Awake Shake dreams from your hair My pretty child, my sweet one. Choose the day and choose the sign of your day The day's divinity First thing you see. A vast radiant beach in a cool jeweled moon Couples naked race down by it's quiet side And we laugh like soft, mad children Smug in the wooly cotton brains of infancy The music and voices are all around us. Choose they croon the Ancient Ones The time has come again Choose now, they croon Beneath the moon Beside an ancient lake Enter again the sweet forest Enter the hot dream Come with us Everything is broken up and dances. Jim Morrison'. Infected files have their time-stamps set to 00:00:00.

```
Awake.1099         81EB 6701 BE82 0103 F3BA AF05 03D3 8134 ???? 463B F275 F7E9
```

Babilon.1000

CER: A stealth, appending, 1000-byte virus containing the text 'BABILON'. All infected COM files have the word 5058h ('XP') at offset 0003h.

```
Babilon.1000       B462 CD21 3C03 7405 E844 03EB F381 C6D0 038C C22E 813C 4D5A
```

BatmanII.3372

ER: A stealth, appending, triple-encrypted, 3372-byte virus containing the text 'Bat Man II'. It infects on Find First/Find Next calls (e.g. the DIR command). Infected files' time-stamps are set to 62 seconds.

```
BatmanII.3372      BF0E 022E A100 002E 3105 4747 81FF 2602 72F5
```

Blin.1457

ER: An encrypted, polymorphic, 1457-byte virus containing the texts '[Treblinka V 3.01 by Blas Pascal] . Argentina . xx/08/1995 .' and 'anti-vir.'. All infected files have byte 1Eh at offset 0010h (initial SP value). The following template can be used to detect the virus in memory.

```
Blin.1457          3DCA B075 038B F8CF C1C8 083D 4B00 7407 86C4 2EFF 2EB3 00E8
```

Cheap.828

CEN: An encrypted, appending, 828-byte virus containing the text 'ChEaPeXe v2.0 Virus by WârßläDÉ '97 USA'. Infected files have the word 4257h ('WB') at offset 0003h (.COM) or 0012h (.EXE).

```
Cheap.828          565D 81ED 0801 2E8A 9640 048D B627 01B9 1903 2E28 1446 E2FA
```

Cheap.1052

EN: An encrypted, appending, 1052-byte virus containing the texts 'ChEaPeXe v1.0 ...by WârßläDÉ/LT…' and 'I loved you, I always loved you, but it was nothing but a waste of time. May you burn in the Hell, if it really exists, my little slut, or suffer what you made my fragile and tender heart suffer. This should be the right Doom for each fucking cheater ..... if it was, it wouldn't be this one the Hell. Ti amavo poiche' mi facevi tenerezza....adesso ti odio poiche' mi fai schifo a proposito: Tanti Auguri, Valentina'. All infected files have the word 4257h ('WB') at offset 0012h.

```
Cheap.1052         8DB6 0D01 B9FC 012E 8B96 0605 2E31 1446 46E2 F9C3
```

Coconut.1323

CN: An encrypted, appending, 1323-byte, fast, direct infector containing the text '[Virus coconut, by The King Lizard]'.

```
Coconut.1323       E8ED FF83 3EDB 0500 740E AD2B C133 C1D3 C0AB FF0E DB05 EBEB
```

Coconut.1940

CN: An encrypted, appending, 1940-byte direct infector containing the texts 'Hey! Look here! You've got a virus!!!', '[by King Lizard]', 'Virus coconut wishes you a merry christmas and a happy new year!!' and '*.COM'. All infected files have the word 4E49h ('IN') at offset 0003h.

```
Coconut.1940       3E83 BE93 0800 740F AD2B C133 C1D3 C0AB 3EFF 8E93 08EB E9C3
```

Coconut.2015, 2071

EN: Encrypted, appending, direct infectors containing the texts 'V I R U S C O C O N U T G R E E T S Y O U !', '(C) King Lizard (Spain-1997)' and '*.EXE'. Infected files have the word 4C4Bh at offset 0012h.

```
Coconut.2015       B9B2 0374 088B F7AD 86E0 ABE2 FAC3 E8A1 FFE8 E3FF B40B 80F4
Coconut.2071       B9C4 0374 088B F7AD 86E0 ABE2 FAC3 E8A1 FFE8 E3FF B40B 80F4
```

Coconut.2099

EN: An encrypted, appending, 2099-byte direct infector containing the texts '[(c) King Lizard]', 'Virus coconut wishes you a merry christmas and a happy new year!' and '*.EXE'. All infected files have the word 4C4Bh ('KL') at offset 0012h.

```
Coconut.2099        C380 BE11 0900 740F B997 068D B662 028B FEAC F6D0 AAE2 FAC3
```

Coconut.2324

CEN: Similar to Coconut.2015, but also contains the text '*.COM'. Infected COM files have the word 4C4Bh ('KL') at offset 0003h.

```
Coconut.2324        8DB6 3601 B968 088B FEAC F6D0 AAE2 FAC3 E803 FAC6 86ED 0901
```

Coup.1957

MCER: A multipartite, 1957-byte virus that contains the encrypted text 'Coup De Main : In Childhood taught me to Love Now that I Love Frenzied,Said me Forget !!!'.

```
Coup.1957           8B1E 8600 8306 8600 0483 2E13 0404 B800 9F50 0750 891E 8600
```

Dada.786

CR: An appending, 786-byte virus. The 'Are you there?' call (AX=ABBAh, Int 2Fh) returns AX=DADAh.

```
Dada.786            FCF3 A4B8 BAAB CD2F 3DDA DA75 03E9 8C00 8BFB B821 35CD 212E
```

Daffodil.525

ER: An encrypted, appending, 525-byte virus containing the text 'Hallo world!'. All infected files have the word DFDFh at offset 0010h.

```
Daffodil.525        9C5D F7C5 0001 75F6 5D55 81ED 0300 8DB6 0300 B9E9 002E 8134
```

Deltaplus.1331

CER: A stealth, encrypted, appending, 1331-byte virus containing the text 'I love you! Let me guide you through life Let me be your best friend Tell me & Bring me all your problems Let me take care of you And get an eternal life full of joy and peace of mind Jesus Christ$Delta Plus Virus - 03/97£.'. The virus marks all infected files by adding 100 years to their time-stamps.

```
Deltaplus.1331      07BE 0700 03F5 8BFE B908 053E 8AA6 0F05 FCAC 32C4 AAE2 FAC3
```

Die.387

CER: An appending, 387-byte virus containing the text 'FRODO LIVES'.

```
Die.387             5E83 EE03 56F8 B8FE FECD 2172 2C8C D848 8EC0 2683 2E03 0023
```

EEM.103.B

CEN: An appending, 103-byte virus infecting one file at a time. It converts all infected EXE files to COM-structure programs.

```
EEM.103.B           B440 B167 8BD7 81EA 6700 E80A 004F B440 B103 8BD7 CD21 C3CD
```

Exorcist.617

CN: An encrypted, appending, 617-byte virus containing the texts ' [ODIUM RELAPSE] ', 'Relapse The Cronic Odium','[Exorcist!dc^96]' and '*.com'. Infected files have the word 4344h ('DC') at offset 0003h.

```
Exorcist.617        CD21 CD19 80FA 057F 0BB8 085F B200 CD21 B201 CD21 5A2E 8137
```

Glitter.1462

CR: An encrypted, appending, 1462-byte virus containing the texts 'Glitter ver 1.03 , Coded by DDISARTHH, Hi Avi Guess Who? Greetings From Siddharth, Mumbai 400 092', 'Wish you a Happy Birthday Love Guess Who ?' and 'CHKL*.*'. Apart from COM programs, the virus infects DOS driver (SYS) files. Infected files have their time-stamps set to 30 seconds.

```
Glitter.1462        83EE 050E 07C6 441D 908A 5420 B98E 05BB 2300 3010 43E2 FBC3
```

HLLC.16052

EN: A companion, 16052-byte virus written in Turbo C++. The virus infects one file at a time and only on drive C. It does not hide the newly created COM programs. Apart from the standard compiler messages, the virus contains the text 'Welcome TO S.R.M Engg. College Your Computer May be infected by Some virus Contact:- Ebi Joyel Dhas.Y , Lecturer Department of Computer Science and Engg. Press 'E' to contine:'.

```
HLLC.16052          E883 1059 807E FE0A 7409 807E FE14 7403 E902 0133 C050 E850
```

HumanGreed.666.B

EN: An encrypted, overwriting, 666-byte virus containing the texts 'That is not dead Which can eternal lie Yet with strange aeons Even death may die LiVe AfteR DeATH...Do not waste your time Searching For those wasted years! (c) Thanks to Raver and Metal Militia/IR Maria K - Life is limited, love is forever... Open to reality, forever in love... Program too big to fit in memory ***HUMAN GREED*** The answer of all evil on earth! Do You Belive?Farwell!', '*.EXE' and '*.COM'.

```
HumanGreed.666.B    BE31 018B 1617 01B9 2301 902E 3114 83C6 02E8 0300 E2F5 C3C3
```

Island.3551

MCER: A multipartite, polymorphic, stealth, 3551-byte virus containing the texts 'TIME', 'TBDSK', '.EXE.COM', 'ADINF', 'ANTI', 'AIDS', 'DRWEB', 'TB', 'SCAN' and 'CHK'. The virus inserts 120 bytes of its polymorphic decryptor inside the original program and appends the rest of its code.

```
Island.3551         C1B9 3129 A199 1109 8179 F1E9 6159 D1C9 4139 B1A9 2119 9189 (files)
Island.3551         A32C 7C06 8EC0 B808 02BA 8000 33DB B90A 00CD 1307 EA32 0400 (MBR)
```

Morph.3500

CER: A stealth, polymorphic, appending, 3500-byte virus containing the texts 'RELIGIOUS VOMIT! MORPHINE-A VIRUS 0.6.4', '[Morphine-A] 0.6.4 by Ren Hoük BA.Argentina', 'ANTI-VIR.DAT', 'CHKLIST.MS', 'CHKLIST.CPS', 'ZZ##.IM' and 'Greets to: PJanes,Rat,Largus & the girls Kill the talking bastard! kill him! Juap! ok..rec-tunn stolen from Vlad Mag.' Infected files time-stamps are set to 56 seconds. The template can be used to detect the virus in memory.

```
Morph.3500          3DFE 5425 F681 FE89 6B75 F0B4 FFBE CDAB CFB9 F600 2E83 3677
```

Zahak.906

CER: An appending, 906-byte virus, containing the texts 'C:\COMMAND.COM', 'CHKLIST.MS', 'CPS' and 'Zahak!'. The payload triggers randomly and overwrites the contents of a hard disk. The virus infects COMMAND.COM by overwriting its last 906 bytes (usually filled with zeros).

```
Zahak.906           80BC 0901 AA75 03E8 DB00 CD2D 81F9 AAAA 7403 E884 0258 8ED8
```

FEATURE 1

Future Imperfect

Carey Nachenberg
Symantec AntiVirus Research Center

The word *heuristic* comes from the Greek *heuriskein* meaning 'to find'. Today, within computer science, the term is almost exclusively used to describe sets of rules that tend to solve complex problems quickly. Such heuristics may yield suboptimal solutions, but tend to offer the benefit of speed improvements. For instance, the 'Travelling Salesman Problem' is a classic in computer science, for which many heuristic solutions have been devised.

Englebert the Twine Salesman and Heuristics

Englebert, a twine salesman, has to visit a number of different cities just once and wishes to minimize his travel time. If Englebert had to fly to Los Angeles, Oregon, Paris, and Rome, he would minimize his journey time by visiting LA first, then Oregon, followed by Paris and then Rome. If he travelled from LA to Rome, back to Oregon and then to Paris, he would spend a great deal more time in the air. Obviously, the first solution is better; in any case, it seems like a trivial task to plan Englebert's travel schedule.

However, what if Englebert had to travel to 500 cities? To find the absolute 'best' itinerary would take today's fastest computers years. Spending this much time on a calculation is infeasible, so computer scientists devised heuristic algorithms to help solve such problems faster. A heuristic algorithm makes assumptions about a given problem which allow it to cut down on computation time and still produce acceptable results. Many heuristic algorithms yield near-optimal solutions but there are no guarantees. Surely, Englebert will be satisfied to travel 100,055 miles if the best he could do would be 100,000 miles, especially if he doesn't have to wait several hundred years for his itinerary.

So, why the digression into computer science theory and twine salesmen? Well, the anti-virus field has its own set of problems to solve – some of which are considered unsolvable. In other words, it is impossible to come up with the exact, best solution to the problem in any finite amount of time! Heuristic algorithms lend themselves to such problems.

Virus Detection: An Inexact Science

The task of determining whether a computer program is a virus is unsolvable. It is provably impossible to write a program that is capable of determining, with a 100% success rate, whether any program is infected or not, considering all the possible viruses that could ever be or have been written. Were it possible to solve this problem, the major anti-virus vendors and MIS personnel would be dancing gleefully in the streets! It would mean the end of costly-to-develop and difficult-to-distribute virus updates, and no more troublesome false positives.

Consequently, anti-virus researchers have devised several innovative heuristic methods to help detect the tens of thousands of known computer viruses, and more importantly, the future, currently unknown ones. The most familiar technique is known as signature scanning. Most people do not think of signature scanning as a heuristic technique, but it is.

An anti-virus program that performs signature scanning searches for a number of 'signatures' in each potentially infected object. Signature scanners have a different signature for each virus they can detect; a signature is a short sequence of bytes extracted from the body of a given virus. Anti-virus companies employ automated systems and dozens of researchers to analyse viruses and extract viable signatures. This sequence of bytes should be unique to the virus and represent only a small proportion of the total virus. The uniqueness reduces the likelihood of the anti-virus program falsely identifying noninfected programs as infected. The small size prevents anti-virus data files growing to hundreds of megabytes in length (in essence, the signature data file would have to include a full copy of every detected virus if this condition were not met).

A signature scanner can identify whether a given program contains one of its many signatures, but cannot guarantee that the program is actually infected with the associated virus. Users trust the guess of the anti-virus program, since the odds are in its favour. However, the identified program could contain random data that coincidentally resembles the virus or actual legitimate program instructions that by chance match the bytes in the virus signature. So, while there is an extremely high probability that an identified program is a virus, a signature scanner cannot confirm this. Consequently, we could call signature scanning a heuristic algorithm. [*That no developers elect to do completely exact identification does not mean it is impossible for many viruses. Exact identification, properly implemented using scanning techniques, could not be labelled 'heuristic'. Ed.*]

Signature scanning is the technique most widely used in anti-virus programs today. While it does have its faults, it is very effective at detecting viruses for which the anti-virus program knows a signature. Unfortunately, virus writers are vigilant, constantly creating new viruses. In most cases, signatures added to the anti-virus data files in order to detect earlier viruses are powerless to discover new ones. With the ubiquitous nature of the Internet, new viruses can be made widely accessible within minutes. These factors contribute to the need for anti-virus technology capable of detecting viruses without signatures, and without the slow and expensive process of virus analysis.

Enter Heuristics

As we have seen, even the most commonly-used anti-virus products employ some form of heuristic logic. However, in the anti-virus industry, the term 'heuristics' is invariably used to describe programs that detect viruses based on analysis of code structure, behaviour and other attributes. For the rest of this article, the terms 'heuristic scanner' and 'heuristics' will be used to denote such technology.

There are two ways to catch a criminal. Police officers can dust a crime scene for fingerprints. If they find any they can check against the database back at the station. If a match is found they can pursue and arrest the suspect.

Unfortunately, many criminals are first-time offenders and have not yet been fingerprinted. The officers can try a different tactic. They can observe each person they come into contact with and assess the likelihood of them being a criminal. The officers can make a reasonable assessment that a person wearing a bullet-proof vest and carrying a shotgun is up to no good, and arrest them. On the other hand, the officers would probably only take a quick glance at a nanny carrying a bouncing baby before moving on. Of course, they might inadvertently arrest an innocent person and occasionally miss some wrongdoers, but a well-trained officer is likely to have a high success rate.

Anti-virus programs that employ heuristics use an analogous approach to detect computer viruses. Each time they scan executable code, they scrutinize its overall structure, programming logic and computer instructions, any data areas it contains, and a number of other attributes. They then make an assessment of the likelihood that the code is virus-infected. Like the police officers, the heur-istic scanners will fail to detect or recognize some of the 'baddies', and will occasionally identify innocent programs as being infected.

According to industry insiders, today's state-of-the-art heuristic scanners achieve 70 – 80% detection of new and unknown viruses. These rates are commendable given the difficulty of the problem. While most heuristic scanning technologies have achieved similar new virus detection rates, they vary widely in their propensities for falsely identifying clean programs as suspect. Some popular anti-virus products which falsely identify many clean programs with their heuristic engines have given heuristics a bad reputation, which is not necessarily deserved.

A big plus for heuristic scanning is that it can detect viruses before they can run and infect a computer. As with signature scanners, you can initiate an 'on-demand' heuristic scan of a new program or diskette before it is used. If you run an 'on-access' scanner with heuristic scanning technology, it can detect a high percentage of new viruses as they arrive from the Internet or are saved from email attachments.

In contrast, other generic anti-virus technologies such as 'behaviour blocking' and 'integrity checking' require that the virus executes on the host computer and exhibits suspicious (and potentially harmful) behaviour before it can be detected and stopped. Both heuristic and signature scanning get a positive checkmark in their ability to stop a virus before it has a chance to wreak havoc on your computer.

How do Heuristics Work?

Heuristic analysis techniques researched to date can be classified into 'static heuristic' and 'dynamic heuristic' architectures. There are heuristic scanning programs that are hybrids of these schemes. The primary difference between the two is the use of CPU emulation to search for virus-like behaviour. For now, we will discuss the attributes common to both architectures.

The typical heuristic scanner (if there is such a thing) has at least two phases of operation. In the first, the aim is to catalogue what behaviours the program is capable of exhibiting. The scanner starts by determining the most likely location for a virus to attach itself to the code. This is important because some programs are many kilobytes or even megabytes long. Performing detailed heuristic analysis on such large programs would be excruciatingly slow. Given that most DOS-based computer viruses are only a few kilobytes in length, a well-designed heuristic scanner can significantly limit the areas to scrutinize in a file. Most often, these regions will be the first and last few kilobytes of the file.

Once the heuristic scanner identifies the likely area of viral infection, it analyses the program logic from that region to determine the capabilities of the code. This is itself an extremely difficult problem since there are so many different ways to write a given program. For instance, consider the following two sequences of instructions.

Example 1:

```
B8 00 4C       MOV    AX, 4C00
CD 21          INT    21
```

Example 2:

```
B4 3C          MOV    AH, 3C
BB 00 00       MOV    BX, 0000
88 D8          MOV    AL, BL
80 C4 10       ADD    AH, 10
8E C3          MOV    ES, BX
9C             PUSHF
26             ES:
FF 1E 84 00    CALL FAR [0084]
```

Both snippets cause a program to terminate and return to the DOS prompt. However, the sequences of machine code bytes look entirely different. The first code sequence calls the operating system using a simple, common technique, while the second uses a much more roundabout approach.

Given that there is an infinite number of ways to write such a snippet of code, it may seem impossible to examine the sequence of bytes to glean any information at all. Luckily, most DOS viruses use straightforward techniques like the first example above. In any case, how is a heuristic scanner

to detect this type of behaviour? Static and dynamic heuristic implementations accomplish this task using markedly different techniques.

A static heuristic scanner recognizes various program behaviours in several ways. It can maintain a database of byte sequences like those above, associating each sequence with its functional behaviour. The scanner can use simple wildcards to help match bytes that may vary from virus to virus. The following sets of byte sequences illustrate this for two common program functions:

Terminate program:

```
B8 00 4C CD 21
B4 4C CD 21
B4 4C B0 ?? CD 21
B0 ?? B4 4C CD 21
```

Open file:

```
B8 02 3D BA ?? ?? CD 21
BA ?? ?? B8 02 3D CD 21
```

These strings should look strikingly similar to the virus signatures published monthly in *VB*, and for good reason. They are! However, standard virus signatures are used to define a specific virus strain. Signatures like those above help to identify whether a program contains the logic to exhibit a given behaviour. If a heuristic scanner locates such a string, it does not necessarily mean that the program is viral, but indicates that the program may be capable of the associated behaviour.

In addition to a database of behaviour signatures, static heuristic scanners may also use more elaborate logic to seek out and recognize other complex behaviours. For instance, encrypted and polymorphic viruses often have simple decryption loops to unscramble the virus when an infected object is launched. The byte sequences comprising these decryption loops can vary widely in appearance; even so, it is possible to write fairly simple subroutines to recognize a large percentage of them. If the 'decryption loop' detecting subroutine of a heuristic scanner locates one, it will catalogue this behaviour.

While static heuristic scanners rely on behaviour signatures and code analysis algorithms to catalogue code behaviour, dynamic heuristic scanners use CPU emulation. After initial sanity checks, a dynamic heuristic scanner loads the suspect code into a virtual computer and emulates its execution. While the program runs, its behaviour is monitored.

Any time the virtual operating system is called by the code in the emulator, the scanner notes the behaviour then allows the emulated program to continue executing. This gives the dynamic scanner an advantage over its static heuristic cousin.

An analogy may help. Assume I am the ambassador (and a crafty spy) at the US embassy in Votslovia. I can take many different paths to spy on the top Votslovian nuclear scientist who works four blocks away. I could go one block north, two blocks east, and one block north again. Alternatively, I could go one block west, two blocks north, and three blocks east. Both routes would get me there. There are many other paths to his lab, just as there are many possible sequences of computer instructions to achieve the same goal.

A Votslovian intelligence official could determine that I was spying in several different ways. Firstly, he could choose to monitor a likely route between the embassy and the lab. If he observed me there, he could report it to the authorities. If I took some other route, he would probably miss me. Of course, he could choose to recruit an additional agent to monitor another route. However, there are many, many possible routes that I could take.

Alternatively, the officer could stake-out in the scientist's laboratory, hiding behind the ficus tree in the back office. Doing this, he has no need to concern himself with how I get to the nuclear lab. Instead, he just snaps a picture showing that somehow I arrived there.

The former strategy is analogous to that employed by static heuristic scanners during the code analysis phase. They look for different behaviour and their success is highly dependent on how the suspect code implements its logic. If the code being analysed uses an obfuscated method of calling the operating system, a static scanner may fail to detect this behaviour, just as the intelligence officer might miss my prying if he monitored only a few routes to the lab.

The latter strategy is analogous to that used in dynamic heuristic scanners. The dynamic scanner lets the program run freely within the virtual machine. The program can use any logic it likes to get its job done, but eventually it will call on the operating system, and when it does, the results of all its computations and machinations will be made clear.

Thus, it seems that dynamic heuristics should be much more effective in analysing and identifying the behaviour of a program. However, they can also be much slower than their static counterparts. CPU emulation is a relatively slow process, and is usually more protracted than scanning for signatures in selected regions of code. Furthermore, CPU emulation is susceptible to the logic and whims of the code being emulated. For instance, what if we wanted to detect the following virus using dynamic heuristic techniques?

Virus pseudo-code:

```
1.  If current hour is even, goto 3
2.  Goto 2
3.  Infect a new program using simple,
    identifiable computer instructions.
4.  …
```

A dynamic heuristic scanner would emulate the program within its virtual computer. The emulator would execute the first instruction, and if the current hour happened to be odd, it would then execute instruction two indefinitely, and not observe any of the virus' other behaviours. If the emulator fails to provide the virus with what it wants, the virus' logic may prevent it from executing its telltale behaviours and giving itself away.

On the other hand, our static heuristic scanner would detect all the behaviours in the above program since it is not constrained by the program's logic. The static scanner looks throughout the virus' body for behaviours, regardless of whether or not they would be reached during a typical execution of the virus.

After analysing the program's logic and instructions, the heuristic scanner searches for any suspicious data or strings stored within the likely viral regions. Virus writers often include expletives or the word 'virus' in their code. Since most programs do not contain such words, their presence is an indicator that a program may be infected. The heuristic scanner logs these strings and other telltale signs of infection for use during the second phase of the process. [*One would hope such 'signs' are given a very low weight, if used at all! Ed.*]

Once a heuristic scanner has obtained a set of behaviours and attributes, the second phase begins: analysis of the observed behaviours. In general, the second phase is much the same for both dynamic and static heuristic scanners. From its list of code attributes and behaviours, the heuristic scanner must decide if what it has 'observed' is virus-like or not. This is another extremely difficult problem!

The heuristic scanner must know, for instance, that appending COM viruses may use the following behaviours when infecting a new executable file:

- find a new COM file in the current directory
- open the file
- seek to the end of the file
- seek to the top of the file
- modify the instructions at the top of the file by writing out three or four bytes to the top of the file
- seek to the end of the file
- write several hundred bytes at the end of the file
- close the file

If all these behaviours were observed during the first phase, it could report with high confidence that it had detected an appending COM virus. Unfortunately, the first phase rarely obtains such a complete list. Code analysis technology – static or dynamic – has its flaws, and will occasionally fail to detect some behaviours of the target code. Therefore, the second phase must be able to make educated guesses as to how much like a virus the code is.

For instance, what if only the fourth through last of the behaviours above were observed during the first phase? For most virus researchers, this subset of behaviours would still raise a big red flag. However, it is one thing for the heuristic scanner to report a possible infection to a researcher who can examine the code and verify the assessment. It is another thing entirely for the scanner to report this to an end-user who is unlikely to have virus analysis skills.

Given that the set of observed behaviours may be incomplete, the behaviour analysis component of the heuristic scanner must be designed with care. If this component is too stringent in its requirements, it will have difficulty detecting a significant number of viruses. On the other hand, if the analyser is too lenient, the resulting product may be overly susceptible to false identifications. You can now see why some anti-virus products with heuristics frequently cry wolf; their designers opting for higher detection rates with fewer behavioural requirements at the expense of higher false alerts.

To date, a number of approaches have been used in making this behavioural assessment. For example, *IBM AntiVirus'* boot virus heuristic scanner uses a neural network to analyse behavioural information. *Symantec's* Bloodhound technology, however, uses an expert system to analyse the catalogued behaviours and assess the likelihood of viral infection. There are probably as many different behaviour analysers as there are heuristic scanning products, but this technology will evolve significantly over the coming years.

Conclusions

Heuristic virus detection is becoming increasingly important, due to the ever-growing number of computer viruses and their ease of transmission over the Internet. In its infancy, and with an unfortunate reputation for being slow and prone to false positives, heuristic scanning will undoubtedly mature and become even more effective. The explosive growth of viruses has made one-by-one virus analysis extremely costly and as anti-virus developers strive to protect their customers better and reduce costs, heuristics will become more and more prevalent.

While heuristic scanning is likely to become a standard component in anti-virus products, it will never entirely replace signature scanning technology. Why? In addition to replicating, many computer viruses intentionally or inadvertently corrupt data and programs on the computer. The One_Half virus, for instance, encrypts the host computer's hard drive two cylinders at a time on each boot-up. While a heuristic scanner can indicate that there is probably a virus present, and a heuristic repair system may even be able to remove the virus generically, such a generic anti-virus product would have no idea that the hard drive has been encrypted, how to decrypt the drive, or what other damage might have been caused. It would be a tragedy for the user to remove the virus, only to find next time they turn on their computer that all their data was lost.

Some combination of signature and heuristic scanning may be able to detect and repair simple variations of existing viruses. It is unlikely, however, that a heuristic product will ever be capable of generically detecting and repairing such complex corruption on its own.

Carey Nachenberg is a programmer and anti-virus researcher holding the position of Chief Architect at *Symantec AntiVirus Research Center*, Santa Monica, CA, USA. Carey was the subject of a recent *Insight* article – see *VB*, April 1997, p.6.

FEATURE 2

The French Connection

François Paget
McAfee, France

France is the largest country in Western Europe, representing approximately one-fifth of the European Union area. Sixty million inhabitants share fifteen million microcomputers, with approximately two-thirds of these in businesses and the rest in the home.

The first serious study concerning the evolution of computer viruses in France was begun at the end of 1992 by the association for Recherches et Etudes sur la Criminalité Informatique Française (RECIF). At that time, when there were 2600 viruses worldwide, the total number of viruses in the wild in France did not exceed forty. To date, more than 450 viruses have been responsible for all reported virus attacks in France.

In 1995, before the macro virus infestation began, RECIF estimated that 2% of French microcomputers had experienced a virus attack in the past year and that the ensuing damage topped US$40 million (200 million francs).

Since 1992, RECIF has recorded the following number of virus alerts in France:

1992	**1993**	**1994**	**1995**	**1996**
410	720	1300	3150	6120*

*The 1996 figure is courtesy of ‘Confdentiel Sécurité’.

With the advent of macro viruses, however, the virus reporting rate has grown dramatically. For example, in June 1997, all the major French corporations seem to have contracted WM/CAP.A. It has not been exceptional to find more than 8000 infected files in one company.

The number of viruses believed to have been written in France is near seventy. For most, the creation year is known:

1988	**89**	**90**	**91**	**92**	**93**	**94**	**95**	**96**	**97**
1	0	3	3	7	15	6	19	8	4

Classified by type, it is clear that, as with viruses in general, most French-written viruses are file infectors:

File infector	52
Boot infector	11
Multipartite virus	1
Macro virus	4

The first French virus was the boot infector E.D.V. Discovered in the city of Le Havre in 1988, it was first named Cursy. Later, in January 1990, it was rediscovered and given the name E.D.V. or Stealth Virus. It was one of the earliest viruses to use stealth techniques and was first featured in *VB* in March 1990.

Various virus encyclopædias indicate three French viruses created in 1990 – Mardi_Bros, TCC and Paris. However, reports of these have been rare. Also, one of the first virus generator kits, GenVirus, originated in France that year.

In May 1991, a popular computer magazine distributed a cover diskette infected with the Israeli virus Frodo (alias 4096). Seventy thousand infected diskettes were probably involved, but only two companies affected by this virus lodged a complaint! However, as a result, two people working in the duplicating firm that made the diskettes were severely punished, receiving a two year suspended sentence and a 100,000 franc fine, in addition to a damages charge of more than three million francs. In 1995, this sentence was overturned. The Supreme Court of Appeal quashed the appeal in 1996 and reopened the case. Six years after the event, this lawsuit is still not concluded.

Until the end of 1992, the twenty or so French viruses were fairly harmless. In fact, the Malaise family internally documents how to disable its reproductive capability!

```
Welcome into the virus
© 1990 by InfoViruses Laboratories
V-IVL110 (COM & EXE)
To inactivate me, just set to “*” the byte in
brackets: [#]
Next time, be more prudent!
```

This tradition of deliberately non-damaging French viruses was broken with the Fichv clan. Fichv.2_0 and Fichv.2_1 are destructive in March, Fichv.Fexe in April. The payload consists of overwriting the first six sectors on each head of the hard drive. The first four sectors contain the repeated message, which translates to ‘Fichv 2.1 has had you’:

```
****Fichv 2.1 vous a eu**
```

The first large-scale infections traceable to French viruses began in 1993. The less virulent ones were due to file infectors like the families Dual_GTM (aliases Beware, and Greviste or Striker in English) and Chaos_3 (alias Chaos_Years), and Com2S, and Hidenowt (distributed by many hypermarkets through boxes of pre-formatted diskettes containing an infected DE.EXE file).

Chaos_3 is destructive, setting a random trigger date to fewer than three months after the initial infection. The payload consists of hard disk overwriting followed by display of the message:

```
YOUR DISK HAS BEEN DESTROYED BY THE “CHAOS
    YEARS” VIRUS…
ACCEPT MY SINCERE SYMPATHY !
SIGNED : THE DARK AVENGER.!
```

The first French virus to spread worldwide was Jumper.B. This quite unoriginal boot sector virus, described in *VB* in April 1995, was discovered early in 1993 near Rouen and

regularly features on wild-lists and virus prevalence reports. In most cases, Jumper.B only replicates. However, on slow computers, when the virus payload activates, it effectively locks the machine by repeatedly displaying the character '∈'. Jumper.B has numerous aliases, which has led to some confusion: Jumper.B, Boot_FR, 2KB, Silly_BP, Epsilon, Neuville, VIRESC, BFR, PM5, and French_Boot.

Anecdotally, Neuville is the name of a commune near the city of Rouen and VIRESC is the acronym of 'VIRus de l'Ecole Supérieure de Commerce' (Virus of the Higher School of Commerce). In 1994 and 1995, before the macro virus infestation began, more than 20% of virus alerts in French territories were due to Jumper.B. There are three Jumper variants, but two of them have almost disappeared because they damage the boot sector of floppies.

Later, in September 1993, AntiEXE appeared in France (see *VB* October 1994). The virus' origins are contested; for example, *VSUM* indicates Russia. In my opinion, this boot sector virus has a Parisian origin. In 1994 and 1995 it was also one of the commonest viruses here (at that time 80% of virus alerts involved system viruses, of which 75% were Jumper.B, AntiEXE, Form and Parity_Boot).

Few new French viruses appeared in 1994, with just a few written by someone nicknamed Turbo Power. These file infectors were Failure, Cowa-Bunga and above all Zarma. The latter pair contain an encrypted message about Claudia Schiffer. This name is spelt 'Schieffer' in Cowa-Bunga – an error much mocked by the French underground community. Zarma rectified this spelling mistake and is of interest for some of the anti-debugging techniques it employs.

The French virus writers' most prolific year was indisputably 1995, with the Werewolf pack of viruses fully described by Igor Muttik in the February 1997 issue of *VB*. These viruses have been commonly reported in France because their author used numerous BBSes to spread them.

The Werewolf variants, encrypted or not, contain clear statements of their creation dates. According to these claims, the first Werewolves date from 1994 and the last from 1996. In fact, they all appeared between November 1995 and January 1996. Far behind boot sector viruses and macro viruses, the Werewolf viruses are the file infectors most encountered in France (they are in twenty-second position in the latest RECIF statistics).

During 1995, two boot viruses affected the East of France particularly badly. Initially nicknamed Goering, these viruses are now known as Form.G and Form.N. When their payloads trigger on 1 January 1997, they immediately destroy the first physical sector of the hard disk and thus can spread no more.The following message is present in both viruses:

```
This is the Hermann GOERING Virus. Heil
    HITLER !
Thanks to Martin BORMANN, Joseph GOEBBELS,
Heinrich HIMMLER and Rudolf HESS.
Sieg! »
```

The early macro viruses were mainly dependent on the national language version of *Word* in use, and thus were not a great threat in France, as most were produced for the English version of *Word*. This early isolation of French *Word* users from macro viruses was aided by the relatively low Internet usage in France.

However, within the last six months, the situation in France has changed – macro viruses account for more than 70% of alerts. We seldom see file viruses, and boot sector infectors are increasingly rare. The viruses most commonly reported now are CAP.A, MDMA, NPad, Jumper.B, AntiEXE, Parity_Boot, Form, Appder, and Wazzu.

Despite macro viruses changing the virus report patterns, to date, only four macro viruses seem to have been created in France. These are Concept.B:FR, Nuclear.B, Wazzu.AF, and Appder.A (aliases FunYour, NTTHNTA). Only the latter is regularly encountered here. Concept.B:FR is a laboratory virus, created in a French bank, and consequently never seen in the wild.

Thus far, French virus creators have not been organized, but this situation seems to be changing. Groups like SLAM and MJ13 now have some French members.

Although creating viruses is not illegal in France, the authorities take a keen interest in the phenomenon. Both services of the Criminal Investigation Department (*Police Judiciaire*) have the ability to tackle the problem. The Home Office also has a virus specialist section in the Directorate for Surveillance of the Territory (DST).

Two professional associations represent the principal actors in the French anti-virus field. Already mentioned, RECIF is only interested in computer malevolence. French anti-virus specialists, including computer security managers from large French companies and Ministries belong to RECIF.

Club de la Sécurité Informatique Français (CLUSIF) is interested in all aspects of computer security. With approximately 270 members, it principally represents distributors, service providers and security product manufacturers. Its Logical Attack Commission addresses anti-virus concerns.

Over the last ten years, CLUSIF statistics show a continuous increase of malevolence, which accounts for 62% of computer disasters in France (24% are due to accidents and 14% to system or user error). The frequency of virus reports has increased, but the consequences are better controlled than in the past. Few deliberate software attacks have been reported to CLUSIF.

Looking to the future, CLUSIF predicts further increases in reported computer malevolence. Contributing causes include France's continuing economic crisis, perceived instability in some areas of computing, the emergency in the telecommunications sector, and the lack of computer ethics education. The communications 'explosion', particularly with the Internet, may also have an impact on the virus phenomenon in France in the future.

VIRUS ANALYSIS 1

Protected Mode Supervisor?

Igor Daniloff
DialogueScience

Since their introduction, PCs have become increasingly complex through advances in both hardware and software. Computer viruses are also becoming more complex and intricate as their authors try to adapt them to changes in the computing environment.

Now there are viruses that infect PC boot sectors, DOS, *Windows*, *Windows 95*, *OS/2*, *Macintosh* and *Linux* program files, as well as documents created in *Word* and *Excel*. Virus authors have devised stealth techniques to help avoid detection, and anti-debugging and anti-antivirus mechanisms to make initial detection, then analysis more difficult. They have incorporated polymorphism in boot sectors, files, and memory, to make detection more laborious and time-consuming for anti-virus designers. Since the release of i386 processors, viruses have begun to use 32-bit instructions. Some polymorphic viruses employ 32-bit operands in their decryptors.

Ultimately, viruses aim to survive and gain the upper hand under existing conditions, using all conceivable software and hardware techniques. With the emergence of 286, and later 32-bit i386 processors, came protected (or virtual) mode operation. Thus far, virus authors have not successfully harnessed protected mode. Some have tried to master it, but their attempts have been unsuccessful because of clashes with important operating system components.

In 1994, the boot virus PMBS was the first to tackle protected mode, but could not cope with other applications or drivers (EMM386, *Windows*, *OS/2*) also using that mode. In the same year, the viruses Evolution.2761 and Evolution.2770 succeeded in tapping part of the power of protected mode, but only when the processor was in real mode. These viruses replaced the actual interrupt vector table with their own interrupt descriptor table (IDT), which they loaded with an IDT register. How did the Evolution viruses use this technique in everyday life? I doubt there is a PC user who runs their i386 – Pentium in real mode.

Although the i386 processor made its debut long ago, viruses have still failed to master its powerful protected mode. I believe that virus designers have cherished this hope for some time, and that one among them finally appears to have realized it.

PM.Wanderer, apparently written in Russia, is a file infector which uses a crude form of protected mode. It is surprisingly stable, interacting more or less correctly with other programs that utilize this mode. The name is derived from the string ‘WANDERER,(c) P. Demenuk’.

A resident polymorphic virus, PM.Wanderer installs its resident part in memory and toggles the processor to protected mode by utilizing the documented virtual control program interface (VCPI) of the extended memory supervisor (EMS, EMM386).

Installation

On starting an infected program, the polymorphic decryptor decodes the virus body and passes control to it. The virus code determines a location in the upper addresses of DOS memory, writes itself to this memory, and hands control to the copy higher in memory. Then it restores the code of the infected file in the program segment (for EXE files, it also configures the addresses of relocated elements) and begins to install its resident component.

First, the virus checks whether there is an extended memory manager (EMM) in the system. It does this by retrieving the address of Int 67h (Extended Memory) through Int 21h function AX=3567h (Get Interrupt Vector), and checking whether the characters ‘EM’ exist in the EMS header. Then the virus verifies whether its resident part is already installed by calling function AX=BABAh of Int 21h and looking for the answer AX=FA00h.

If there is no active EMM, or the resident part of the virus is already installed (and in subsequent operations, if there is no VCPI or an error occurs installing the resident copy), the virus frees the memory reserved for installing the resident copy and passes control to the host program. This completes the life cycle of the virus in the system. However, if environmental conditions are favourable, the virus intercepts Int 01h and traces Int 21h looking, for the word 9090h (two NOPs) in the original Int 21h handler code of MS DOS versions 5.00 – 7.00.

If this word is detected, the virus retrieves the address of the Int 21h handler kernel, which is usually located in the high memory area, and writes this address into its body. This is subsequently used for calling the Int 21h handler kernel directly, while infecting files.

Then the virus verifies the presence of VCPI and reserves the physical addresses of four memory pages. It next retrieves the addresses of the VCPI, page table, and GDT (Global Descriptor Table. This consists of three elements: the first is the code segment descriptor, and the other two are used by the VCPI driver.). The virus writes a reference to the pages allotted by the VCPI driver into the page table, and retrieves the physical address of the memory page of the segment in which the virus is currently located. It also records the GDT and IDT registers. Next, the virus creates three (code and data) descriptors and a descriptor for the task state segment (TSS) in GDT.

Finally, it prepares the values for the registers CR3, GDTR, IDTR, LDTR (Local Descriptor Table Register), TR (Task Register), and the address CS:EIP of the protected mode entry point. Using VCPI tools, the virus toggles the processor into protected mode with the highest privilege level, known as supervisor.

In protected mode, the virus corrects IDT by creating two segment descriptors, then searches for the TSS descriptor. Next the virus defines two breakpoints, one at the first byte of the code of the current Int 21h handler (0000:0084h) and the other at the first byte of code at 0FE00:005Bh (linear address 0FE05Bh). This BIOS location usually holds the 'near jump to reboot'. The virus then corrects IDT to set debug exceptions at Int 01h and Int 09h. It also defines two handler descriptors; trap gate and interrupt gate.

After these preliminaries, the virus writes its code to the memory page and switches the processor back to virtual mode in order to free the DOS memory in upper addresses and to return control to the infected program. From this instant, the infected program begins its 'normal' work, but Int 01h and Int 09h have been redefined by the virus as trap gate and interrupt gate respectively.

Keyboard Handler

On receiving control, the virus-defined Int 09h handler verifies whether the two virus-defined breakpoints exist, and restores them if either has been zeroed. Using the register DR7, the virus checks whether the two breakpoints (0 and 1) are defined, without verifying their linear addresses. If either of the breakpoints is missing, the virus calls a procedure that instantly restores both to their initial status. The virus-defined Int 09h handler also keeps a close watch on the pressing of Ctrl-Alt-Del and 'resets' all breakpoints when this key combination is used.

Debug Exception Handler

The virus-defined debug exception handler verifies whether either of the virus breakpoints has been reached by checking the command address. If control passed to this handler from the 'near jump to reboot' in the BIOS, the virus resets all breakpoints just as the virus-defined keyboard handler does when the key combination Ctrl-Alt-Del is pressed.

If the exception was caused by the breakpoint in the original DOS Int 21h handler, the virus analyses the AX register to determine the function of Int 21h, and behaves accordingly. Prior to analysing this, the virus sets the resume flag (RF=1) in the stack's EFLAGS register that is intended to return control to the breakpoint. This flag is set should a debug exception take place while returning control to the breakpoint.

If Int 21h is called with AX=BABAh, the virus recognizes this as its 'Are you there?' call. If PM.Wanderer is installed it writes the value FACCh in the AX register and returns control to the original DOS Int 21h handler. On exiting from the DOS handler, the AL register is set to zero. The register value AX=FA00h informs the non-resident virus that a copy is already active.

If Int 21h is called with either AX=4B00h (start program), or AH=3Dh and the lower 4 bits of AL set to zero (open file for reading), the virus decides to infect. The virus writes its code to 9000:0000h (linear address 90000h), prepares a stack, and toggles the processor to 8086 virtual mode with an IRETD command at the third and least privileged level.

File Infection

In virtual mode (V-mode), the virus code verifies the last two characters (OM or XE) of the filename extension, creates a polymorphic copy, and infects files longer than 4095 bytes. PM.Wanderer does not infect if the seconds field of the file's time-stamp is 34, assuming that the file is already infected, nor does the virus alter file attributes. Therefore read-only files are not infected. Further, the virus does not infect a particular program with a seven-character filename. I could not find the name of this file: the virus defines it implicitly by computing the CRC of its name.

The virus does not take over Int 24h (DOS Critical Error Handler), so when critical errors (for example, writing to write-protected disks) occur during infection, the standard DOS query – Retry, Ignore, Fail, Abort? – is displayed. The virus infects a file by calling the DOS Int 21h handler directly, using the address obtained from tracing Int 21h at installation. The virus code is prepended to the header of COM files and inserted into the middle of EXE files, immediately below the header. Prior to this, the relocations field in the header is zeroed by moving the original program code to the file end. The 'real working code' of the virus is 3684 bytes long, but the size of infected files increases by more than 3940 bytes.

Exit from the V-mode of DOS-machine

The virus uses a smart technique to exit V-mode and to transfer control to the breakpoint of the DOS Int 21h handler that called the debug exception, so that DOS functions normally. Were the virus to infect a file while in P-mode, everything would be simple – it would be sufficient to execute the IRETD command. Since the virus has toggled to V-mode with privilege level three, it is not possible for the debug exception handler to switch back to P-mode. Therefore, the virus plays an elegant trick to surmount the situation.

If an error occurs during infection or while exiting from virtual mode, the virus calls Int 21h with AX=4B00h. When Int 21h is called with AX=4B00h, control jumps to the first command of the DOS Int 21h handler. This command contains a virus-defined breakpoint. Control must now be transferred to the debug exception handler in P-mode. However, the V-mode monitor discovers the need to process the next debug exception. The point is that the virus debug exception handler has not returned control to

the breakpoint and is still busy processing the current debug exception. Therefore, the V-mode monitor terminates the Int 21h call, aborts processing the current debug exception, and returns control to the breakpoint with the values stored in the registers of the previous Int 21h call.

Payload

If the debug exception handler is passed AX=3506h (such a call for getting the Int 06h address usually exists in all programs compiled from high-level languages, such as C and Pascal), PM.Wanderer scans the linear address space 0 – 90000h looking for a string that obviously belongs to the Russian integrity checker *ADinf*. If this string is found, the virus modifies it in order to disable the alerts *ADinf* usually raises on detecting changes to files and disks.

Search for the Virus in Memory

It is clear from the above that conventional memory scanning methods are incapable of detecting the resident copy of the virus at level zero privilege in protected mode. The resident copy can be detected only after toggling to the highest privilege level of protected mode with the help of GDT or IDT. However, this virus can be trapped by other conventional methods. Here, the linear addresses of the first two breakpoints (0 and 1) must be determined and compared with the values described above. The possible presence of PM.Wanderer in memory can be decided from these addresses. It is imperative that such operations be carried out only in a DOS session. In assembler language, this can be done as follows:

```
.8086
MOV AX,0BABAH    ;simulate the virus checking
INT 21H          ;its presence
CMP AX,0FA00H    ;did resident copy respond?
JNE ExitCheckMemory
.386P
MOV EAX,DR7      ;read register DR7
AND EAX,20AH
CMP EAX,20AH     ;are 2 breakpoints defined?
JNE ExitCheckMemory
MOV EAX,DR1      ;read linear address of
                 ;breakpoint 1
CMP EAX,0FE05BH  ;set at 0FE00:005BH in BIOS?
JNE ExitCheckMemory
.8086
MOV AH,9
MOV DX,OFFSET VirusIsFound
INT 21H          ;alert possible presence of
CLI              ;virus in the memory
JMP $+0          ;'hang up' system
ExitCheckMemory:
INT 20H          ;terminate operation
```

Test

After infecting several thousand files, the virus behaves like a 'lodger' with all infected files remaining operative. A file becomes inoperative only if, after infection, its stacks are located within the virus code. When infecting EXE files, PM.Wanderer does not modify the initial SS:SP values in the EXE header. As already mentioned, the virus is capable of reproduction only if EMS (EMM386) is installed in the system. If EMM386 is installed with the /NOEMS option, when the virus toggles the processor to protected mode, the system will reboot. The computer may also reboot at this point if QEMM386 is installed.

The virus loses its ability to replicate under *Windows 3.1x* and *Windows 95*. These operating systems cut off an already resident PM.Wanderer, because while loading they install their own handlers in the IDT and zero all breakpoints. Prior to terminating a session and returning to DOS, *Windows* restores the previous status of the interrupt descriptor table. On pressing a key in a DOS environment, the virus gets control, installs its own breakpoints, and continues its activities. Due to the absence of VCPI in a DOS session within *Windows*, the virus cannot return to protected mode there. For the same reason, the virus is also inoperative under OS/2.

Conclusion

PM.Wanderer is the first virus to utilize i386 protected mode and not conflict with the dominant *Microsoft* operating systems which also use that mode. It is possible that future viruses may completely overwrite the supervisor with their own code supporting the DPMI, EMS/VCPI, XMS, and Int 15h extended memory interfaces. Who knows?

PM.Wanderer

Aliases:	None known.
Type:	Memory resident in protected mode, polymorphic.
Infection:	COM and EXE files.
Self-recognition in Memory:	See description.
Self-recognition in Files:	Seconds field of time-stamps set to 34.
Hex Pattern in Files:	The virus is polymorphic, and there is no usable hex pattern.
Hex Pattern in Memory:	Virus works in protected mode; see description.
Intercepts:	In IDT: Int 09h for enabling breakpoints, Int 01h for infection.
Payload:	Patches the integrity checker *ADinf* in memory, disabling its alerting functions.
Removal:	Under clean system conditions, identify and replace infected files from clean backups or by reinstalling the software.

VIRUS ANALYSIS 2

SlovakDictator

Dr Cai-Gong Qin
Sophos Plc

Since the first *Word* macro virus, Concept, came to light in 1995, the computing community has observed that macro virus problems are getting worse. Although most anti-virus products are now honed to detect and disinfect a number of known macro viruses, *Word* macro viruses stand consistently at the top of virus prevalence reports (see recent issues of *VB*, p.3). The number of new *Word* macro viruses recognized by anti-virus companies is also increasing dramatically. This phenomenon represents a trend which sees virus writers turning their attention to macro viruses.

Soon after Concept was released, polymorphism became a real feature of macro viruses. The first reported 'polymorphic' macro virus, Outlaw, could only play with macro names by assigning different names with each replication, while the body of the viral macros remained the same (see *Virus Bulletin*, November 1996, p.12).

However, SlovakDictator stands out as being a truly polymorphic macro virus; considerably more complex than Outlaw. Although the virus contains a macro with the fixed name AutoClose, its body changes with each replication. So, how does its polymorphic engine work?

Overview

Unlike the Concept virus, which uses the macro AutoOpen, SlovakDictator uses AutoClose to receive control whenever a document is closed.

Most *Word* macro viruses are encrypted within an infected document using *Word's* execute-only feature. However, the macro AutoClose in SlovakDictator is not, allowing it to be easily extracted for analysis. That sounds very straightforward, but hopes of an easy analysis were quickly dashed by reading the macro source code in WordBasic. The code is full of confusing 'words'. For example, a typical for-loop looks like this:

```
For DMUBQJVPRCDA = PSOISDGFQR To
    GHKAHJVNLNQKEUN : Insert
ACFIDQANBQ$(DMUBQJVPRCDA) : InsertPara : Next
    DMUBQJVPRCDA
```

which is equivalent to:

```
For K = 0 To 179 : Insert V$(K) : InsertPara
    : Next K
```

The virus uses a number of variables in the macro. The names of the variables, subroutines, functions and labels, all consist of ten to nineteen upper case letters from A to V, each of which changes randomly with every replication. This complicates analysis of the macro.

Polymorphic Engine and Transient Macro

The AutoClose macro maintains eighteen global variables, of which there are two string arrays of 31 and 200 characters respectively. As the variable names are not fixed, I will refer to the two arrays as array1$ and array2$.

These arrays are used as two archival tables and assigned string values in a subroutine. Array1$ preserves the names of identifiers for all the variables, subroutines, functions and labels in the current instance of the macro. Therefore, the virus can readily employ *Word's* 'find-and-replace-all' function to replace each identifier's name with another randomly generated name.

The first 180 elements of array2$ are assigned string values; each of them is actually an encrypted text. In the same way as encrypted DOS viruses, the virus macro preserves the encryption key for the purpose of self-decryption whilst executing. The encryption key here is an integer, randomly selected, between four and thirteen. The encryption itself is quite simple: each character is encoded into one whose ASCII code equals that of the original character plus the encryption key. In one instance,

```
array2$(87)="If MacroName$(i, j) =
    @@AutoClose@@ Then CheckInstalled = 1"
```

is encrypted into

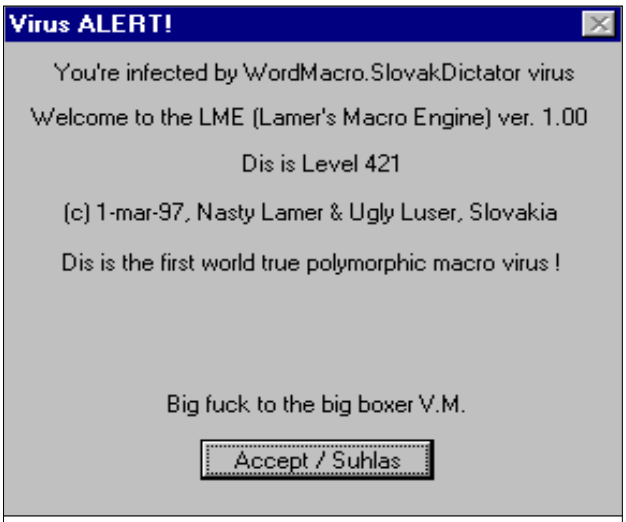
```
array2$(87)="Pm'ThjyvUhtl+/p3'q0'D'
    GGH|{vJsvzlGG'[olu'JoljrPuz{hsslk'D'8"
```

with the encryption key seven. In fact, if all of the string values of array2$ are decrypted and put together with '@@' being replaced by '"', they will form another macro. This is a hidden and transient macro which contains the payload and resides briefly in the global template.

In Operation

Like many other macro viruses, SlovakDictator attempts to protect its operation and remain unnoticed. To this end, it prevents the user from interrupting its operation by pressing the ESC key, so that it can execute completely. It also suppresses screen updates while running and disables built-in auto macros. Next, the viral AutoClose checks the version of *Word*. If it is not version 7, the macro simply terminates. Otherwise, it grabs the encryption key and decrypts the string values stored in array2$.

The virus then silently makes a macro in the Normal template by inserting the decrypted string values of array2$ into the new macro and substituting '"' for '@@'. Also, the virus uses the WordBasic 'find-and-replace-all' function to replace each identifier's name with a new name. This new name consists of randomly-chosen letters between A and V with a random length between ten and nineteen characters.



The dialog box displayed by SlovakDictator.

The assignment statements for array1$ and array2$ are appended to a subroutine in the macro. Then the hidden macro is invoked. This transient macro contains a non-malicious payload: a dialog box will pop up on the 4th and 11th of each month. This is the only obvious manifestation of SlovakDictator's presence most victims would notice.

The main task of the transient macro is to replicate the virus. Similar to an 'Are you there?' call in a DOS virus, the macro checks whether the Normal template and the active document are already infected. This infection check is made by comparing 'AutoClose' with the name of each macro associated with both the Normal template and the active document.

If the global environment is not infected, the transient macro creates an AutoClose macro in the Normal template and copies the viral code into it. The current string values of array1$ are inserted, a random number between four and thirteen is generated as the encryption key, and the key is copied along with the newly-encrypted string values of array2$ into the macro. Then the transient macro exits.

Should the Normal template be infected and not the active document, a similar process results in the document's infection. To ensure that the macro in the newly-infected document will activate, the document is saved as a template file-type.

Immediately after the transient macro in the Normal template has finished, the calling routine deletes the macro. Finally, the virus restores the previously-disabled functions, including screen updating, auto macro invoking and macro interrupting. Thus, a *Word* user may not notice anything abnormal following SlovakDictator's execution.

Summary

SlovakDictator represents a new breed of macro viruses, implementing techniques thus far common in their DOS counterparts; namely polymorphism, encryption and stealth. As far as typical *Word* users are concerned, polymorphic macro viruses are of little interest and pose no additional threat. However, SlovakDictator and its kind certainly set new challenges for anti-virus workers. Macro viruses are clearly becoming more complicated, though are currently less sophisticated than their DOS counterparts.

SlovakDictator

Aliases: Slovdic, Slow.

Infects: *Word 7* documents and templates.

Self-recognition: Checks documents and templates for the presence of an AutoClose macro.

Hex Pattern in Files:
```
240C 673B 8005 0664 67D7 0073
0100 0C6A
```
This may be unreliable, but a longer constant pattern is not available.

Trigger: Displays a dialog box when infecting on the 4th and 11th of any month.

VB'97 – *The* Anti-virus Conference

The *Virus Bulletin* international conference is now in its seventh year. This annual conference is recognized as the world's leading event addressing the computer virus threat. *VB'97* will run as two parallel tracks, one corporate and one technical, and is being held at The Fairmont Hotel, San Francisco, USA on 2/3 October 1997. Over three hundred delegates are expected to attend the presentations led by a panel of internationally renowned virus experts. The *VB'97* exhibition, featuring the world's leading anti-virus vendors, will run alongside the conference programme. Exhibitors include *McAfee*, *Dr Solomon's Software*, *Sophos*, *Command Software*, *NCSA*, *IBM*, *Integralis*, *Trend*, *DataFellows*, and *Elsevier Science*.

The conference provides delegates with good opportunities to meet the industry experts and speakers. The social programme includes a welcome drinks reception and the spectacular black tie Gala Dinner. An interesting partners' program is available for delegates' partners and/or family.

A brochure for *VB'97* was included with this copy of *Virus Bulletin*, but if a colleague has already used it, or if you would like further information on *VB'97*, please contact Alie Hothersall at *VB* (email alie@virusbtn.com) or visit the *Virus Bulletin* web site; http://www.virusbtn.com .

FEATURE 3

Do you Know the Way to VBA?

Nick FitzGerald

Over the last few months, a debate has raged behind the scenes in the anti-virus industry. At issue have been the merits of anti-virus researchers 'up-converting' *Word* and *Excel* macro viruses from earlier *Microsoft Office* files to *Office 97* formats.

At times the tenor of the discussion became more strident than some were comfortable with, and the language used was more like that usually reserved for describing virus writers, rather than anti-virus colleagues. What caused this?

Follow the VBA Road…

The heart of the issue seems simple enough: some *Word* macro viruses that have, to date, only been seen in files with formats native to *Word 6/7* (the versions that came in *Microsoft's Office 4.x* and *Office 95* respectively), successfully replicate when opened in *Word 8* (from *Office 97*) *and* remain infective. The same is true of several *Excel* viruses, which have only been seen in files from the earlier versions but will spread when opened in *Excel 97*.

But didn't *Microsoft* put virus protection into its *Office 97* products to prevent this? Yes, *some protection*, but it is easy to either turn off those options, or just click the 'Enable Macros' button when *Word* or *Excel* detects it is opening a document or spreadsheet containing macros. Not all *Word* documents or *Excel* spreadsheets containing macros contain viruses.

In some workplaces, users have become accustomed to clicking away the 'Warning' dialog with the 'Enable macros' button when opening 'internal' standard company documents. The risk here is they will become blasé about it for 'other' documents. Worse, for those to whom the incessant clicking quickly becomes an irritant, *Microsoft* thoughtfully placed a check-box for turning this option off right there in the dialog box such users come to loathe. Uncheck 'Always ask before opening documents with macros or customizations' and the frustration is gone for good. For simplicity's sake, the rest of this article will refer to *Word* and documents, but the same principles apply to *Excel* and its spreadsheets.

There have already been some *Word 8* up-converts from 'ordinary' user sites that appear to have originated through processes similar to the above.

There is a second, though now much less common, route for macros to be up-converted. Due to the inclusion of the macro virus protection features very late in the development cycle, and the wide distribution of pre-release and beta copies that do not have any of the protection features, there are still copies of *Office 97* around without these features. In fact, probably the first *Word 8* macro virus was a Wazzu.A up-convert from *Microsoft's* WWW site early in February this year, prepared with a beta version of *Word 8*.

The third up-convert route is manual transcription, where the source code of a macro virus from an earlier version of *Word* is typed or pasted into *Word 8*. Such up-converts are usually dismal failures without significantly restructuring the macro code (viral or not!) because of the changes necessitated with the move from the simpler WordBasic of earlier versions of *Word* to Visual Basic for Applications (VBA5) used in the *Office 97* suite. This sort of exercise is more akin to porting a program from one operating system (or environment) to another.

Should we be Worried?

OK – so 'old' *Word* macro viruses can be upgraded to new, whizzy VBA5 macro viruses. Is there a problem here?

Well, in a word, yes. Just because JammyScan detects WM/Wazzu.A does not (necessarily) mean it will detect W97M/Wazzu.A. 'Why not?' several of you gasp, surprised. The answer is easy – because VBA5 is a different macro language from WordBasic, even macro code that has exact functional equivalents in both is represented differently in the byte-stream that stores the macro within a *Word* document. A reasonable analogy is with compiled high-level languages. Syntactically correct C source code, for example, will (generally) result in four different binary representations when run through four different compilers.

You may now be feeling relieved, safe in the knowledge that you have the very latest version of JammyScan, and it is an acknowledged macro virus master. Comforted by this belief, many, many users of anti-virus software are unaware of a fundamental assumption almost hidden here. If you use *Word 8*, or are likely to move to it soon, ask yourself this: Have the developers of my chosen anti-virus products checked their detection rate against all existing *Word 6/7* macro viruses when infected files are opened in *Word 8*?

Better yet, ask the developers of your chosen anti-virus products or their vendors.

The truth is that some developers have not done this, or have only done it for a small subset of extant macro viruses. However, probably even more surprising is that some developers have avowed that they will not do this!

I believe that this is a short-sighted and indefensible position. Macro viruses exist. Their prevalence is increasing rapidly, as are the number of known virus families and variants. Many people are already using, or moving to,

Office 97. Most current macro viruses are for *Office 4/95* applications (at least by family and variant count). Some existing *Office 4/95* macro viruses successfully replicate in the new *Office 97* environment and some remain infective.

To me, that spells 'clear and present threat'. Security-conscious developers should be considering such things when designing their products. They can be sure security-conscious consumers will be. For me, these are compelling reasons for anti-virus developers to go into the lab, run *Word 8* and start opening their macro virus samples from earlier *Word* versions, and to add that step to their tests and analysis of new macro viruses.

Once the current backlog is cleared, this should not add unduly to the time it takes to analyse new *Word 6/7* viruses. Some developers proudly claim that it only takes five to ten minutes to add most new macro viruses to their products. Assuming an average of twelve new macro viruses per work day, adding an up-convert test would add two hours of analysis time per day. In practice it would not, as many (even most) up-converts will not remain viable and thus will require less analysis time.

Et tu, Brute?

Why are some developers not doing this? The answer to this is where things started getting contentious. Some developers not working on up-converted macro viruses argue that opening 'old' *Word* macro viruses in *Word 8* creates new viruses. Anti-virus developers have faced the largely bogus, though commonly repeated, claims that the anti-virus industry created and then drummed up the virus threat to protect their jobs. Some developers feel strongly that any claim they have ever created a virus is so undesirable that they should take all possible steps to avoid having this label applied to themselves.

This is understandable, and on the face of it, surely a reasonable stance? It is reasonable – in fact, I encourage anti-virus developers not to develop and distribute new viruses. But to claim that up-converting macro viruses opens the industry to charges of being the virus writers is not a valid argument.

Notice, above, I said 'develop and distribute new viruses'. The virus writers the anti-virus industry is concerned about are the ones that distribute (even to one other person) the viruses they write and I am advocating that up-converts should be made by bona fide anti-virus researchers under controlled conditions in their test labs. I am not encouraging the distribution of such up-converts, even between these professional researchers. In fact, I disagree with such exchange of lab-generated viruses.

Let's look at another computer security field; system security specialists. They advise clients about known and future/possible security threats. They monitor installed systems to ensure that their recommendations have been properly implemented and that no new threats have arisen. Sometimes they 'watch' attempted attacks progressing to learn new tricks from the attackers. At their client's request, and with appropriate authorization from suitably senior management of the organizations concerned, they will attempt to hack a system from outside to test the security implementation and that internal procedures for when an attack is suspected or discovered, are followed.

Some may claim this as evidence that system security analysts create their own industry niche, but it would be a difficult argument to sustain. Exactly the same relationship holds between responsible anti-virus developers and claims that the anti-virus industry created and maintains itself. A few spotty-faced, ethically underdeveloped 'hacker underground' types might claim the industry is self-preserving, but their opinions are based on largely uninformed conjecture that touches on an unfortunate grain of truth that once applied to a very small number of early and/or insignificant anti-virus developers.

Arguing that making up-converts sets us on the slippery slope of facing such claims is a cop-out – like it or not we are already on that slope. Better to be known for acting impeccably and doing everything one could to ensure anti-virus consumers had the best possible protection against known, current risks than for denying that the anti-virus industry has a few skeletons in the closet and a few differently-pigmented sheep in the back pasture.

Purchasers of anti-virus software should get the best protection available. Today we know that some macro viruses can successfully up-convert to *Word 8*. More than being a theoretical possibility, however, some old *Word* viruses *have* been up-converted at real client sites. Some developers probably brand the users of machines where such real-world up-converts have happened as 'virus creators' who therefore deserve what they get: the old 'blame the victim' approach. A responsible attitude demands that product developers be responsive to the real threats faced by real computer users, or at least by those who use their products. The anti-virus community is aware of this threat and in this case, reducing the additional risk their consumers face can be done at minimal cost to the developers. It is not a question of if, but of when!

If your anti-virus developers expect you will be protected against *Word 8* viruses by the crude and easily circumvented 'protection' features *Microsoft* cobbled in at the last minute, they are being disingenuous. You may think I have been being rather rude about the built-in 'protection' features, but my opinions, as expressed here, reflect those of the product developers themselves.

When is a New Virus not a New Virus?

Those in the 'thou shall not up-convert' crowd still maintain they cannot face the moral scourge of being labelled as virus writers. So let's look more closely at the issue: does up-converting a *Word 6/7* macro virus to *Word 8* make you a virus creator?

Technically, it might. As already stated, it is the virus writers who distribute or release their viruses that we dislike. This does not make virus writing acceptable: it seems that few people who have ever created a virus can contain themselves from giving a sample to their best friend, their classmates, whoever – witness the number of so-called 'research only' viruses in virus collections. An anti-virus developer 'making' an up-converted macro virus so as to add detection and disinfection of it to their product, *prior* to the virus actually appearing in the wild, does not increase the total threat. However, adding such a sample to their virus collections would, as most developers share samples regularly, and sharing samples made purely for internal research purposes is not a good idea.

I have thus far avoided a technical issue, and depending on how you look at this one, the discussion in the last couple of paragraphs may be moot. Are W97M/Wazzu.A and WM/Wazzu.A really different viruses? Some of the experts say they are different, some not. Who is right?

There are good arguments for both sides. You could argue they are different viruses based on the fact that anti-virus programs need different definitions for each in their products. A reasonable counterclaim would seem to be that relationship between WM/Wazzu.A and W97M/Wazzu.A is similar to that between the EXE-infecting form of a multi-partite and its MBR or SYS-file form. Some such viruses require separate definitions in anti-virus products for each form, but most researchers have no trouble seeing both forms as one virus.

One suggestion here is to classify Wazzu.A as a multi-partite, or even a cross-platform, virus. If you think of it as cross-platform, you would consider the forms native to each platform as separate manifestations of the same virus. In the case of a cross-platform virus, it would seem likely for the code representations to differ on the different platforms.

To date, the debate over the issue of whether the *Word 6/7* and *Word 8* forms of, say, Wazzu.A are the same has revolved around the CARO taxonomy which makes them different. According to the CARO standard, if the binary representations of the non-variant parts of the two differ, they are different viruses. Unfortunately, this has little to do with the generally accepted definitions of what a computer virus is. These definitions focus on behaviour, not code representation. It seems perfectly plausible, in fact *likely*, that a cross-platform virus will have different representations for its different platforms but manifest the same behaviour on those platforms.

The CARO taxonomy is surely convenient and immensely practical. It means that painstaking, expensive behavioural analysis of *every* different virus need not be undertaken. To date, it probably has not mattered that the CARO taxonomy has ignored cross-platform issues because there have been very few, if any, cross-platform viruses until now. However, given that it ignores such issues, and that what is under discussion is a cross-platform virus, the fact that the CARO naming scheme labels the *Word 6/7* manifestation of this virus as WM/Wazzu.A and the *Word 8* manifestation as W97M/Wazzu.A does not justify or validate claims that they are different viruses.

If you accept that a macro virus that viably up-converts to *Word 8* is a different platform manifestation of the same virus as its *Word 6/7* relation, then there is no issue here for the moral stalwarts who argue that anti-virus researchers should never create new viruses. Laboratory-generated up-converts are thus much like the tens of thousands of polymorphic samples that most researchers generate to test their product's detection of that polymorphic engine.

To Test or not to Test?

That is a tricky question, and where the heated discussion mentioned at the beginning of this article began. It was suggested that the *NCSA* should include up-converts in its test-sets. For the meantime the *NCSA* has backed away from this idea, which is probably prudent.

What is *Virus Bulletin's* position on including up-converts in test-sets? For now I am against the idea of including self-made up-converts, but not so much on principle, but because of *VB's* testing procedures. It is unfair to test a virus detection product, publish results saying it missed X and then withhold X from the developer. If *VB*-made up-converts were included in a test, they should be sent to a developer whose product missed them, and I disagree with distributing a virus that I have made, regardless of the standing and reputation of the receiving party. Samples replicated from naturally occurring up-converts are a different story: recently some of these have been added to *VB's* In the Wild File test-set.

There is a good technical reason too. There are many different releases of *Word 8* in the world – all the different language versions plus the betas and some earlier pre-releases. There may have even been unannounced code revisions at some point in the short history of the product. For example, was the *Office 97* code released for sale in Australia and New Zealand before Christmas 1996 really identical to the other 'International English' releases in February 1997? The upshot of all this is that up-converts in your *Word 8* may not result in exactly the same macro code representations as those in others. It may be possible to work around such problems once all the possibilities are discovered, but this means that lab-generated up-converts will not necessarily match up-converts of the same virus that will eventually arise in the outside world. That said, most users of a given version of *Word* are unlikely to have tweaked the options known to result in different up-conversions, so lab-made up-converts can still be a useful indicator.

I look forward to the results of further research in the area of *Word* macro up-conversions. Once more is known about these specifics, we will be better placed to decide the role of up-converts in formal testing. For their customers' sakes, developers should be working with up-converts now.

PRODUCT REVIEW

SWEEP for Windows NT v2.97

Martyn Perry

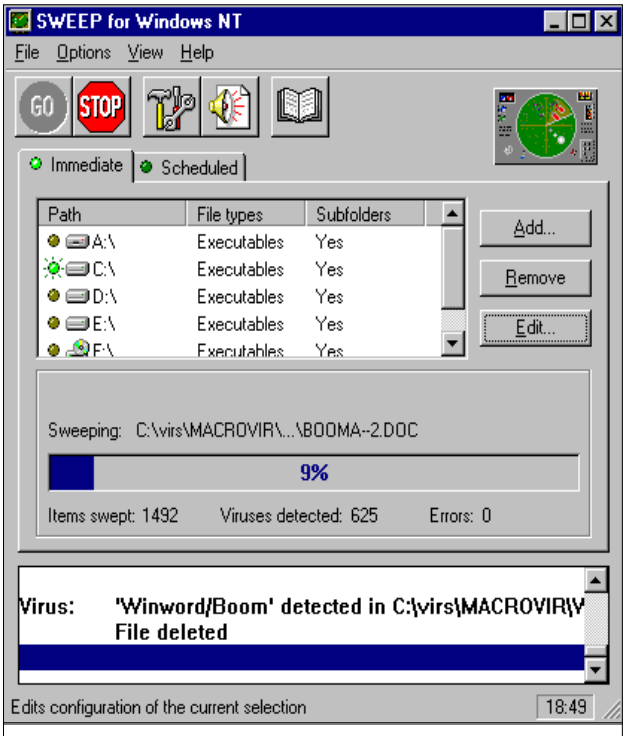
This month's product is from *Sophos*. *SWEEP* has been available for a number of years now and I was most interested to see how the product had migrated over to newer environments. The version under evaluation is *SWEEP for NT* v2.97 released on 1 May 1997. It claims to handle 10650 viruses.

The server licence can either be purchased separately or incorporated into a site licence including workstations.

Presentation and Installation

It makes a pleasant change to receive a full shipping set of the product, as this provides an opportunity to see the support material available.

The product submitted for review came with disk sets for *Windows NT*, *Novell NetWare* and *Sophos' InterCheck*. As far as documentation is concerned, *SWEEP* is shipped with user manuals for *Windows NT*, *Windows 95* and DOS clients. The packing is completed with quick reference cards for running the scanner in these environments, sample sheets of labels to show a PC has been scanned, and warning labels to indicate that a PC is infected. There is also a mouse mat.



SWEEP's main screen with an Immediate scan in progress.

Finally, there is the 'Data Security Reference Guide'. This book, besides acting as a brochure for the various services and products from *Sophos*, contains a very clear and comprehensive virus tutorial, which I thoroughly recommend to any newcomer to the world of computer viruses.

SWEEP for NT now comes as a three-diskette set, with two diskettes holding the GUI version and one the command-line scanner. Opinions vary over the need for command-line utilities in our increasingly GUI world, so whatever your view on the matter, *SWEEP for NT* gives you the choice.

On running the installation program, you are first presented with a choice of Local Installation/Upgrade or Central Installation/Upgrade. The Central Installation option makes a copy of the installation files on a server so they can be used for installation on remote servers and workstations.

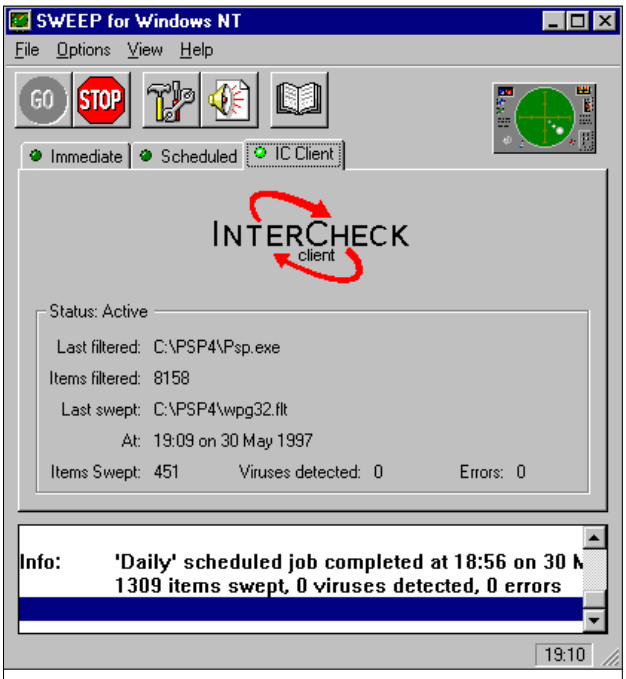
If you choose the Local Installation option, you may enable *InterCheck* support and scheduled scanning across the network separately. A local installation was used for this review. The folder 'C:\Program Files\Sophos SWEEP for NT' is the default installation location.

Once you have set these basic options, the setup program presents your configuration for checking, giving you the option of going back to change things. This screen, somewhat puzzlingly, implies that choosing to create short-cuts in the Start Menu and, significantly, allowing *SWEEP* to appear in the Add/Remove Programs control panel are configurable options – perhaps in the next version?

On clicking the final *OK* button in the Setup dialog, *SWEEP* is installed and the opportunity provided to run the scanner immediately.

Although not the configuration tested in this review, I looked at the central location installation procedures. Not surprisingly, somewhat different configuration options are offered. The settings now include auto-upgrade from a server, and prevent removal via the Add/Remove Programs control panel (the default is to allow removal). The default central installation folder is 'C:\Program Files\Sophos SWEEP for NT\NTInst\i386'. The options selected here are recorded and used as the defaults for any subsequent client installations run from this directory. Again, there is a confirmation screen listing your selections and allowing you to go back and change things before starting the installation. Subsequent client installation from this source was not tested.

Electing to continue, files were then copied to the server from diskettes labelled Installation, Library, *InterCheck*, and DOS. There is an option to skip the last two diskettes. This means pressing the skip button the requisite number of times for each file on each skipped disk.



With *InterCheck* activated, another tab appears on *SWEEP's* main window.

During the installation process, a warning message may be displayed: 'SWEEP for Windows NT setup program was running on the machine for more than 5 minutes. Do you want to exit?' This option allows an 'out' for autoinstall on clients, should they stall or fail for some reason. In a corporate environment you would now be ready to run SETUP.EXE from the server on each of your clients.

Scanning with SWEEP

SWEEP has three scan modes – Immediate, Scheduled and *InterCheck*. The latter is *Sophos'* real-time protection component and, taking a different approach from most on-access scanners, is covered in more detail a little later.

An immediate scan can be started and stopped from the console. If stopped then restarted, the scan returns to the beginning of the directory structure rather than resuming. Progress is displayed on a typical 'thermometer gauge'. Being someone who likes to know things are progressing, it may sound a little perverse to say that there is a useful option to disable this progress indicator. However, on large network drives, *SWEEP* may take several minutes making its initial count of the items to scan before beginning the scan itself.

By default, this version scans files with extensions of ADD, BID, COM, DLL, DMD, DOC, DOT, DRV, EXE, FLT, I13, IFS, MOD, OV?, SCR, SYS, TSD, VSD, VXD, and XL?.

You have the choice of either Quick or Full modes of operation, where Full mode examines the complete contents of a file while Quick mode just checks the areas where viruses are most likely to be found. To minimize the impact of scanning a server, you can set *SWEEP* to run as a low-priority background task. Further options include the ability to check executables compressed with PKLite, LZEXE and Diet, and checking for Macintosh viruses.

SWEEP provides the following actions in the event that it detects a virus:

- disinfect boot sector
- disinfect documents
- rename to non-executable
- delete infected executables
- 'shred' (delete and overwrite) infected executables
- move to quarantine folder
- copy to quarantine folder

The last two options use the folder INFECTED, below the main *SWEEP* program folder.

Each scheduled scan is set up as a 'job'. Each job defines the scan mode and actions. Essentially, the scheduled scan has the same modes, actions and reports as the immediate scan. The only exception is that floppy drives cannot be included in a scheduled scan. However, CD-ROM drives can be, and this could provide a method of scanning a new CD overnight if necessary, although a CD-ROM containing 629 MB scanned in about four minutes in a 3-speed drive.

Multiple scheduled jobs can be configured and saved, but only one can run at a time. If a job is scheduled to start and another has not completed, the second one is queued and starts immediately the first finishes. As with immediate scans, scheduled jobs can also be set to run at a lower priority if required.

Defining a scheduled scan, you can choose to run it at start-up or at regular intervals. If the latter option is taken, you select the start time and date for the initial scan and the subsequent days to run it at that time.

Real-time Scanning with InterCheck

InterCheck is a separate component which provides on-access or real-time scanning. *InterCheck* helps reduce the workload on a server and network when virus-checking files. This is done by avoiding the need to check files every time they are accessed, and is achieved by passing them through an 'authorization' process. The first time a file or diskette is accessed, the scanner checks it, and if it is clean the file or diskette is authorized (a checksum is made and stored). On subsequent accesses, checksums are compared and a scan initiated only if the object has changed.

In day-to-day operation, a file or diskette is checked to see if it has been authorized. If it has, then operations proceed as normal. However, if a file or diskette is new and has not been authorized, *SWEEP* is called to scan the object. If it scans 'clean' it is checksummed and its value added to the authorized list; otherwise, an infection warning is issued. This has the effect of creating an overhead when a product is newly installed or updated. However, once the task is

completed, the scanner is only used when new files or diskettes are accessed. This reduces the on-access scanning overhead. It also traps files that suddenly 'appear' on a network, be they Internet downloads, decoded email attachments or those neat little utilities a visiting consultant feels you really must have.

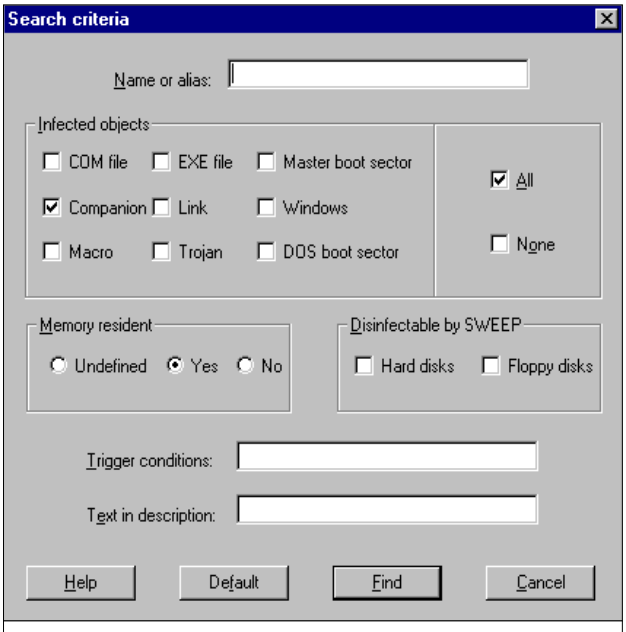
InterCheck's operation is controlled by options configured by a workstation's administrator. These determine how *InterCheck* operates at start-up as well as during runtime. Of special interest should be the setting that detects *Windows* program files and OLE files by their contents, rather than just depending on their file extensions.

Administration and Extras

SWEEP for Windows NT is 'NT security aware', and recognizes members of the Administrator group, letting them modify its settings. Non-administrator users can only start and stop *SWEEP* scans.

The Virus Library provides information on the viruses in the detection set. A search can be made for a name or alias. If this is not known, selections can be made as to the type of infection that is occurring. Having made a selection, a shortform display of the characteristics of the chosen virus is presented. More detailed information can be selected which includes advice on eradication. For those amongst us with less than 20:20 vision, there is a toggle switch to increase the text size.

A compressing backup utility, SB.EXE, is also provided. This is needed to decompress the VIRPATS.SB file which contains the virus descriptions. These are compressed with this utility, which makes use of *Sophos'* encryption technology (SPA) to ensure no tampering has occurred.



Want details on all memory-resident, companion viruses? This search-by-virus-feature option may help.

Reports and Activity Logs

The default location for report files is the REPORTS folder in the *SWEEP* installation directory. Report files have the extension .REP. Scheduled jobs generate reports with a filename matching the job name and reports from immediate scans have names matching the user who invoked them. Event logging can cover all jobs or just specific ones, with the notification level configurable for no messages, viruses, viruses and errors or all messages.

Alerts can be sent to specific desktops, or broadcast network-wide. They can also be sent using the SMTP Internet email protocol if your network supports this.

Command-line Support

Some companies appear to be shy about any command-line interface their products may have. Perhaps they feel that it has no place in our increasingly GUI environment. I feel there is still a place for such a facility, and I am particularly pleased to see a command-line interface version, replete with '-options' retained in this version. The options include those from other platform versions as well as some specific to *Windows NT*. These options include controlling the priority of the scanner and the writing of information the *Windows NT* event logs. Also, using the command-line interface allows remote servers to be scanned.

These various options can be stored in a text file and run with the @ option. For example:

```
NTSWEEP @SAMPLE.TXT
```

where SAMPLE.TXT contains the scan options. This allows different configurations to be defined without having to do screeds of retyping.

Updates

SWEEP is updated monthly. Updates are installed by running the installation program on the supplied diskettes and selecting the 'Upgrade existing installation' option. The installer will detect older versions of *SWEEP*, and if one is found prompt whether to retain the existing settings. If the *InterCheck* option is selected, the scanner will be run once the upgrade installation completes.

Urgent, interim virus definition updates, between the monthly upgrades, are distributed in plain ASCII files. These so-called IDE files, if stored in the *SWEEP* program directory, are automatically loaded *when the service starts*. This means you have to stop and restart the service component to activate such updates.

Detection Rates

The scanner was checked using the usual test sets – In the Wild Boot, In the Wild File, Standard, and Polymorphic – and against the new Macro test-set. See the product summary box for more details.

The tests were conducted using the program's defaults for file extensions and for the Quick scan mode. *SWEEP* was set to delete infected files. The residual files were then used to determine the detection rate. The results were excellent.

Using the default Quick scan mode, *SWEEP* failed to detect only the two samples of Positron from the Standard test-set. However when the test was repeated with Full scan, 100% success was achieved: this is because Positron inserts its code in an area not covered by the Quick scan option. This suggests running a Full scan initially and a Quick scan on subsequent, scheduled runs.

Real-time Scanning Overhead

A significant issue for on-access scanners is their impact on overall workstation performance – few people would tolerate a virus detector that ground their computers into the dust, even if it was a top performer, in terms of detection. To measure scanner overhead, 200 clean COM and EXE files, comprising 21.2 MB, were copied from one folder to another using XCOPY. The default NT scheduling setting of 'Maximum Boost for Foreground Application' was used for consistency.

Because of the different processes which occur within the server, these tests were run ten times at each setting and an average taken. The tests were:

- Neither *SWEEP* nor *InterCheck* loaded. This establishes the baseline time for the copying process.
- *SWEEP* and *InterCheck* loaded but inactive. This measures what impact the applications have in a quiescent state.
- *SWEEP* loaded, *InterCheck* active with Purge List set to No. This tests the impact of *InterCheck* in a normal workstation configuration.
- *SWEEP* loaded, *InterCheck* active with Purge List set to Yes. This tests the impact of *InterCheck* when recalculating the checksum each time the copy is run.
- *SWEEP* loaded, *InterCheck* active with Purge List set to No and Scan running. This tests the impact of *InterCheck* without recalculating the checksum each time the copy is run.
- *SWEEP* loaded with *InterCheck* active but with Purge List set to Yes and Scan running. This tests the impact of *InterCheck* recalculating the checksum each time the copy is run.
- *SWEEP* unloaded. This is run after the other tests to check how well the system returns to its former state.

The detailed results are presented in the product summary box.We can see the impact of *InterCheck* running in its usual mode (no purging of checksums) and how it can help to keep real-time overhead down. Also shown is the effect of having to calculate the checksums and the performance hit that can be expected the first time *InterCheck* is run after upgrading the scanner.

Summary

The scanning results were excellent, maintaining the tradition of consistently high detection rates over a number of years. The use of *InterCheck* to help ease the load on day-to-day access is a good approach which is not compromised by new files appearing on the network.

The one area of concern is that there is still no facility for managing a domain of multiple servers and workstations, although remote servers can be scanned from both the GUI and command-line scanners. The Auto-update feature partly addresses this, with its ability to update all servers and workstations attached to the distribution server, but this does not allow for flow-on upgrades from servers that are themselves auto-upgraded. With the developers having achieved so much with the product – excellent scan results combined with low overhead – it can only be a matter of time before this is addressed.

SWEEP for Windows NT v2.97

Detection Results

Test-set[1]	Viruses Detected	Score
In the Wild File	527/527	100.0%
In the Wild Boot	90/90	100.0%
Standard	772/774	99.7%
Polymorphic	13000/13000	100.0%
Macro	710/710	100.0%

Overhead of On-access Scanning:

The tests show the time (in seconds) taken to copy 200 COM and EXE files (21.2MB). Each test was repeated ten times, and an average taken.

	Time	Overhead
Baseline	14.5	–
SWEEP & *IC* loaded, inactive	15.0	3.5%
IC active, no purge	18.9	30.3%
IC active, purge	21.0	45.0%
SWEEP & *IC* active, no purge	40.0	175.8%
SWEEP & *IC* active, purge	58.6	304.4%
SWEEP & *IC* unloaded	15.1	4.3%

Technical Details

Product: *SWEEP for Windows NT, v2.97.*

Developer: *Sophos Plc*, The Pentagon, Abingdon, OX14 3YP, England. Tel +44 1235 559933, fax +44 1235 559935, email sales@sophos.com, WWW http://www.sophos.com/.

Price: Single *NT* workstation £195. *NT Server* with up to 24 clients £495, *NT Server* with 25 – 199 clients £895. For site licensing and bulk purchases, contact *Sophos'* sales team.

Hardware Used: *Compaq Prolinea 590* with 80 MB Ram, 2 GB Disk, *NT 4.0* with service pack 1.

[1]**Test-sets:** For a complete listing of all the viruses used, see *VB*, July 1997, p.16.

SUBSCRIPTION RATES

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

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

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire, OX14 3YP, England

Tel 01235 555139, International Tel +44 1235 555139
Fax 01235 531889, International Fax +44 1235 531889
Email: editorial@virusbtn.com
World Wide Web: http://www.virusbtn.com/

US subscriptions only:

June Jordan, *Virus Bulletin*, 590 Danbury Road, Ridgefield, CT 06877, USA

Tel +1 203 431 8720, fax +1 203 431 8165

END NOTES AND NEWS

***CompSec 97* will be held in London** from 5 – 7 November 1997. The conference aims to help highlight the risk to IT systems, assess security shortcomings, and protect against fraud, disaster, and negligence. Information is available from Amy Richardson at *Elsevier Science*; Tel +44 1865 843643, fax +44 1865 843958, or email a.richardson@elsevier.co.uk.

The **24th Annual Computer Security Conference and Exhibition** will be held in Washington DC from 17–19 November 1997. This event features over 120 sessions covering such topics as Network Security, Encryption, and Product Issues. Information can be found on the *CSI's* Web site; http://www.gocsi.com/.

The *Roderick Manhattan Group* has announced **a major push into the UK consumer and small office/home office market** by *Computer Associates* with its *Cheyenne AntiVirus v4.0 for Windows 95*. *RMG* is distributing *Cheyenne AntiVirus* through such well-known resellers and retailers as Byte, Dixons, Electronic Boutique, PC World, Staples, Software Warehouse and Watford Electronics.

Australian anti-virus developer ***Cybec* has formed an alliance with UK-based *Integralis Ltd***. The OEM deal will see *Cybec* incorporate its *VET* virus detection technology with the network connectivity specialist's mail unpacking and scanning technology, *MIMEsweeper*. *NT*-based *VET Mail* is available now, claiming to intercept, decode and virus-scan all incoming, outgoing and internal SMTP mail. More information is available on the WWW at http://www.cybec.com.au/.

Would your organization survive a crisis? Flood, fire, earthquake, the year 2000? The *9th Annual Business Continuity in Action Conference* and associated exhibition, to be held at the Bournemouth International Centre on 5/6 November, focuses on disaster and crisis planning. Call *Survive!* on +44 181 8746266 for bookings.

The **7th Annual Virus Bulletin Conference**, *VB'97* will be held on 2/3 October 1997. Full details are given on p.16 of this issue.

The 20th National Information Systems Security Conference is being held from 7 – 10 October 1997 at the Baltimore Convention Center, Maryland, USA. Covering such critical IT issues as secure electronic commerce, Internet security, and virus detection the conference attracts more than 2000 participants. For more information, visit the conference's Web site at http://csrc.nist.gov/nissc/, Tel +1 410 8500272, or email NISSConference@dockmaster.ncsc.mil.

The infamous hacker **Kevin Mitnick was recently sentenced** to 22 months for parole violations and cell-phone fraud. The charges relate to his arrest in North Carolina in February 1995. He is still awaiting trial on a further 25 counts of computer and cell-phone fraud, and of damaging computer data.

The **Secure Computing Tactical Conference**, will be held 7 – 9 October 1997 in the Connaught Rooms, London. To quote the organizers: 'This conference offers the chance to cut to the heart of the solutions – no high-blown theory – no waffle – real interactive discussions and in-depth case studies.' One of the four conference tracks is devoted to anti-virus strategies. Contact Norman Bullen; Tel +44 1792 324000, email nbullen@westcoast.com or visit the magazine's Web site; http://www.westcoast.com/.

McAfee, *Security Dynamics Technologies*, *RSA Data Security* and *VeriSign* have announced SecureONE, **'the first complete network security environment'**. This initiative comprises a set of alliances and cross-licensing agreements to ensure that the partners' products work well together and the development of a set of APIs to be delivered in RSA's toolkits. More details are available from the partners' Web sites, for example; http://www.rsa.com/.