March 1991 ISSN 0956-9979

VIRUS BULLETIN

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

Editor: **Edward Wilding**

Technical Editor: **Fridrik Skulason**, University of Iceland

CONTENTS

VIRUS BULLETIN **©1991 Virus Bulletin Ltd, 21 The Quadrant, Abingdon Science Park, Oxon, OX14 3YS, England. Tel (+44) 235 555139. /90/$0.00+2.50**

EDITORIAL

The Fundamental Things Apply...

If patriotism is the last refuge of a scoundrel then fundamental principles are the last refuge of the editor. Providing accurate, reliable information and advice is a burden and a responsibility, particularly so if that information appears in a journal such as *VB* which arrogantly defines itself as 'authoritative'. To commend the use of products, methods and procedures which are not attested, and which could actually exacerbate the problem, would be foolhardy in the extreme.

Dr. Harold Highland, commenting on the recent proliferation of self-confessed virus 'experts' and exotic anti-virus gadgetry, (*Computers & Security*, February 1991) quotes Mark Twain:

''Be thankful for the fools for without them we could not exist.''

It is a singularly apt quotation to describe the attitude of those software developers who take advantage of public ignorance in the pursuit of a 'quick buck'. A large number of anti-virus software developers subscribe to *VB*, so the following discussion may help their marketing departments and PR men avoid potential battles with the *Advertising Standards Authority* or equivalent advertising 'watchdogs' worldwide.

There follows a brief discourse on the sort of hyperbole and potentially misleading statements which should set the alarm bells ringing. As a brief aside, prospective purchasers should summarily dismiss any promotional literature which incorporates selective quotations from *VB* product reviews, or other evaluations which infer unqualified support for their product. At least two product manufacturers have employed this '*VB* endorsed' tactic by the use of highly selective quotations. A sound academic approach is appropriate here - always refer to original sources!

A cursory examination of the most common claims often shows them to be illusory - features cited as advantageous often conflict with fundamental security practices while many statements, when analysed, are meaningless.

Consider the following statements:

Detects all Known Viruses!

Prospective purchasers should cast a jaundiced eye over any virus-specific product which 'detects all known viruses'. Known to whom and at what time? *Presumably* the virus is known to the person who wrote it *before* the anti-virus product developer. If not, they are obviously in league or one and the same person. The statement is obviously fatuous - at any one time the research community is oblivious to the development and circulation of numerous computer viruses.

Detects all Unknown Viruses!

Many packages proceed with a highly contentious statement to the effect that the software will 'detect all unknown viruses'. How do the developers know this? It is by no means impossible for software to detect unknown viruses with a high degree of assurance. One UK package has even gained certification to this effect. However, any software claiming this capability will, by necessity, employ a secure and extremely well-implemented integrity checking system which must be configured properly and maintained carefully. Given the relative complexity of this approach, awkward questions are more than warranted in the face of such claims.

Detects Viruses in Memory!

Certain software is said to detect viruses in memory. Again, this is by no means impossible (although it is becoming increasingly impracticable), but without access to the viruses themselves, the user will be unable to test this claim. The unsuspecting user is entirely at the mercy of the programmer's judgement, skill and quality-assurance procedures. To avoid undertaking a lengthy vetting process (let alone the potential embarrassment of failing to detect a resident virus), it is best to invest in a clean write-protected system disk!

Disinfects all Viruses!

These products boast that they can 'disinfect' files. Some disinfection programs overwrite infected files which is a proven and effective method of destroying virus code. Other programs adopt 'surgical' practices and attempt to *remove* the virus and restore the infected file. Positive erasure (multiple overwriting) is a sound practice in computer security, other tactics are questionable. If a package uses 'surgical' disinfection, the infected file must be returned to its *exact* state prior to its becoming infected. Overwriting viruses cannot be disinfected in this way. Again, the user is at the mercy of the programmer and will have no way of proving these claims.

Ambiguous and misleading statements traditionally emanate from marketing departments and it is, perhaps churlish to decry even their more extreme assertions as scandalous.

However, anti-virus software developers are in the privileged position of being able to issue statements with virtually no possibility of their claims ever being put to the test. In this respect they are unique within the software industry. The user has little recourse to independent advice and cannot avail himself of the virus code with which to test their products.

Only by preserving a healthy scepticism and sticking to fundamental truths some of the more absurd assertions be invalidated before they have any chance to gain credibility.

Finally, not to our great surprise "the universal anti-virus PC security product" was received at the *VB* offices recently - maybe this mean that all our problems are over.

TECHNICAL NOTES

Viruses in Disguise

The use of *LZEXE, PKLITE, DIET* and other compression programs to hide viruses from scanners presents a continuing threat. Reports of infected programs which have been packed by these utilities and uploaded to Bulletin Board Systems have become more common in the past few months.

Some anti-virus programs are able to unpack and scan compressed files, but if they are not able to do so, they should at least *identify* the file as "packed", and alert the user to the fact that it cannot be scanned for viruses.

One method used to disguise the MIX-2 virus was particularly devious. The attacker took a pornographic *text* file, created a **self-extracting archive** containing it and infected it with the virus. The resulting file was then uploaded to a BBS. When the program was run, it would only seem to create a *text file* named SISTERS.TXT, but the virus was installed in memory, ready to infect any program executed. The attacker assumed (correctly) that people would not suspect a 'text file'. Beware of self-extracting archives, pornographic or otherwise!

Detecting V2P6

The encryption method used by the V2P6 is by far the most variable of any known virus. Even the Whale virus with its 30 or more possible encryption methods can be detected with a set of identification strings, but this is not practical for V2P6, as the number of possible decryption routines is in the thousands. The virus is a determined attempt to expose the vulnerability of virus-specific scanning programs and has come close to rendering this approach to virus detection obsolescent. Detecting the virus is, nevertheless, still possible. A highly technical algorithmic detection method has been developed by *VB's* Technical Editor, Fridrik Skulason. The information is considered unsuitable for publication, but it will benefit bona fide anti-virus software developers who should contact *VB* in the UK or Skulason at the *University of Iceland.*

Write-Protection for Diskettes

Ray Glath of *RG Software Systems* in the United States has submitted the following report:

> "We regularly receive diskettes for analysis of suspected viruses. Often, the diskettes are write-protected, or shall we say, the user *thinks* they are write-protected because they have placed tape over the write-enable notch on the diskette.
>
> Transparent *Scotch Tape* over the notch does **not** write-protect a diskette. Neither does a piece of red tape write-protect the diskette.The reasons for these methods being ineffective are that floppy drives mostly use a technique of shining a light on the area where the notch appears and registering whether or not the light passes through. If the light is "seen", the diskette is write-enabled. Otherwise, it is write-protected.
>
> Similarly, there are drives that use a red Light Emitting Diode for this decision process. In this type of drive, a red coloured tab will write-*enable* the diskette.
>
> To write-protect a diskette, use the silver or black write-protect tabs which are usually supplied with new diskettes. Or purchase write-protect tabs from your favourite office supplies dealer. Don't take short cuts."

Glath's advice to avoid short-cuts is unassailable. On 5.25 inch floppy disks the application of the write-protect tab means that nothing can be written to that disk. On 3.5 inch disks the appearance of an open window on the sliding shutter indicates that the disk is write-protected. Write-protection of diskettes is a hardware function and, properly implemented, makes software manipulation impossible.

However, there have been conflicting reports regarding the (in)effectiveness of silver (or black) write-protect tabs on 5.35 inch floppy disks. Older drives use a mirror under the floppy disk notch to reflect light back to the photo-sensitive element next to the light source. Using a silver (or shiny black) write-protect tab can reflect light and make the drive believe that the disk is **not** write-protected. **No such associated problems have been reported with <u>matt black</u> write-protect tabs.**

The best rule, if in doubt, is to attempt to copy a file onto a disk write-protected with a tab (silver or black) of your choosing. Take heed of Ray Glath's warning, *Scotch Tape* and other 'home-brew' remedies are not recommended!

Revolutionary Techniques?

There is the ever-present danger of being misled by the announcement of 'revolutionary' but fundamentally flawed new techniques. One such method was recently announced which was based on an analysis of INT 21H calls, the register values when the function is performed and the approximate distance between the INT 21H calls in bytes.

At a recent conference, the developer of the technique described how he had created numerous variants of the Jerusalem virus, none of which were detected by existing scanners, but which *were* consistently detected by his method.

Apart from the fact that not all viruses use INT 21H calls, it must be noted that by modifying these viruses, for example by swapping entire blocks of code, one could easily produce a variant which would not be detected by this method, but which would *continue* to be detected by the overwhelming majority of pattern scanning programs.

As ever, the search for an all-encompassing solution goes on.

THE *VIRUS BULLETIN* CONFERENCE

Hotel de France, St. Helier, Jersey
12th-13th September 1991

CONFERENCE PROGRAMME

Thursday 12th September

8.15 - 9.15 Registration

9.15 - 9.30 **Opening Address**, **Edward Wilding**, Editor, *Virus Bulletin*, UK.

9.30 - 10.20 **Dr. Jan Hruska**, *Sophos*, UK.

Introduction to MS/DOS Viruses: Virus types, virus structure, payloads, hiding mechanisms, viruses on networks.

10.20 - 11.00 **Vesselin Bontchev**, *Bulgarian Academy of Sciences.*

The Bulgarian and Soviet Virus Factories: The sabotage mentality, methods by which East Bloc virus writers distribute computer viruses to the West, the availability of virus source code, the impact on the national software industry.

11.00 - 11.30 Coffee

11.30 - 12.10 **Ross Greenberg**, *Software Concepts Design*, USA.

MS-DOS Anti-Virus Tools and Techniques: Specific defences against viruses - scanning programs, TSR monitors, static analysis of malicious code, disinfection and inoculation routines.

12.10 - 12.50 **Yisrael Radai**, *Hebrew University of Jerusalem*, Israel.

Integrity Checking Methods: Checksumming techniques for anti-viral purposes - cryptographic checksums versus CRC programs.

12.50 - 2.10 Lunch

2.10 - 2.50 **Jim Bates**, *Computer Crime Unit*, UK.

Disassembly, Forensics and Recovery: Tools and techniques to analyse the exact effects of virus code under test conditions, 'armoured code' to prevent disassembly, the hazards of analysis, disk utilities and post-attack recovery.

2.50 - 3.30 **Steve White**, *IBM T. J. Watson Research Center*, USA.

IBM's Response: IBM's anti-virus strategy, internal research and development efforts, worldwide monitoring of virus attacks and the role of IBM's High Integrity Research Laboratory.

3.30 - 4.00 Tea

4.00 - 4.30 **Dr. Simon Oxley**, *Reuters*, UK.

A Corporate Strategy: A worldwide education, training and awareness programme to minimise the virus problem - the tactics to prevent, isolate and recover from virus attacks.

4.30 - 5.00 **Mike Perryman**, *Manufacturers Hanover Trust*, UK.

Software Control: Secure global distribution of anti-virus software from an office responsible for software security. Guidelines, controls and the software quality assurance process.

5.00 - 5.45 **Fridrik Skulason**, Technical Editor, *Virus Bulletin*, *University of Iceland*.

Future MS-DOS Viruses: An assessment of subversive programming techniques likely to be found in future generations of computer viruses. Evasion and damage maximization techniques and their implication for detection and recovery.

Thursday Evening

7.30 - 8.00 Drinks reception, *Hotel de France*.

8.00 Gala Dinner, *Hotel de France*. Speaker: **Sqn Ldr Martin Smith MBE**, *Royal Air Force*.

Friday 13th September

9.30 - 10.10 **John Norstad,** *North Western University*, Illinois, USA.

The Macintosh Virus Threat: A survey of Macintosh viruses and anti-virus tools. Examination of probable future developments in virus programming and generic defences.

10.10 - 10.50 **Ken van Wyk**, *CERT*, USA.

CERT, The Computer Emergency Response Team: CERT was established in the wake of Robert Morris's infamous worm program which crippled the U.S. Internet network in 1988. The lessons learned from this incident are explained and the organisation, structure and communications of a computer 'SWAT team' are outlined.

10.50 - 11.20 Coffee

11.20 - 12.00 **David Ferbrache**, *Heriot Watt University*, UK.

Unix: Trust and Mistrust: The Unix environment is used to demonstrate the weakness of discretionary access controls (DAC) and the inadequacy of 'Orange Book' mandatory access controls (MAC) to prevent virus and worm propagation. Data integrity and availability are shown to be low priorities under ITSEC criteria while formal software development and control is proposed as the key to Unix security.

12.00 - 12.40 **Professor Eugene Spafford**, *Purdue University*, Indiana, USA.

Securing Unix: It is commonly believed that Unix systems are inherently insecure. This is largely because of the way the systems are configured and administered rather than intrinsic shortcomings in the systems themselves. An examination of standard Unix features which, when implemented correctly, will diminish the threat from malicious software and human intrusion.

12.40 - 2.00 Lunch

2.00 - 2.40 **Kent Anderson/Steve Rowley**, *European Security Programme Office, Digital UK* .

Protecting Distributed Systems - The Digital Approach: Digital tools, procedures and guidelines to prevent virus and worm attacks on distributed systems with particular reference to VAX/VMS, Unix and the management of PCs.

2.40 - 3.20 **Martin Samociuk**, *Network Security Management*, UK.

The Enemy Within: An examination of blackmail, extortion and espionage through logic bombs, Trojan horses and covert channels, including case studies of criminal programming and corrupt work practices. The methods to combat and react to these incidents are highlighted.

3.20 - 3.50 Tea

3.50 - 4.50 Panel Session

The speakers will answer questions from the floor.

4.50 Close of Conference

Full details are available from Petra Duffield, Conference Manager, *Virus Bulletin Conference*, 21 The Quadrant, Abingdon Science Park, Oxfordshire OX14 3YS. Tel 0235 531889, Tel International +44 235 531889, Fax 0235 559935, Fax International +44 235 559935.

INTEGRITY CHECKING

The Flawed Six-Byte Method

An integrity checking program can be run periodically in order to check the integrity of a system and its software. Under normal circumstances the program, upon finding no anomolies, simply displays a single “OK” indication to the user. However, under suspicious circumstances it will typically display a (configurable) message similar to the following:

```
A virus infection is suspected on this computer -
please call PC-support (phone XXXX-XXXX)
```

Integrity checking programs determine whether system and program file lengths, attributes and composition have been altered in any way. If suspicious changes are detected the user is informed, so he may diagnose the problem. *V-ANALYST* (*VB*, October 1990) and *VACCINE* (*VB*, December 1990) are examples of well-implemented programs of this type.

As an example of a flawed approach to integrity checking which highlights the difficulties involved in developing a secure program of this nature, one simple method will be described. It was developed by *A. Padgett Peterson, Computer Security Plus, Suite 890, 200 East Robinson Street, Orlando, Florida 32801, USA*. This particular method is very limited and, as will be seen, the method is transparently **insecure**.

The idea behind the six-byte method is simple - a virus which wants to remain resident in memory must somehow allocate a block of memory to itself. The method involves obtaining the values of three words which contain information about memory allocation and comparing their values to the values they have when the system is known to be in a “clean” state.

The following words are used:

- The return from the INT 12H call which indicates the total amount of conventional memory (up to 640K).
- The amount of memory in use by DOS, device drivers and resident programs. This is determined by the CS value of the checking program.
- The amount of free memory, obtained by INT 21H, function 48H, with BX = FFFFH.

This method *will* detect viruses such as Brain and most other boot sector viruses which decrease the amount of memory, for example reducing it from 640K to 636K. It will also detect viruses which use conventional TSR methods to allocate memory, such as Jerusalem. Finally, Padgett Peterson’s method will detect viruses such as Datalock, which reduce the last memory block thus creating “free” space near the top.

Superficially, the method appears attractive as most common viruses belong to one of these three groups and will be detected. However, this technique is of no use against most Trojans or non-resident viruses, as the memory allocation will not have changed after they have executed.

The real ‘acid-test’ is whether this simple check is sufficient to detect resident viruses, as it is intended to. Specifically, several currently known memory-resident viruses will *evade* detection by the six-byte method.

The Stupid/Do-nothing virus is **not** detected because it does not allocate memory to itself. Instead it just copies itself to a fixed area in memory, starting at 9800:0000, a method which has several drawbacks - the virus will only work on computers with at least 640K memory, and some programs may overwrite this area, causing a system crash.

A virus may evade detection if it hides *above* the 640K memory boundary. The E.D.V. virus searches for free RAM at E800:0000 and moves downward, until it finds a free block.

The Number of the Beast virus will **not** be detected, as it occupies a 512-byte block within the area allocated to DOS. A similar method is used by the Micro-128 virus, and several other recent samples from Bulgaria, which hide the virus code in the upper half of the Interrupt Table.

Another major weakness of the six-byte method is its inability to distinguish between a normal TSR program and a virus which uses the standard TSR techniques to allocate memory for itself. The six-byte check can detect that some program has allocated a block of memory, but it remains for the user to determine whether this is a legitimate TSR or not. This problem disappears if the user only runs a fixed set of TSR programs and they are all loaded in AUTOEXEC.BAT. The method would be of little or no use to anyone using a large and variable set of TSR programs.

A well-implemented integrity checking program should include the capacity to take a “snapshot” of the Interrupt Table on a clean system in its standard configuration. If no changes are introduced by a virus, the “snapshots” are identical; conversely, any changes would indicate suspicious activity. This cabability will be sufficient to detect the Micro-128 virus as it overwrites 128 bytes of the Interrupt Table, and also the Stupid/Do-nothing virus, which hooks into INT 21H.

Further steps are necessary to detect all resident viruses. Full integrity checking should address not only memory allocation and the Interrupt Table - it should also verify the integrity of the operating system itself. For example, it is easy to detect the Number of the Beast virus by checking the number of the disk buffers in use by DOS, as it will be too low. Thus a comprehensive integrity checking program should also compute a checksum for the copy of DOS in memory, which would detect any attempts to patch it.

It is not easy to write a foolproof integrity checking program - the six-byte method is adequate testimony to this fact.

PROGRAM TACTICS

Fridrik Skulason

Developing a Virus Scanner

Most, if not all, anti-virus products include a virus scanner, the purpose of which is to check programs and boot sectors for infection by known viruses. A comprehensive *Virus Bulletin* review of eleven prominent virus scanning programs is currently underway and we hope to publish our findings in April. This article, the first in a series on software development, will serve as background reading to supplement the upcoming review. Throughout, I adopt the programmer's perspective and consider some of the design and implementation issues involved in writing this type of program.

The Hex Pattern

Almost all viruses known today can be detected with a simple identification pattern in hexadecimal, but a good scanner is more than just a search engine with a database of virus patterns. The programmer must consider several factors which affect the speed and accuracy of the scanner.

Selecting the Search Pattern

The selection of a reliable identification pattern is critically dependent on selecting a suitable area of the virus code from which to extract it. Unfortunately it is not possible to give any fixed set of rules on how to select a pattern. An immediate concern is the avoidance of false alarms. The selected pattern should minimise the prospect of false-positive indications whereby a virus is indicated in a clean file which happens to contain an identical code sequence to that of the virus. *VB* occasionally publishes amended patterns when informed of such false alarms. It is impossible to select a pattern from a virus and state categorically: “this is the best identification pattern for this particular virus”. The programmer should look for an unusual code fragment typically indicative of viral behaviour and unlikely to occur in legitimate software routines.

The length of the pattern is a key concern - a lengthy pattern reduces the danger of false-positives, but is more likely to be invalidated by an intentional modification to the virus.

Detecting Variants

Several approaches are possible when dealing with different variants of the same virus. An extreme approach is to select a highly specific identification pattern, which enables an *exact* identification of the variant, but it is also possible (and more practicable) to select a “family” pattern, which can be found in many related viruses.

A highly specific search pattern will not guarantee that a variant can be identified with absolute certainty, as changes might have been introduced *elsewhere* in the virus code. A scanner could, theoretically, contain a copy of every variant, and perform a bit-by-bit comparison with an infected file. However, this is an unrealistic proposition in light of the abundance of virus code in existence.

A more practical tactic is to identify the areas of every variant which contain *invariant* code - the code which is identical in all instances of a particular virus. A checksum would then be computed for the invariant area of each variant and the corresponding areas in any sample under investigation.

Why should a software developer expend such effort in trying to identify an offending virus with such exactitude? If the user deletes all infected files and replaces them from write-protected backup copies of master software, it really does not matter which variant, or even which virus infected his system. Most users will have little interest in such details; they will be anxious about the effects caused by a virus, but will remain quite indifferent as to whether the culprit was, for example, Jerusalem-A or Jerusalem-C.

While this is generally true, precise identification is of **vital** importance if the anti-virus program attempts to *disinfect* the file. The 4096 byte variants of AntiCAD/Plastique are good examples to explain the difficulties in developing disinfection software - three different variants of the same virus, but of the same length. Most identification patterns which detect one of the variants will detect them all. A disinfection program must be able to distinguish *between* them because they store a part of the original host program at entirely different locations internally. A failure to differentiate between the variants will result in a corruption of the files if any attempt is made to remove them using disinfection software.

Prospective developers of disinfection software, and anyone currently using such software, should be aware that no margin for error is permissible when adopting this strategy.

Precision Scanning or the ‘Brute Force’ Approach?

Normally any search pattern will always be found at the same location in every copy of the same variant. A scanner may store information on where the identification pattern is expected to be found - relative to the beginning of the infected program, the virus' entry point or the end of the infected program, depending on the structure of the virus.

Instead of laboriously searching throughout the infected file, the scanner can just check whether the identification pattern is present at its expected location. This is a faster but far less secure tactic than scanning the entire file for the identification pattern (the ‘brute force’ approach) or scanning code segments near the beginning and the end of the file. Reference to last month's *VB* product review of *Turbo-Antivirus* (particularly to the variation in scanning times between the ‘turbo’ and ‘non-turbo’ modes) will prove informative on this point.

Precision scanning substantially reduces the likelihood of detecting variants of the virus. If the virus code is disassembled, changes made and the resulting code assembled again to produce a working virus, the offset of the pattern (which is an essential parameter to precision scanning) may change. Precision scanning can also be undermined by unexpected viral behaviour or anomalies. Unless developed with extreme attention to detail, precision scanners will fail in the event of unusual program behaviour typified by overlay files (*.OV?) (Technical Notes, *VB*, February 1991). Moreover, ‘virus sandwiches’ such as the Plastique-Jerusalem-Plastique phenomenon (Technical Notes, *VB*, October 1990, pp.4-5) are likely further to confuse precision scanners.

Multiple Patterns

Any modification to an existing virus may invalidate the identification pattern used to detect it. The Bulgarian virus writers have expended much time and effort in subverting *John McAfee’s SCAN* program through patching sections of the search patterns contained in the program within the original virus code. Changes may also be accidental, caused by a random memory error, but are usually deliberate moves to prevent detection by a specific scanner. There is also the possibility that such patching is incorporated to collect a reward offered by an anti-virus company for any “new” virus submitted!

One possible method to increase the chance of detecting new variants is to use multiple search patterns. For every variant a set of patterns is defined and if all the patterns can be found the scanner will assume it has correctly identified the virus. If only some of the patterns are found, the scanner may report an infection by a previously unknown variant of the virus. The drawback of using multiple patterns is obvious - a marginal increase in scanning time.

Speed

From a security stance, the accuracy of the scanner is the overriding consideration, but practicality demands that a reasonable scanning speed is maintained - nobody wants to wait for an hour while a relatively small disk is being scanned. Various methods can be adopted to enhance scan-speed - some of which apply equally to other types of software, other methods are exclusive to virus scanners.

An obvious tactic is to write the scanner, or the most time-critical parts of it, in assembly language. It is also possible to enhance scanning speeds by knowing when *not* to search for a virus. It is patently obvious that searching programs for boot sector virus identification patterns is inane! Searching ‘target-rich’ areas can be refined. For example, if a large .COM file starts with a JMP to a location near the end of the file, all

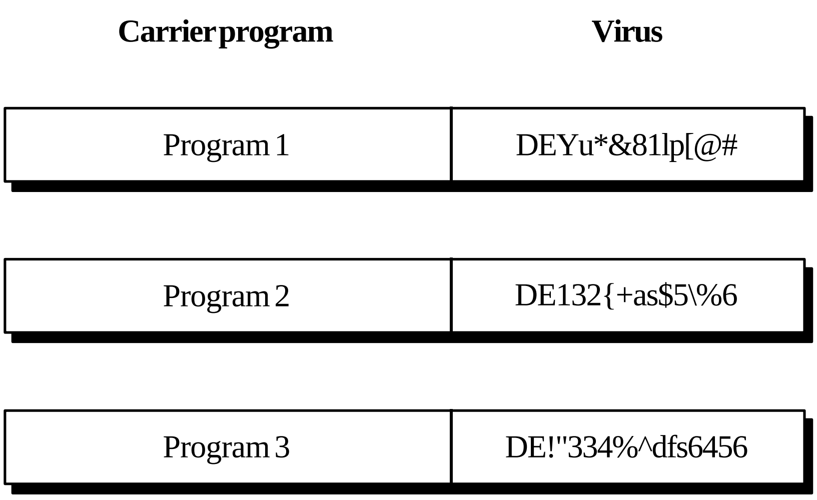

Figure 1. Three programs infected with an encrypting virus. Here the letter ‘DE’ in the virus code symbolise a static decryption key which provides a suitable code fragment from which to extract a search pattern. Cascade is an example of such a virus. The greatest threats are those viruses which employ a self-modifying (variable) decryption key offering no such opportunity. A recent virus example, V2P6, is close to being undetectable by virus-specific scanning programs.

viruses which infect the beginning of .COM files can be excluded from further consideration. Storing the first byte of the virus code for each virus, and comparing it to the first byte of the file being scanned (or the first byte in the code section, in the case of .EXE files) optimises scan-speeds. (To quote Alan Solomon "You don't search for London buses at the bottom of the Atlantic Ocean". Ed.)

When a Pattern is Inadequate

Although simple identification patterns are usually sufficient to detect viruses, there are a few exceptions. The encrypted viruses may cause problems, as the search pattern can only be taken from the decryption routine, which may be very small. In the case of viruses like Cascade this is not a problem, but viruses using self-modifying encryption are more difficult to handle. In some cases only a single byte in the decryption code may vary from one sample to another, so it is possible to provide a search string where one byte is marked as a '??' or "DON'T CARE" flag. The Ontario and USSR-1594 viruses are examples. In other cases a set of standard hex patterns may be provided for a multiply-encrypting virus - for example, one pattern for each decryption routine as in the case of the Whale virus (which results in thirty such patterns, see *VB*, December 1990).

In a few cases even these tactics are not practical - the 1260, Casper, V2P2 and V2P6 viruses are the best examples of self-modifying encrypting viruses from which no pattern is extractable. An algorithmic approach is necessary to detect these viruses, as their decryption code is too variable to use any search pattern. (V2P6 is the closest any virus has come to being undetectable by a scanner - for security reasons our original intention to publish data to detect this virus have been cancelled. The arrival of an undetectable virus adopting a 'sparse infection strategy' appears to be imminent. For a discussion of 'sparse infection', see '1260 Revisited', *VB*, April 1990, p.10. Ed.)

Summary

Algorithmic methods are the only option in a few rare cases, but are they practical for more general use? The authors of current virus scanners generally avoid them and employ identification patterns predominantly or even exclusively.

However, the continuing appearance of 'research viruses' (mostly written by Mark Washburn) which confound conventional detection have accelerated research into algorithmic detection methods.

Searching for identification patterns is still the fastest and most effective method to detect current computer viruses (and is liable to remain so for the forseeable future), but several other methods are nevertheless feasible and research and development into these methods is underway by a number of software developers.

IBM PC VIRUSES

Amendments and additions to the *Virus Bulletin Table of Known IBM PC Viruses* as of 24 February 1991. The full table was published in the January 1991 edition of *VB*.

Hexadecimal patterns can be used to detect the presence of the virus with the 'search' routine of disk utility programs or, preferably, can be added to virus scanning programs which contain pattern libraries.

Dutch-555 - CER: A compact 555 byte virus from Holland. The virus contains no side-effects.

```
555     5B58 072E FF2E 0500 813E 1200 4D5A 7406 ;
        Offset 19E
```

Casino - CR: Virus infects COM files increasing their lengths by 2330 to 2345 bytes. Upon triggering on the 15th January, April or August (any year), the virus plays an interactive game of Jackpot, the loss of which results in the destruction of the file allocation table in the default drive. (*VB*, Mar 91)

```
Casino  594B 7504 B866 06CF 80FC 1174 0880 FC12
```

INT13 - CR: A 512 byte stealth virus which is virtually immune to detection when memory-resident. The following search pattern will be found in infected files but only if the INT13 virus is not resident in memory. (*VB*, Mar 91)

```
INT13   E200 50BF 4C00 5733 ED8E DDC4 1DBF 7402 ;
        Offset 0
```

Micro-128 - CR: This virus from Bulgaria is the smallest memory-resident virus known. It occupies a portion of the Interrupt Table. The virus contains no side-effects.

```
Micro   4231 C931 D2CD E0B1 03B6 03C3 E90A 0080 ;
        Offset 030
```

Number One - CN: An old, simple overwriting *Pascal* virus which was originally published in *Computer Viruses: A High Tech Disease* by Burger. The length of the virus is dependent on the compiler used. 11980 and 12032 byte examples have been isolated. As with many other high level language viruses, a single search pattern is not possible.

Sex Revolution - MR: Two variants of this virus are known but both contain the text:

```
EXPORT OF SEX REVOLUTION
```

The virus is a minor modification of the New Zealand virus and is detected by the New Zealand (2) pattern last published in *VB*, January 1991.

Swedish Disaster - MR: The name is derived from a text string inside the virus. Awaiting analysis.

```
Swedish 0102 BB00 02B9 0100 2BD2 9C2E FF1E 0800 ;
        Offset 04A
```

Vienna-622 - CN: A new variant of the Vienna virus from Bulgaria. It is detected by the Vienna (4) search pattern (*VB*, January 1991).

SOFTWARE STRATEGY

Dr. Jan Hruska

Defining Executable Code in the Advent of Windows

Arguably, the single most important characteristic of a computer virus is that its code must **execute** before it can replicate.

The executable path used by viruses is normally covert and exploits one or more features of the operating system. For example, boot sector viruses execute when the PC is (accidentally) booted from an infected disk, parasitic viruses execute in advance of a legitimate program to which they are attached, while companion viruses execute when the operating system executes the COM version of a program instead of the legitimate EXE version.

It is of paramount importance to bear this in mind during any anti-virus activity, ranging from planning an anti-virus strategy to dealing with a virus outbreak. Almost any physical or logical storage device can contain a virus, but if the virus hopes to be executed at some stage, it must also modify an item which would normally be executed by the operating system or other executables.

For example, a virus may hide itself in Track 40 of a floppy disk, but since Track 40 is neither executable nor normally reachable by the operating system, it must also place a ‘bootstrapping’ link somewhere in the executable code. An obvious place is the boot sector, which is executed at startup.

Taking this argument further, let us once again clarify the recent confusion in the popular press regarding the possible infection of *data files* by computer viruses.

The 4K virus (see *VB*, November 1990, p.5) is an example of a data infector. The virus does, indeed, infect any file in which the sum of characters in the extension is 223 or 226. Apart from COM and EXE files, this includes files such as WITCH.OLD, MONKEY.MEM, PHUT.PIF etc. All of these files **will** contain the virus code, but only COM and EXE files will become *carriers* in the true sense of the word, and, hence, pose a threat. All other files will probably be diagnosed as corrupt if accessed on another (uninfected) PC.

It is important to differentiate between the ‘inactive’ virus code (such as that carried in the *data file* MONKEY.MEM infected with 4K) or ‘active’ code (such as that carried in the infected *executable file* ZOO.EXE). Typing ‘MONKEY’ at the C:> prompt will not infect the computer, while typing ‘ZOO’ most certainly will.

Fortunately, with MS-DOS it has traditionally been easy to identify executable items by following the PC bootstrap process (see *VB*, April 1990, p.11).

The executable items are:

1. **Master Boot Sector** (sector stored at absolute Track 0, Head 0, Sector 1).

2. **DOS Boot Sector** (logical sector 0 in each DOS partition). On floppy disks, this is the same as the Master Boot Sector.

3. **Operating system files** (first two files in the root directory, normally called IBMBIO.SYS and IBMDOS.SYS or similar).

4. **Device drivers** specified in the CONFIG.SYS file, which by itself is **not** executable.

5. **COMMAND.COM** - command line interpreter (a special COM file).

6. **AUTOEXEC.BAT** - a sequence of batch commands executed on every power-on (a special BAT file).

7. **COM, EXE and BAT files executed at the command line** level, for example WS.COM executed by typing ‘WS’ or SEX.BAT executed by typing ‘SEX’.

8. **Overlay files** - executable code pulled into memory by COM and EXE files as and when necessary. They are usually called OVL, OV1 etc.

In addition, an executable file can execute ‘Macros’, which are a form of interpreted code. This comprises *Lotus 1-2-3*˜ macros, interpreted Basic programs and so on.

The overwhelming majority of viruses, both in the number of individually identifiable specimens as well as the number of infections, are transmitted by DOS boot sectors, COM files and EXE files. That is understandable, as those items are readily exchangeable between computers - the first as a part of floppy disks and the second and the third in the form of applications software.

From the point of view of the virus-writer, success is measured in the number of infected systems, and the more a particular item is exchanged, the more likelihood there is of the virus spreading.

This probably accounts for the comparative paucity of BAT file viruses (although there are a few around) as well as viruses designed to exploit other executable items such as *Lotus* macros in spreadsheets.

In addition, the virus needs a common mass-platform in order to spread. There is little point in a writing a virus which runs on 31-bit ‘Goof’ processors, of which there are only three in existence. *Some 36 million IBM-PCs and compatibles, used in developed and developing nations worldwide, all supporting a common disk format and capable of executing the same code are a much more suitable platform*.

Fortunately, the IBM-PC architecture and the PC-DOS operating system (unlike the Apple Macintosh) allow a reasonably straightforward definition of what is executable and what is not. However, this clearly defined situation could become more complicated in the near future with the increase in popularity of *Microsoft Windows*˜ , which seems to be well on the way to becoming a de-facto standard GUI platform.

Unfortunately, *Windows* will inevitably prove to be a tempting target for virus-writers. The potential threat cannot be ignored and the traditional concept of what is executable under PC-DOS must be augmented by additional items which are executed by *Windows*.

Specifically, *Windows* adds two further vulnerable groups of files to the established list of potential target items:

- PIF files
- Dynamic Link Libraries

PIF files are normally created during the installation of the software, but are sometimes also copied between PCs. They describe command line arguments for various applications and in that sense resemble BAT files. It would be easy for a modified PIF file to implement the *Windows* equivalent of a companion virus. The integrity of these files is extremely important. It should be noted that the 4K virus already corrupts PIF files - albeit as a result of an unusual infection method rather than as an intentional effect.

Dynamic Link Libraries are files which can have any extension, although DLL and DRV are the most common. They contain executable code which is common to applications residing in memory and which can be shared between them. It is unlikely that DLL files will be copied between PCs to the same extent as EXE and COM files, so the threat of virus infection carried in them is reduced. However, a virus specifically targeted to spread via Dynamic Link Libraries is feasible.

The Implications

What are the readily apparent implications of *Microsoft Windows* for anti-virus software and anti-virus policies?

First, scanning software will not be affected until the first *Windows*-specific virus is discovered. After that happens, the affected groups of files will have to be added to the list of file extensions inspected by default by this type of program.

Second, checksumming software should be adapted to include PIF and Dynamic Link Library files in the list of items to be fingerprinted. This addition is relatively easy on most of the better checksumming software packages and should be undertaken on any PC running *Windows*. A *Windows*-specific virus may be a year of two away. Nevertheless, obvious countermeasures can be implemented today.

VIRUS ANALYSIS 1

Jim Bates

The INT13 Virus - A New Level of Stealthy Sophistication

Most examples of virus code display a distinct lack of programming finesse. However, every once in a while, a virus appears which shows undoubted expertise on the part of the author. INT13 is one such example. It is difficult to decide which is more reprehensible - the childish destructive code produced by the amateurs (or “pimplies” as *VB* correspondents generally refer to this group) or the more accomplished ‘creations’ of experienced programmers who misuse their skill to damage their own industry. Patience is a virtue and we should try to educate the “pimplies”. However, there is absolutely no excuse for the professional virus-writers. It is no use their claiming that a particular virus was written for “research” purposes - every virus will cause some disruption and possibly damage if it is allowed to infect a PC environment unhindered.

True Stealth Characteristics

A highly sophisticated virus arrived in December 1990 in a package of examples from Eastern Europe. The virus has been called INT13 after a plain text string at the end of the code. **It contains a potent ‘stealth’ capability which will avoid detection by all but the most determined memory-scanning programs if it is active at the time of scanning.**(See TSR Monitors and Memory Scanners, pp. 18-19, Ed.)

The total code is only 512 bytes in length and there is no trigger routine in the sample which I analysed (although file corruption will occur under certain circumstances - see report below). A simple, uncommented disassembly listing was also included in the package but examination of this was inconclusive in determining whether it was the author’s original source or an unknown researcher’s primary efforts at disassembly.

The virus is categorised as infecting only COM files of certain lengths and bears some remarkable similarities to the Number of the Beast (666) virus already discussed (*VB*, May 1990). No change occurs in the file length, attribute settings or date/time fields and the COMMAND.COM file is specifically excluded from infection.

Installation

This virus overwrites the first 512 bytes of its target program file and the code therefore initially appears at offset 100H of the program code segment. The very first instruction is a two-byte LOOP Next command which is the hex word E200H. This is used later by the virus as an infection indicator. After this first instruction (which does nothing), the normal machine interrupt vectors for INT 13H and INT 21H are collected by direct access to the Interrupt Table and copies are stored in the entries for INT 9CH and INT 9EH respectively.

An undocumented function of DOS is used to collect the ROM entry point of the INT 13H routine and this address is copied into the INT 9DH vector. The address of the first DOS Disk Buffer is then obtained and the virus code (all 512 bytes) are copied into it. The chain of buffer addresses is then updated to remove this first buffer from subsequent DOS operations. Up to this point, no re-vectoring of the original interrupts has occurred. This first section of the program then pushes the relevant segment and offset addresses onto the stack and issues a RETF instruction to transfer processing into the relocated code in memory.

The continuation code starts by accessing low memory and re-vectoring (by direct access) the interrupt addresses for INT 13H and INT 21H to handlers present within the virus code. Then the environment segment address is collected from within the host program’s PSP and the host program is loaded into memory in exactly the same place as it was previously. This might seem a pointless exercise but as the operation of the INT 13H handler becomes apparent, it will be seen that this second load of the host program will produce a correct (i.e. uninfected) copy of the original program file in memory. Once loaded, the host program is executed under normal DOS control.

Memory-Resident Operation

The operation of the virus interrupt handlers is somewhat involved and displays an intimate knowledge of how DOS handles requests for file information. It is difficult to describe the functions of the handlers separately since they operate together, so this analysis will describe the sequence of events after both handlers have been installed and examines particular aspects of each as they occur.

It should first be noted that the virus contains a *third* interrupt handler which is temporarily hooked into the INT 13H vector whenever the infection routine is processed. It is this temporary handler which provides the key to the low-level subversion undertaken by the virus.

Infection Routine

Starting with the infection routine, the INT 21H handler intercepts only function 12H (FCB FIND NEXT). The requested function is first issued through the newly created INT 9EH (original INT 21H) call. This is checked for error free completion and the virus then accesses the Disk Transfer area to examine the filename which was found. If the last two letters of the file extension are ‘OM’ and the second and third letters of the name are NOT ‘OM’ then size checks are initiated (this is to avoid infecting COMMAND.COM). Otherwise the handler aborts and returns control to the caller.

The size checks consist of first rejecting any files less than 512 bytes in length, and then testing bits 5 and 6 of the file size field for zero. This is an effective method of determining how much allocated space is available beyond the end of the file (in a similar way to the 666 virus). Provision is also made for the difference in allocation unit size between fixed disks and floppy disks although no actual checks are made to determine the true allocation unit size. If examination of the file size field reveals that there is at least one sector available beyond the end of the file, it is suitable for infection.

At this point the temporary INT 13H handler is hooked into the interrupt chain, bypassing the main INT 13H handler. The existing INT 13H vector is stored (but not used) at the INT 9FH address in the interrupt table. The target file is then opened for READ ONLY access using the original DOS INT 21H vector and a SEEK to End of File instruction is issued with an offset of -1. This effectively seeks to the last byte in the file where the temporary INT 13H handler is invoked.

During a disk access call to INT 21H, DOS will call INT 13H a number of times as information is required from the disk. During the SEEK to EOF function, the final call to INT 13H will contain the logical Drive, Track, Head and Sector address of the last cluster of the file. The temporary INT 13H handler does nothing more than store these address variables into data areas within the virus code and then continues with the original (ie: unintercepted) INT 13H. So these data areas now contain the low-level address of the last cluster of the file and they are accessed and stored on the stack.

Next, the virus issues a SEEK to Beginning of File instruction and then uses the DOS INT 21H address to read the first word of the file. This word is checked to see if it is the virus ID word E200H and if it is, the handler aborts (after first unhooking the temporary INT 13H vector). When DOS INT 21H is asked to read from a file, the whole of the relevant allocation unit is read into the DOS disk buffer. Thus the single word read call collects the first cluster **and** sets the address of that cluster into the temporary INT 13H variables. The first word of the file is also checked to see whether it is "MZ", indicating a fake COM file with an EXE header.

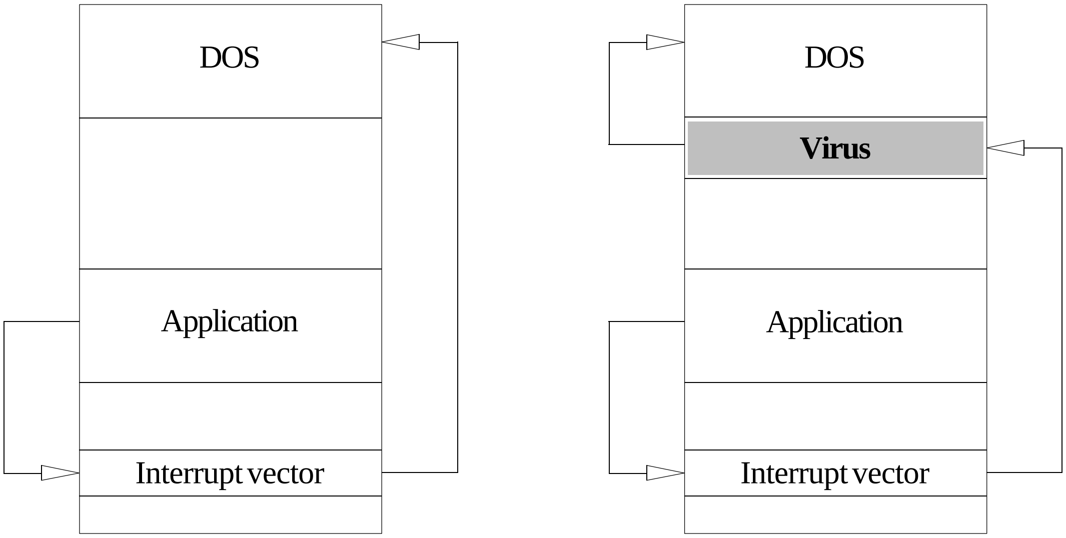

Figure 1. Interrupt routing before and after infection. INT 13 is a 'stealth' virus which uses interrupt interception to escape detection. MS-DOS applications use software interrupts to communicate with the operating system in a portable way. The jump addresses are stored in the interrupt table located at the beginning of memory. If a virus changes one or more of these addresses, any jumps to the operating system can be routed through the virus, which then decides what should be done with a particular request. The very first MS-DOS virus, Brain, employed interrupt interception. When active in memory, the virus intercepts any attempt to read from disk the contents of Sector 0 (the hiding place of Brain). The virus re-vectors such calls to point to the legitimate contents of Sector 0 which it has stored at a completely different location. Hence, when Brain is in memory, any attempt to look at the boot sector will not reveal its presence.

The check does **not** include the less usual "ZM" header signature. Files with an "MZ" header are not infected.

The last cluster address variables are popped off the stack and used to address a READ SECTORS request to the ROM entry point of the INT 13H BIOS routines. This bypasses the temporary handler and thereby does not update the address variables. These two sectors are read into a buffer created in the middle portion (offset 7800H) of the video memory area at segment B800H. So we now have the first sector of the file in the DOS Disk Buffer and the last sector (plus the subsequent sector of unused but allocated space) in the temporary buffer in video memory.

> *"It contains a potent stealth capability and if active in memory will avoid detection by all but the most determined scanning programs."*

The complete virus code (containing the temporary INT 13H data variables, pointing to the last cluster) is written to the first sector of the file using the unintercepted INT 13H vector, thus overwriting the first 512 bytes of the file. The original first sector of the file is copied to the second of the two sectors stored in video memory and these two sectors are written back to their original position. The target file is now fully infected - with the virus code in the first 512 bytes, and the original first sector stored beyond the end of the file. Finally, the target file is closed, the temporary INT 13H handler is unhooked from the system and the original register settings are restored before processing returns to the original caller.

Interrupt Interception to Escape Detection

We must now consider the operation of the main INT 13H intercept routine as it functions within the DOS scheme of operations. When a COM file is loaded prior to being executed, the DOS INT 21H handler will call upon the installed INT 13H vector for information and data. At some point, the first cluster of the file will be loaded into the disk buffers before being moved to the caller's load buffer.

This virus' INT 13H handler will examine each sector load to see whether it starts with the E200H virus self-recognition signature. If this is not found, the handler immediately returns to the calling routine. Otherwise, the address data variables are collected and used to read the last two sectors of the file (using Track, Head and Sector addressing into the ROM entry point of the INT 13H BIOS routines). The second sector (the original program header) is then copied into the original caller's buffer before processing returns.

It can thus be seen that once the virus intercepts have been installed, any infected file loaded into memory using normal DOS function calls (or even normal BIOS calls) will result in a **clean** file being loaded.

Once the INT13 virus is in memory, it subverts both DOS and BIOS calls and will thus deceive most scanning programs into thinking that every infected file on the hard disk is clean.

Conclusions

This complex method of replacing the original file header by subverting the BIOS access routines produces some interesting results which directly affect the virulence of the infection.

When an infected file is copied on a system with the virus resident in memory, the resulting file is uninfected and may only be re-infected during a DOS FCB FIND NEXT function request. However, since this virus does not infect COMMAND.COM, different implementations of DOS (with different sized system files) may not allow system infection. It is quite probable that a power-down reboot will leave a machine uninfected. This becomes more likely when one considers that only files with certain size limitations can be infected. So, the effect of copying infected files on an uninfected machine will vary depending upon the copying methods used. Some implementations of DOS will copy on a sector by sector basis and these will produce an infected (i.e. overwritten) file which does not contain the original program header within the free space beyond its end. Such files will be irreparably damaged. Other versions copy on a cluster by cluster basis and these will produce a correctly infected file with all the requisite information intact.

Other problems of a less predictable nature will occur as a result of several shortcomings in the virus code. First, the absence of a check for the "ZM" header may occasionally cause this virus to try to infect a fake COM file. Since the virus uses a simple load to re-execute the program code at offset 100H this will cause unpredictable effects. Second, the use of video memory as a buffer could cause display problems on packages which would normally used the 1024 bytes stolen by the virus as a temporary buffer. Screen corruption would probably be the only effect here although this is only speculative. Finally, the INT 13H intercept which prevents the correct loading of any sector which begins with E200H will cause problems if this particular byte sequence is encountered within a file as a normal part of the file's contents (remembering that this could be data or a split instruction across a sector boundary).

Admittedly these conditions may be rare but there is no doubt that problems **will** occur when they are met. The removal of one of the DOS Disk Buffers from service may also cause slight performance degradation for tightly configured file intensive applications.

The Moral of the Story

This virus, which is singularly ‘slippery’ when resident in memory, should impact upon existing attitudes towards detection. Software *could* be constructed to disable the virus while it is active in memory with some (unassured) degree of accuracy. However, the benefits of expending effort on such a task would be better directed elsewhere - most anti-virus software documentation would benefit far more by the insertion of a prominent warning in blood red ink, saying:

BEFORE USING THIS PROGRAM,
YOU MUST BOOT YOUR MACHINE
FROM A CLEAN WRITE-PROTECTED SYSTEM
FLOPPY DISK.

As with other ‘stealth’ viruses searching for instances of INT13, whether with a disk utility or with a scanning program can only be conducted safely if the machine has first been booted from such a disk.

Summary

The INT13 virus is an overwriting ‘stealth’ virus which subverts both DOS and the BIOS in concealing its whereabouts on disk. The code length is 512 bytes although this is not added to the observed file length. Infection only takes place on selected COM files and only during the FCB FIND NEXT function call. The virus does not encrypt, either on the disk or in memory. A reliable search string is as follows:

```
E200 50BF 4C00 5733 ED8E DDC4 1DBF 7402
```

This pattern will be found at offset 0 in infected files **but only if the virus is not memory-resident at the time of scanning**

The Washburn Viruses

The April 1991 edition of *Virus Bulletin* will tell the story behind Mark Washburn’s V2P1 (1260), V2P2 and V2P6 computer viruses. Washburn explains the motivation behind their development and the way in which they were distributed. Describing V2P6 as the first “true patternless virus”, Washburn claims his viruses are ‘experiments’ to demonstrate the weakness of “relying on fixed scan strings as the sole method of detection”. *VB* will analyse the impact which the use of self-modifying encryption in virus code will have on conventional scanning techniques and will provide a technical analysis of the viruses themselves.

VIRUS ANALYSIS 2

The Casino Virus - Gambling With Your Hard Disk

This virus (which *VB* called Casino for reasons that will become clear) was received from a user in Malta where instances of data destruction were reported at a number of sites on January 15th 1991. Casino is particularly unaccomplished both in its coding and execution. Nevertheless, it is highly infective and very destructive.

It is a memory-resident virus which infects only COM files, increasing their length by between 2330 and 2345 bytes. January 15th is one of three trigger dates.The code is highly convoluted and there are several mistakes which will cause various extraneous effects on infected machines. The code is appended to the target COM file after rounding the file size up to the next paragraph boundary (divisible by 16). Execution is arranged by modifying the first four bytes of the target to become a jump instruction into the body of the virus code.

The first action that the code takes is to calculate a data segment value which will allow access to the various values stored throughout the code. This primitive method of producing relocatable code needs to be self-modifying and involves cumbersome segment manipulation routines. This is just one of the many indications that the programmer is technically incompetent.

Once this data access re-segmentation is completed, the code immediately repairs the host file header by returning the first four bytes to their original value. A call is made to obtain the system date which is checked to see whether it is the fifteenth of either January, April or August (any year). If one of these dates is found, the trigger routine is executed (see below).

If a trigger date is not found processing continues by issuing an “are you there?” call to determine whether the virus is resident in memory. This consists of placing a value of 4B59H into the AX register and issuing an INT 21H request. If the virus is resident, the function returns with 0666H in the AX register and the code immediately resets the DS register (to the CS value) before jumping to the host program.

Memory-Resident Installation

If the “are you there?” call goes unanswered, the first 256 bytes of program code are copied to a buffer area within the virus (seemingly to make room for data storage!) and an attempt is made to access COMMAND.COM on the first hard drive. The actual file specification is hard-coded into the virus as C:\COMMAND.COM and if found, the file is checked for infection and infected if found to be clean.

The next phase of disassembly provided some light relief as the programmer's efforts to make his code resident became apparent. A file called COMMAND# .COM is created on the default drive where the # character is the infamous 'phantom space' ASCII character code 255 which was used within the AIDS Trojan in December 1989 (see *VB*, January 1990).

Once this file has been created, the virus code is written to it and the file is closed. Then a LOAD & EXECUTE call is issued to the system so that this file is loaded and executed as a child process. Once the file is loaded, the data segment calculation is again in evidence before a new INT 21H handler is installed. The child process terminates using an INT 27H call to make the code permanently resident. This incredibly convoluted method of installing resident code will leave an unusable area of memory (a 'hole') equivalent to the size of the host program plus twice the length of the virus code.

INT 21H Handler.

The first routine within this intercept checks for the virus "are you there?" call and answers with the 666H value in the accumulator. The next section checks for incoming requests for the FIND FIRST (11H) and FIND NEXT (12H) functions. These are the older style requests which deal solely with files handled using File Control Blocks (FCBs). No attempt is made to subvert the parallel functions dealing with file handles (4EH and 4FH).

Within the FCB FIND intercept routine, the coding is fairly straightforward except for the introduction of a clumsy attempt to make this section of DOS available for re-entrant use. If either of the intercepted function calls is accompanied by a value of 66H in the AL register, the request is allowed through without interference. Thus the routine first completes the function request by adding this re-entrant flag and issuing an INT 21H request. Then the address of the Disk Transfer Area is collected and the DTA is examined to see whether the found file has a COM extension. If it does not, processing is returned to the caller. COM files detected in this way are then examined by subtracting the length of the virus code (2330 bytes) from the length of the file and checking the remainder (the original length of the file, rounded as described above) to see whether it is an exact number of paragraphs. If it is, the FCB size field is modified to the rounded figure before processing is returned to the caller, otherwise the FCB is left unchanged. Correction is not applied if the found file size is less than the length of the virus code. Thus any file which matches these criteria, whether infected or not, will have its reported FCB length truncated in this way.

The next section of the intercept deals with the re-entrance problems by maintaining two flag bytes, one for execution of INT 24H and the other for current use of INT 21H. Each time this section is executed, a value of zero is placed in the INT 24H flag and the value of the INT 21H flag is checked for 0FFH. If this value is found, processing returns immediately to the caller. Otherwise, the value 0FFH is placed into the INT 21H flag and a check is made for functions 4BH (LOAD & EXECUTE) and 36H (GET FREE SPACE). If found, each of these is intercepted by an appropriate routine. If some other function has been requested, the INT 21H flag is reset to zero before processing returns to the caller.

The LOAD & EXECUTE intercept collects the default drive by issuing a function 19H request (this function is issued twice within the code, an example of the messy and immature programming within this virus). A test is then conducted to see whether the default drive is a floppy (A: or B:) or fixed drive. If it is a floppy drive, calls are processed to read and then write the boot sector (using INT 13H). No modification occurs - an apparent attempt to determine whether the floppy is write-protected. A similar check is made during the initial part of the function 36H intercept but with corrections introduced for the differing drive numbering convention used.

Infection Routine

Processing for both intercepts converges and if the target drive is available for write requests (i.e. a fixed disk or an unprotected floppy) a search is conducted using the file handle functions 4EH and 4FH (FIND FIRST and FIND NEXT) for any available COM files. Once found, the size of the target COM file is tested to see whether it is above 62905 bytes. If it is, checking aborts. Otherwise the attributes of the file are checked for the SYS attribute (in which case the check routine is aborted) or the READ ONLY attribute (in which case this is reset). The target file is then opened (using function 3DH) for READ/WRITE access and the first byte is checked for a value of 90H (NOP). If this is found, the file is assumed to be infected and processing aborts, otherwise a flag indicates that the file is uninfected. If an infected file is found, the search continues for the next uninfected COM file.

When an uninfected COM file is found, a 'standard' infection routine is called which which collects the file date and time, calculates an initial jump offset and inserts this into the head of the host program (including the NOP indicator), storing the original program header and appending the virus code. A temporary critical error handler (INT 24H) routine is installed and this modifies a flag when it is executed. There is also provision for inserting a zero into an uninitialised address variable. This portion of the code makes no sense and the results are unpredictable since the target address is always 0000:0000. No attempt is made to repair the READ ONLY attribute on files which possessed it before infection, although the file time and date are restored.

The Trigger Routine

This is invoked during initial execution of the virus code (before it becomes resident) if the system date indicates the 15th of either January, April or August. No particular significance has been found which relates these three dates. **The first action of the trigger is to collect and store the first 80 sectors of the default disk, and then overwrite them with garbage from page zero of memory**. A primitive decryption routine is then processed which decrypts the messages relevant to the “game” which is about to commence. Once decrypted, the game screen appears and the user is forced to participate in a macabre game of roulette (see below).

The ‘‘Game’’

```
          DISK DESTROYER . A SOUVENIR OF MALTA

      I have just DESTROYED the FAT on your Disk !!
    However, I have a copy in RAM, and I'm giving you
       a last chance to restore your precious data.
    WARNING: IF YOU RESET NOW, ALL YOUR DATA WILL BE
                     LOST - FOREVER !!
           Your Data depends on a game of JACKPOT

                CASINO DE MALTE JACKPOT
                     +=+  +=+  +=+
                     +L+  +?+  +c+
                     +=+  +=+  +=+
                       CREDITS : 5

                     LLL = Your Disk
                    ??? = My Phone No.

                      ANY KEY TO PLAY
```

As any key is pressed, the game is played with the windows showing progressive characters and stopping in sequence displaying one of the three characters “L”, “?” or “c”. The results of playing the game are as follows:

LLL - The FAT area of the disk is repaired and the following message is displayed:

```
BASTARD ! You're lucky this time - but for your
own sake, now SWITCH OFF YOUR COMPUTER AND DON'T
TURN IT ON TILL TOMORROW !!!
```

??? - A message is displayed and the machine hangs. The FAT of the default disk is destroyed. The message is:

```
No Fuckin' Chance; and I'm punishing your for
trying to trace me down !
```

Any other combinations and the game is lost. This results in the following message being displayed before the machine hangs:

```
HA HA !! You asshole, you've lost: say Bye to
your Balls ...
```

Comments and Conclusions

Both the concept and execution of this virus are infantile in the extreme. However, there are some strong clues which might help to identify the author. The most obvious of these is the Americanised language used in the messages, particularly the use of the apostrophe for the missing “g” in the second message and the use of the word “asshole”. There are other interesting anomalies but publication of these has been withheld pending police investigation. Preliminary enquiries indicate that this virus does indeed originate from Malta.

The lack of expertise in the code would indicate an adolescent programmer with very limited experience. The trigger routine was probably copied from elsewhere - this is a standard practice among virus-writers - the falling characters display of the Cascade virus, for example, is believed to have been purloined from a public-domain joke program.

Casino is thought to be the first ‘interactive’ virus which, upon triggering, challenges the user to participate in a ‘game’ of the virus-writer’s choosing.Taunting and profane messages and bizarre screen and sound effects have traditionally predominated as standard computer virus side-effects. However, the appearance of more computer virus samples which incorporate game routines (possibly stolen from professional games software) should be anticipated.

Disinfection

Casino will infect only COM files within the current default directory, and if detected before triggering, will be easy to remove by deleting all infected files and replacing them from master copies (**after a clean system reboot**). The virus employs primitive ‘stealth’ methods designed to conceal its existence from the user, but detection will present no challenge to professional anti-virus software developers.

Summary

The Casino virus infects COM files smaller than 62905 bytes. It is memory-resident and subverts function calls 11H and 12H to hide its length from directory searches. The virus infects on function calls 4BH and 36H. The infective length is between 2330 and 2345 bytes depending upon size of original file. The code is non-encrypting except for the game message area (this prevents the display of plain-text during a cursory file examination).

A reliable search pattern has been extracted and is as follows:

```
594B 7504 B866 06CF 80FC 1174 0880 FC12
```

Infected files are recognised by having a NOP (character 90H) as their first instruction. The virus self-recognition in memory (the “are you there?” call) is hooked to function 4BH as subfunction 59H and a returned value of 666H in the AX register indicates that the virus is resident.

OPINION

Ed.

TSR Monitors and Memory Scanners - The 'Playground' Approach to Virus Detection

A fax was sent to *VB* recently from a software developer who had been supplied with the detection patterns of the WHALE virus. The developer wanted to know how the virus could be detected in memory. The reply was simple - "you don't allow something as slippery and slimy as WHALE to get there in the first place!"

Speaking as a bemused *user*, it is not yet clear to me why anybody in their right mind should wish to search for viruses in memory. I have been told that such a capability would be helpful in the event of a network infection - it would allow the file-server to remain powered-up and in 'normal use' while the virus-hunt commenced. I would contend that attempting to disinfect a LAN infection while allowing users to continue running and transferring infected executables is a patently ludicrous proposition.

I work on a fundamental precept - to purge a memory-resident virus from a computer or network you deny it the chance to survive. **By offering it no opportunity to become resident you immobilise the virus and render its ability to intercept and re-vector interrupt calls null and void.**

It is a fallacy to pretend that operations can proceed undisturbed during a virus attack. A file-server infection (which is among most system managers' bleakest nightmares) can be described as a genuine denial-of-service attack requiring a commensurate sacrifice in system availability. Operations (and peace of mind) can only be restored by a *systematic* and *foolproof* approach to virus detection; the cornerstone to such an approach is an old friend - the **write-protected system floppy disk.** In blunt terms, there is no point trying to detect a virus if that virus is in control and in all likelihood monitoring attempts to locate it. The virus adopts the Transient Program Area as the field of battle, the key to victory is to deny it this battleground and fight it on territory of your own choosing.

The fundamental approach to computer security is far more enlightening than the pursuit of quackery. One of the first principles of software security (ironically, often better understood by hackers than security professionals) is:

_ software barriers can be breached by other software.

Let us paraphrase this principle:

_ anything done by a TSR monitor can be undone by a virus.

From a security viewpoint, memory-resident computer viruses should be considered as TSR programs deliberately designed to subvert other programs in memory and that *includes* virus-monitoring programs be they of a generic or specific nature. This is not to imply that the standard Jerusalem virus (or, indeed, many other equally tedious viruses) actually undermines memory-resident anti-virus software - it does not. However, a consistent and fundamental approach does help counter the small but increasing number of viruses (such as Flip, 8 Tunes and INT 13) which **do** actively subvert memory-resident virus monitors.

Moreover, the fundamental approach is essential to combat those viruses which conflict with TSRs of all types (including anti-virus monitors). Having witnessed Titanic and often inconclusive battles waged in the Transient Program Area between WHALE and various other memory-resident programs, my conviction that the TSR approach to virus detection is bound to be short-lived has become absolute.

Two anti-virus monitors *Bombsquad* and *Flushot+* have already been subjected to such subversion; the fact that other monitors are not being singled out for attack is, I hope, a reflection of this type of program's general unpopularity and limited use. Be in no doubt - should a TSR anti-virus program be widely adopted it will become the subject of attempted subversion; as *VB* reported in December 1990, the Bulgarian 'virus factory' is going to extraordinary lengths to undermine the popular *McAfee Associates' SCAN* software.

> *"These programs are to computer security what the rhythm method is to contraception..."*

Flushot+ and *Bombsquad* are generic monitors which intercept 'suspicious' interrupt calls in the same way that viruses intercept 'legitimate' interrupt calls - **the irony and weakness of this approach is that both the virus and the monitor function on <u>exactly</u> the same principle**. The mechanism used by anti-virus software for intercepting disk reads and writes by manipulating either the vectors in the DOS Interrupt Table or the code to which the vectors point, is *precisely* that used by computer viruses. Thus the battle, far from being fought by applying fundamental security principles, takes place between programmers trying to outwit each other. It's the "anything you can do, I can do better" school-playground approach to virus detection.

Defining 'suspicious' activity is an intractible task. Disk utilities are obviously contentious, but many programs issue legitimate interrupt calls which deviate from the INT 21H thoroughfare and other well documented highways. Such legitimate activity can be misinterpreted as ‘suspicious’ by monitors and results in tiresome false positive warnings to the user. Conversely, any virus which does not comply with the monitoring program’s concept of viral activity will remain undetected. Here, the designer of the protective software must stretch his intuition, insight and foresight into predicting all the tricks in ‘the book’ and the countless hundreds or thousands that aren’t. The research community has been taken by surprise often enough to know that it dare not make too many predictions. The days when computer viruses stood to attention and did what they were told have long since passed.

Forcing the TSR into memory ahead of any other application and **keeping it there intact** so that it assumes and maintains immediate control is a seemingly insoluble task. The first thing that ROM starting code places into memory is, of course, the boot sector - thus any anti-virus program summoned into action by CONFIG.SYS (a standard tactic) will load *after* a boot sector virus, giving the *virus* preliminary control.

System performance degradation is another penalty to be paid by following the TSR approach - the recent *VB* product review of *Norton AntiVirus* found that the TSR component (*Virus Intercept*) imposed an overhead on file copying of between 25 and 300 percent as files are scanned prior to being loaded. The danger of recurrent false alarms combined with the inconvenience of overhead associated with memory-resident programs could well result in their being disabled by the user - a disastrous diminution of security.

This article has mainly addressed generic monitors which try to ‘second guess’ the virus programmer. I contend that these programs are to computer security what the rhythm method is to contraception - the collective term for consenting adults who adopt the rhythm method is, of course, ‘parents’. However, there are also advocates of scanners which *search* memory - these programs incorporate the location of a known virus in memory and search for it accordingly. With both the virus and anti-virus program battling it out in memory, the assumed advantage to the anti-virus program is that it knows where to find the enemy (assuming of course that the virus does not have equal ‘intelligence’ about the scanner).

With regards to security, this approach is just as fundamentally flawed - the virus can re-vector interrupt calls to conceal its location, or identify the hostile program by some specific characteristic and surpress it. Again, it is a straightforward battle of wits - but fought on the *virus-writer’s* chosen battlefield. Even discounting the danger of intentional subversion (which is tantamount to dismissing ‘stealth’ viruses), the user of such a program must have absolute faith that the anti-virus programmer’s calculations are precise - if they are not, there is every likelihood of the virus escaping detection and surviving to fight another day. It should be remembered, also, that on 80386 and 80486 machines memory and I/O locations can be remapped in order to assist multi-programming - this renders memory-scanning an even more imprecise artform.

The logic of routinely running a memory-resident scanner from the hard disk (which presumably is the intended mode of operation) is questionable given the limitations of virus-specific software which must be updated regularly - try *updating* software on 1,000 PCs every month and the logistical difficulties will soon become apparent. With so many viruses appearing on such a regular basis, how do software developers find the time to incorporate these superfluous routines? Is it, as I suspect, being done at the expense of the fundamental and ongoing requirement to detect computer viruses reliably as and when they appear? Memory scanning, inoculation and disinfection routines are of dubious value and pale into insignificance in the face of the need for reliable, secure detection of the ever-increasing number of virus specimens.

On computer architectures which allow memory ownership, hardware memory protection and privileged instruction sets, memory-resident products can, theoretically, be implemented securely. None of these features are available on Intel 8086/88 processors or on the MS-DOS operating system - by ignoring this fundamental fact, the developers of these products continue to peddle an inherently insecure methodology.

TSR monitors and scanners designed to operate in an infected environment are symptomatic of an obsession with total automation - inoculation software is perhaps the most bizarre expression of this obsession. The ‘automation movement’ appears to be more interested in flexing its programming muscles than addressing practicality. *Detecting viruses is, and should remain, a relatively straightforward task*. It is being made complex by ill-informed marketing men who demand that superfluous and wholly irrelevant features be incorporated into anti-virus software. Systems managers, on the other hand, would be well advised to adhere to fundamental principles, one of the most important of which, is:

<u>Memory-resident virus scanners are inherently insecure</u>

To remove a virus from memory simply switch the PC off. To make sure it doesn’t return to memory, boot the machine from a write-protected system floppy disk. You are now in a formidable situation - you have **absolute** control over any virus code residing on the hard disk. As long as no suspect program is run (remember that the boot sector and all system files on the floppy disk are clean and secured by a write-protect tab), the virus, however ‘clever’, cannot hide itself. By adopting this method, no devious interrupt-interception and associated ‘stealth’ tricks which might enable the virus to hide are ever possible. Using an up-to-date scanner, the computer security officer can set about destroying the virus on a suspect PC wherever it may lurk with absolute assurance. In the words of a *U.S. Air Force* pilot speaking on television recently (albeit about an unrelated matter) “it’s a turkey shoot”.

[The views expressed are those of the editor and do not necessarily reflect those of VB’s technical editor, writers and editorial advisors].

END-NOTES & NEWS

Dr. Joseph Lewis Popp has been extradited to the United Kingdom to stand trial on charges of blackmail for his alleged part in the AIDS Information Diskette extortion attempt of December 1989 (*VB*, Jan 90). Popp, escorted by officers of the *Computer Crime Unit*, arrived in London on 22nd February 1991.

The UK's *National Computing Centre* reports that **computer virus infiltration within business computing systems in the United Kingdom** has increased by 29 percent since a similar survey was conducted in 1989. The centre describes its findings as "contrary to current theories held by many security specialists, who feel that the virus problem has been over-estimated. 34 percent of 497 corporate respondents had been infiltrated at some time by a virus. London and the South West, where 41 percent of respondents from the region reported a computer virus attack, were hardest hit. A similar survey conducted two years ago revealed that only 5 percent of respondents had suffered a virus attack. Dr. John Perkins of the *NCC* commented: "This increase could be ascribed to the user of IT now being able to identify virus attacks more easily''. The report is available from the *NCC*. Tel 061 228 6333.

S & S Ltd. has signed a **non-exclusive distribution agreement** with *IBM UK's PS/2 Division* for the distribution of *Dr. Solomon's Anti-Virus Toolkit*. IBM dealers will be able to source standalone packs and site licences using IBM part numbers. *S & S* says that orders worth over £30,000 at retail value have been placed. Mark Drew, IBM UK's Information Protection Programme Officer is said to have said: "I have a lot of respect for Alan's [Editor's note: Dr. Alan Solomon] professionalism in virus research. He is extremely thorough and conscientious." Tel 0442 877877.

Sophos Ltd, UK, has announced that the **price of *VACCINE* is to be cut** by half from 1st March 1991. The product is to retail for £99.50 while site-licence copies will be charged at £19.50 per computer. The company has also introduced a 24 hour, 365 days per year technical helpline in support of its products. Tel 0235 559933.

4th Annual Computer Virus & Security Conference, 14-15th March 1991, New York, USA. Contact the *Computer Society of the IEEE*, USA. Tel 202 371 1013.

Computer viruses and network security are the sublects of two seminars being held by *State of the Art Seminars*. The seminars will be presented by Dr. Douglas Tygar, Assistant Professor of computer science at *Carnegie Mellon University*, USA. The events will be held in Rome (10-13th April 1991), Munich (17-19th April 1991) and London (22-24th April 1991). Information from *SAL*, UK. Tel 071 404 3341.

Elsevier Seminars, UK, is holding seminars on **Commonsense Computer Security** (London, UK, 18-19th March 1991), and **Contingency Planning and Disaster Recovery** (London, 17-18th April 1991). Tel 0865 512242.

VIRUS BULLETIN

Subscription price for 1 year (12 issues) including delivery:

USA (first class airmail) US$350, Rest of the World (first class airmail) £195

Editorial enquiries, subscription enquiries, orders and payments:

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

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

US subscriptions only:

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

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein.