November 1990 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

EDITORIAL

Hope Springs Eternal

Some beneficial developments have arisen in the wake of the WHALE virus (which is reported in two separate articles in this month's edition, see pp. 17-20).

Initial analysis suggested that this particular program would be a 'tough nut to crack' and it soon became clear that collaboration between researchers would be necessary to accelerate the process of disassembly.

In fact, ad hoc arrangements to minimise the duplication of analysis and harmonise the disassembly process emerged within days of the program's discovery. Information and annotated disassemblies were transferred (in encrypted form) by e-mail among a central corps of programmers. The priority was to find a method by which the virus could be detected consistently - WHALE obligingly "lifted her skirts" and a number of proven detection methods have now been identified. (These methods are not being published but are available to bona fide anti-virus software developers through *VB*.)

The most encouraging aspect of this process was the unparalleled cooperation and consultation between the various researchers and programmers involved.The strengthening of communications links and increased cooperation among researchers should significantly reduce the response time in analysing rogue software in the event of a future emergency.

Various initiatives are currently under way in the United Kingdom to establish a government- and/or industry-funded research and reporting centre along similar lines to the *Computer Emergency Response Team* in the United States or the *Information Security Research Centre* at the *Queensland University of Technology*, Australia. It is important that any equivalent British centre is devoid of commercial interests or affiliation. This would suggest that it should be controlled by an academic or government body. *VB* hopes to be able to report some 'concrete' initiatives presently.

There is also an air of cautious optimism that MS-DOS and PC-DOS virus code is reaching its defined limits. Thorough study of undocumented operating system features has proved very helpful in analysing some of the latest computer viruses while standard virus writing tricks are becoming ever more readily identifiable. The 'armoured' viruses, of which WHALE is the most tortuous, convoluted and ludicrous, are a sign of resignation by the virus writers; the defensive bulk which they carry actually increases the likelihood of detection.

An additional benefit of ongoing research is the mass of forensic evidence which accumulates and which is carefully indexed for future reference. This information, which is constantly updated, is available in the event that the police should require technical support pending an arrest or prosecution.

Slowly but surely, an informal network is developing which can respond to the threat of malicious software. However, to be truly effective it will need to be formalised and properly funded. Regrettably, this is appears unlikely to happen until a genuine disaster occurs.

Cutting Out the Middle-Men

There is a well-founded suspicion that certain virus writers, far from wishing to cause computer havoc, are simply seeking publicity and/or notoriety for their programs.

The fastest, easiest way to do this is to send a virus directly to a researcher, usually anonymously or, as happened on at least one occasion, by claiming that the sample was 'found'. Documented incidents include the appearance of the TP series, Murphy-2, New Vienna, SVIR and Icelandic-2.

There have been some instances whereby a sample made available for disassembly was *atypical* - it contained additional text strings or replication code was absent. In some of these cases this would only be possible if the virus had been sent directly by its writer or a close associate.

The original Datacrime virus is understood to have been sent directly to a researcher, which may explain its apparent non-existence in the wild. (Not one incident of any of the Datacrime family actually triggering has been reported to *VB* this year, a statistic confirmed by other organisations active in the field.)

This phenomenon, whereby the virus is sent directly to a research specialist is referred to as "cutting out the middle-man". The situation makes virus researchers analogous to lightning conductors and might be construed as beneficial - dangerous code ends up in the hands of those best qualified to deal with it.

Unfortunately, each such sample has to undergo the same painstaking analysis as other malicious software because there is no guarantee that the virus can be isolated - the originator could still release the virus into the wild, if he has not already done so.

Suggestions have been made that the *VB Table of Known IBM PC Viruses* should reflect whether or not entries are likely to be encountered in the real world as opposed to being isolated or laboratory specimens. It would certainly be feasible to indicate the set of forty or so viruses and variants which are currently known to be causing genuine infections. However, to dismiss functioning viruses of dubious origin as being of no consequence would be folly - the research community is simply not in a position to categorise them as such.

TECHNICAL NOTES

High Level Languages

Although most viruses are clearly written in assembly language, it is possible to use a high-level language such as C or Pascal instead. Several such viruses are known today, including AIDS, AIDS-2 and Kamikaze, but their number is low compared to the 250 or so different virus variants written in assembly language. As writing in C or Pascal is undoubtedly easier than writing in assembly language, some explanation must be sought. The High-Level Language viruses (HLL) generally show evidence of being created by programmers with less knowledge than many of the assembly language viruses. The reason may be that the assembly language programmers want their "creations" to reflect their ability -being able to write a program in assembly language indicates a higher level of technical ability than being able to program in Pascal. Another reason is that writing an advanced virus in "pure" C or Pascal is difficult, as many interrupt level calls are required for any function beyond simple replication.

A characteristic of HLL viruses is their size - due to compiler overhead, a HLL virus may be ten times larger than an equivalent assembly language virus. To the virus writer the HLL virus offers one considerable advantage, which partially offsets this obvious drawback: it is difficult to select a usable search pattern for the virus, as any segment of code within the virus could easily be created by the compiler when a totally different program is being compiled.

Overwriting Viruses

As an overwriting virus will cause irreversible damage to any program it infects by writing itself over the beginning of the host program, one might think that this type of virus has the potential to become a serious threat. This is not so.

A virus cannot become a serious problem unless it is able to spread. When a virus is detected, some action is usually undertaken to eliminate it, although this action may be deferred if the virus causes no serious disruption. All other factors being equal, a virus which does not make its presence known will spread faster and more widely than a virus which is detected early or interferes with normal processing. Overwriting viruses cause serious disruption by destroying their host programs which results in immediate detection because infected programs will not run normally, if at all.

'Cooperation'

The 2100 virus from Bulgaria illustrates a new trend - 'cooperation' between different viruses. The phenomena of interaction between viruses is not new - there are already several 'anti-virus' viruses known. The 2100 virus is able to recognise the Anthrax virus, which may indicate that both viruses were written by the same author, probably the person calling himself 'Dark Avenger'.

When the Anthrax virus infects a disk, it will place a copy of itself on the last track of the disk in addition to infecting the boot sector. If the original boot sector is restored by some anti-virus program or utility which does not overwrite the last track, the disk will contain an inactive but functional copy of the virus, even after it has seemingly been disinfected. The 2100 virus (itself a parasitic virus which infects COM and EXE files) is able to locate this "spare" copy of the Anthrax virus and will reactivate its dormant code thus causing the boot sector to become reinfected.

Unintentional Side-Effects

It is well known that several viruses contain malicious code intended to cause damage when some specific trigger condition is met. This damage usually involves formatting the hard disk, deleting programs or corrupting data. What is less well known is the fact that some viruses may cause unintentional damage, sometimes only when certain hardware is present.

A well documented example of unintentional damage is that done by the New Zealand virus which can corrupt the FAT on some hard disks and which overwrites the third sector of the root directory corrupting 1.2 Mbyte 5.25" diskettes with more than 32 files. Another example of unintentional corruption is the "Den Zuk" virus and its variants which format track 40. They may cause a loss of data when infecting a 3.5 inch and/or high-density diskette, where track 40 is already present.

Another side-effect is FAT corruption caused by running CHKDSK or a similar program with a "stealth" type virus active in memory. The virus hides itself, which causes a discrepancy as the number of blocks allocated for the infected files may not match the number of blocks actually required, according to the reported length. CHKDSK will report a FAT error and attempt to correct the problem, corrupting the FAT in the process.

So, even if a virus is reported as "harmless", there is always a possibility that some particular combination of hardware and software may cause a corruption of programs or data.

Lost Property

Several viruses have been reported in the past but never made available to any virus researcher. Some of the viruses mentioned on the next page have appeared in past issues of the *Virus Bulletin* classified as 'Reported Only', but a sample has never been made available. These viruses may never have existed ('vapourware') or their reported existence may have arisen as the result of misinterpretation.They may also have existed at one time, but being very rare or slow to spread have become extinct.

The 'lost' viruses include:

1702: Reported as a 1702 byte variant of Cascade, which is either 1701 or 1704 bytes.

2730: A search pattern for this virus appeared in one of John MacAfee's scanning programs. It does not match any currently known virus.

Agiplan: This virus was described in a West German newspaper nearly two years ago. From the description it seems similar to the Zero-Bug virus. Both add 1536 bytes to the start of infected COM. A search pattern was provided for the virus but no infections have been reported outside of the West German AGIPLAN company which is believed to have ordered all infected files to be destroyed.

Cookie: A virus displaying the message "Gimme a cookie" called the 'Cookie Monster' has often been reported. It is probably a myth.

Gates of Hades: A virus reported to be able to cause physical damage to hard disks.

Hyperspace: Reported to display a special visual effect, followed by the message "Welcome to hyperspace" and a reported increase in processing speed.

Missouri: This was reportedly a boot sector virus. It is generally believed never to have existed.

Nichols: Another "myth" virus

Poem: Reported to display fragments of a poem.

Retro: A "recurring" virus - probably another myth.

Screen: Reported by Ross Greenberg in an article in *BYTE* (June 1989). His copy of the virus seems to be lost, and as no other reports have surfaced, the virus is probably extinct by now. The same article's description of the dBASE virus (also reported by Ross Greenberg) bore little resemblance to the actual example of the dBASE virus which *VB* examined in December 1989.

CIAC Warn of Virus Propagation on Novell

The *Computer Incident Advisory Capability*, *University of California*, USA, issued bulletin number A-33 on September 21, 1990. It warns of a virus threat to MS-DOS system networks. This bulletin provides a possible explanation for Jerusalem-B's apparent ability to replicate on Novell networks, a phenomenon first reported after tests were undertaken by Dr. Jon David at *Novell's* Paramus, New Jersey, facility in June of this year.

CIAC report that file servers on Novell use attribute bits to perform write-protection on stored files. Many viruses will clear these attribute bits before attempting to infect files, thus circumventing the write-protection scheme. If a diskette infected with Jerusalem-B is executed on a Novell network node, the virus will become memory-resident. When the user logs on to the file server (using login.exe), the virus infects this program despite the fact that it is write-protected. Login.exe is a shared program which is executed automatically when users log on to the Novell network. Thus the network configuration enables the Jerusalem-B virus to spread more quickly than if it had spread through the exchange of floppy disks.

CIAC. Tel (USA) 423 4416 Fax (USA) 415 423 0913
E-mail ciac@tiger.11n1.gov

MAC THREATS

MDEF C

A new strain of the MDEF (Garfield) virus has been detected in Ithaca, New York. The virus appears to have been released into the wild just prior to the apprehension of its author by the *New York State Police* (*VB, October, 1990*).

The virus, similar to MDEF, has two characteristic features. First, the system MDEF resource is changed to 6982 when replaced by the virus-infected MDEF 0 resource. Second, in contrast to MDEF (with its 'Garfield' resource), MDEF C adds no characteristic resource name.

MDEF C avoids detection by anti-virus INITS by re-vectoring *ChangeResource* and *AddResource* traps to ROM (thus bypassing RAM resident handlers installed by protection software). The virus may also cause system crashes and other unpredictable behaviour when running under Finder.

Software upgrades are being made available for the major anti-virus packages. Updated releases and user upgrades are summarised below:

Symantec's SAM Version 2.00 traps MDEF C and the following virus definition can be added to previous releases:

Virus Name	MDEF C
Resource Type	MDEF
Resource ID	0
Resource Size	556
Search String	4D4445464267487A005EA9AB
SearchOffset	set 448

Disinfectant Version 2.3 will detect and remove both MDEF C and a new ANTI A variant. Gatekeeper Aid Version 1.1 has also been upgraded to trap MDEF C.

Virus Detective Version 4.0.3 will trap MDEF C with the addition of the following search string:

```
Resource MDEF & ID=0 & WData
4D44#A6616#64546#6A9AB
```

FROM THE FIELD

4K - A Warning of Data Corruption

The recent furore over possible infections by the 4K (Frodo) virus centred on the virus' trigger date of 22nd September. Newspaper, radio and television reports concentrated exclusively on the most superficial aspects of the virus, i.e. its text messages and their reference to J.R.R. Tolkien's *The Lord of the Rings*.

The fact that a bug in the code invariably caused the affected machine to hang when an infected file was executed tended to make most observers dismissive of the problem. As usual, the dire warnings published by some sections of the computer press prior to the trigger date were inaccurate both in their prediction of the number of infected machines and also in the description of the effects of infection.

This virus has been known to researchers for nearly a year now and a preliminary report was published in the May 1990 issue of the *Virus Bulletin* in which mention was made of this virus' capacity to corrupt **data files.** No mention was made of this fact in news coverage and *VB* received no reports from elsewhere that it had been mentioned.

Since the corruption of data files by 4K can cause both immediate and long term problems for affected users, more detailed information concerning the specific effects of data-file corruption has been gathered and is reported here so that future misunderstandings and omissions may be avoided.

File Infection

4K recognises both COM and EXE files by the unusual process of summing the ASCII values of the three characters which comprise the filename extension. If the total value of the extension characters of an uninfected file is 223 (COM) or 226 (EXE) then infection will take place. It should be noted that the individual characters are first AND'ed with 0DF hex to convert lower case characters to upper case.

The total number of possible file extension which fit these criteria has been calculated at 1284 (including inversions and rotations) and several of them have been noted as common data file extensions. Among these are OLD, MEM, PIF and QLB which total 223, and DWG, LOG and TBL which total 226.

The distinction between the two sets is important since the virus necessarily distinguishes between the techniques necessary to infect COM or EXE files and the amount of corruption to the original file contents will be greater for EXE (sum 226) type files.

The Effects

4K is a "stealth" virus and contains code which misinforms DOS about the contents and length of infected files. This means that while the virus is resident and active in system memory, infected files will appear "clean" to the operating system. This also means that such files will similarly appear "clean" to any program using DOS services.

Corrupted data files, if copied to backup disks or tapes will carry the virus with them. The effect of this is remarkably similar to the dBASE virus which deliberately sets out to corrupt data files (*VB, December 1989*).

While the virus is resident, all files will appear clean and application programs will function normally. However, when the virus is removed (by replacing all infected program files), application programs will "see" the corruption introduced by the virus and the effects will be unpredictable.

In one incident involving DWG (drawing) files, the application program aborted with an error when a corrupted data file was encountered on a clean system - although the program functioned normally when the machine was re-infected for test purposes.

In this instance (EXE type infection), various bytes within what *would* have been the EXE header were altered by the virus and the 4K of virus code was appended to the end of the file. Since this first section of the file contained vital header information, an error was encountered as soon as an infected file was accessed and the application program aborted. EXE infection from the 4K virus usually changes five fields within the EXE header as follows:

```
WORD at offset 04H = Number of Pages

WORD at offset 0EH = Stack Segment value

WORD at offset 10H = Stack Pointer value

WORD at offset 14H = Instruction Pointer value

WORD at offset 16H = Code Segment value
```

The Cure

The original contents of these fields can be recovered from the beginning of the appended section of virus code.

On EXE type files where the sum of the extension characters is 226, the original 28-byte header is stored at the beginning of the virus code and may be identified by comparison with the unmodified fields of the header.

The actual storage position will always be 4 bytes beyond a paragraph boundary (i.e. divisible by 16) and will be near the start of the last 4096 bytes of the infected file.

Removing the virus code can easily be accomplished by replacing the original header and truncating the file by exactly 4096 bytes.

A similar pattern is used for the infection of COM type files where the sum of the extension characters is 223 but in this case, only the first six bytes of the file are altered and need to be recovered (although the original 28 bytes are saved exactly as in EXE infection).

Obviously the effects of such data corruption are unpredictable and will depend upon the type of information and how the relevant application program accesses it. The DWG files mentioned above contained graphic data and this may be particularly sensitive to this type of corruption.

The extensions mentioned above were selected simply because they are extremely common and may produce strange effects on recently disinfected systems.

Among the COM type extensions:

* MEM is used by dBASE programs as variable storage and may produce errors which could be difficult to trace.

* PIF files are used by various versions of *Microsoft* WINDOWS and again, the effects will vary and could be intermittent.

* QLB extensions are used by *Microsoft's* QuickBASIC environment libraries and infection here will usually produce immediate errors when the programming environment is invoked with an infected library file.

* OLD is an extension used by many packages and along with QLB it presents a special problem.

Corrupted Backups

The fact that corrupted data files may exist on backup disks may be a problem when data integrity is paramount but the virus code is unlikely to find its way back into the processing stream and thus become "live" again.

However, when considering the OLD and QLB extensions (and possibly others), there is a distinct chance of this "dormant" code being reactivated The QLB files for example are actually in EXE format (with the familiar MZ header word) but they do contain executable code which remains untouched when the COM type infection of 4K is introduced. Thus under certain circumstances, invocation of QuickBASIC's environment may execute the virus code and re-install it in memory to begin the replication process all over again.

The OLD extension presents even more risk since there are several program optimisation utilities which rename an original program file with an OLD extension prior to generating a modified file.

One of the best known to do this is the LZEXE file packing program (*see VB, June 1990*). LZEXE is an excellent utility in widespread use which produces significant reductions in size when applied to ordinary EXE files. The reduction is done by creating a self-extracting archive of the original program file which unpacks itself in memory when it is run. The original (unpacked) version of the file is renamed with an OLD extension and the danger with 4K infection is that when the packed file becomes infected, a user might delete it and rename the OLD file back to EXE before either using it or repacking it. In spite of the fact that files with an OLD extension are infected as COM files, renaming an infected one as EXE and trying to run it **will** re-infect a system.

With infections caused by the 4K virus, it is therefore plain that **all** files should be checked for infection and replaced with clean copies if possible.

Since backups may be corrupted, a good, reliable, disinfection program is a must to recover damaged data files During tests to confirm some of the effects described in this article, we had occasion to try four 4K disinfection programs; two of them did not disinfect the target file correctly and errors occurred even after the virus code was removed. The relevant vendors have been informed of the problem.

Data Corruption

The corruption of data files reported here is almost certainly an unintentional side-effect resulting from the unusual technique which the 4K virus employs to recognise COM and EXE files. In effect, the virus attempts to 'infect' data files believing them to be executable program files. The affected data files subsequently become corrupted. **Note: the virus cannot propagate by appending itself to pure data files. For a virus to spread it must have an executable path.**

There are only a handful of viruses which intentionally attack pure data, the most significant of these being the dBASE virus which attacks .DBF files and randomly transposes bytes as the corresponding letters are entered at the keyboard. The user will be oblivious to this action as the corruption of the data being entered is not shown on the screen, nor does it appear if corrupted data backups are restored onto an infected processor. The corruption can only be seen if the data is viewed in a clean DOS environment. A detailed description of this virus was published in *VB*, December 1989.

Corruption of data in this way poses a new threat to the validity of backups to recover from computer virus induced damage and emphasises the essential requirement to test backups regularly and thoroughly.

FOR MANAGEMENT

This is the first of a series of articles which examine different aspects of microcomputer security. Here, the inherent weakness of the PC operating system is highlighted and some options to confound a determined intruder are examined.

PC Security Part I. Controlling The Operating System

PCs are inherently insecure; by default, MS-DOS and PC-DOS grant absolute authority to anyone who switches the microcomputer on. Most microcomputers incorporate no hardware mechanisms and no privileged instructions to isolate users from sensitive system processes. In effect, a microcomputer running under DOS provides the user (regardless of his identity) with total system privileges. The designers of the personal computer and its operating systems never accounted for security - microcomputers are intended to be ‘user friendly’ which, in turn, can make them ‘attacker friendly’.

The microcomputer operating system is insecure, a fact which profoundly undermines any concept of ‘PC security’. With current processor designs and operating systems, security administrators should appreciate an inevitable and constant state of ‘*PC INsecurity*’ and work to reduce associated risks in a cost-effective manner (*see Figure 1.*).

There are three fundamental problems associated with microcomputers, namely:

1) the hardware, operating systems, programs and data are insecure;

2) personal computer equipment (including peripherals) is accessible and easily removable, and;

3) sensitive, confidential and even classified data is increasingly processed using this equipment.

Reducing System Insecurity

As described above, the critical concern is with system security. Many ‘personal’ computers in business use are wholly ‘impersonal’. With multiple users accessing standalone and networked PCs, there is a need both to authenticate the identity of each individual user and to enforce segregation (i.e. to impose user ‘rights’). Otherwise, every user is permitted ‘world’ rights and anarchy (a state without government) prevails. The process of imposing control over users is called logical access control. There are a number of access control products on the market, either in software, hardware or a combination of the two.

Before discussing logical access control, it is important that a fundamental tenet pertaining to current PC architecture is understood:

Without integral hardware enforcement, it is impossible to prevent a determined attacker from accessing or modifying parts of the operating system and thus circumventing intended security mechanisms. Software access control, in particular, will be readily circumvented by a determined, technically competent attacker. In a secure system the computer’s CPU must not access the processor controlling the security mechanisms other than through a single controlled I/O port. This cannot be done with software because the security program has to run on the same CPU upon which it is attempting to impose controls.

PC Access Control

Access control is the technique for preventing an intruder from accessing computer resources. In its simplest form (called physical access control) this is done by means of locked doors and a lock on the computer itself. On mainframes, which have traditionally had a number of users, access control evolved as an integral component of the operating system. On personal computers, the demand for access control has been an afterthought. Access control on mainframes is usually well regulated and ranges from physical access to the terminals to the strict enforcement of user names and passwords.

● DATA	❑ CORRUPTION	❑ DELETION	❑ UNAUTHORISED DISCLOSURE
● PROGRAMS	❑ CORRUPTION	❑ DELETION	❑ UNAUTHORISED EXECUTION
● PROCESSOR	❑ MALFUNCTION	❑ THEFT	❑ DAMAGE/DESTRUCTION
● MEDIA	❑ MALFUNCTION	❑ LOSS/THEFT	❑ DAMAGE/DESTRUCTION
● PERIPHERALS	❑ MALFUNCTION	❑ THEFT	❑ DAMAGE/DESTRUCTION

Figure 1. Microcomputer vulnerabilities

The architecture of most microcomputers does not support privileged and unprivileged instructions, nor ownership of different areas of memory. This makes it impossible to design provably secure access control systems.

Attempts have been made by numerous PC expansion board manufacturers to provide secure hardware access control. Unfortunately, it appears unlikely (with current processors) that provably secure logical access control for the PC will emerge. Expansion boards can offer adequate security provided they are not removed from the the PC's expansion board rack. Unless carefully implemented, such 'add-on' security features can also interfere with network operation, other expansion boards, and software execution.

While well designed access control is better than none, a determined intruder will not be prevented by any PC access control product. Hardware access control is better than software, but both methods are inherently insecure. Having said this, logical access control is invaluable for imposing control over general PC use, provided that users comply with certain rules - the most important of which pertain to the use of passwords. Also, potential users of such products should consider the likelihood of a determined attack upon the system (which will often be negligible) in any risk assessment.

Some points in selecting an access control product include:

* The system must prevent bootstrapping from a system floppy disk. If it does not, an unauthorised user can bypass the security mechanisms completely.

* Each user should be identified by a combination of a user ID and a password. Passwords should be entered in half-duplex (i.e. they must not appear on screen when entered). The user ID and password must be accepted or rejected only in combination, otherwise an attacker can establish whether or not a user ID is valid. The system should enforce a minimum password length (6 to 8 characters) and the maximum number of characters allowed should be extensive enough to enter a series of words. Passwords should have expiry dates with periodic, mandatory changes to the password imposed by the system on the user. The system should prevent re-use of a password, which, obviously entails the storage of redundant passwords.

* The system should automatically deny access if successive (usually 3) incorrect attempts are made to enter the user ID/ password combination.

* The clock providing time and date should be separate from the DOS clock. If this is not so, packages which allow access only at certain times of day are useless, as the attacker can change the DOS clock setting. The provision of a separate clock is, of course, only possibly in hardware.

* A user should be assigned specific times of the week during which he/she is allowed to log-on. This prevents night-time browsing by 'impersonators' with a legitimate password.

* A user should have defined privileges (*read, write, delete, execute*) as to the directories and programs which he/she can access. Privileges should be defined and granted by the security manager.

* An audit trail should be kept of which users have logged on and when. The system log file must be secured against deletion or modification by an attacker; it should thus be stored in encrypted form. A comprehensive audit trail would record system start-up (time/date/user account), session initiation (log-in-time/log-out-time), program initiation and termination (program name and run times) and access (data file name(s)). This information can be used to reconstruct, review or examine system abuse.

* The user should be allowed to 'lock' the computer by using a password. The screen should be blanked when this sequence is entered. This enables the user to leave the processor active but unattended. Equally a single key should be available to blank the screen in order to prevent onlookers from seeing confidential information.

* A keyboard inactivity monitor should be provided. If nothing has been typed for a predetermined time (10 minutes or so), the PC automatically logs out.

* The security manager should be able to create new accounts, change privileges etc. User passwords should not be chosen by the security manager; the users should be responsible for choosing passwords.

* Automatic encryption of sensitive files and passwords should be provided. This should use a recognised provably secure encryption algorithm such as DES. Faster, 'DES-alike' algorithms may not be as secure. Unpublished proprietary algorithms are often trivial and should not be trusted unless technical details (which can be assessed for security) are forthcoming.

* It is important that any access control package does not interfere with hardware or software on the PC. Access control packages often reach 'deep' into the operating system and unforeseen complications can arise.

The most important point to remember is that PC access control software and/or hardware is reliant on the user's compliance with the system and its procedures.

There is nothing to stop a legitimate user from revealing (intentionally or otherwise) his user ID and password, thus enabling an impersonator to gain access to the system. To a lesser extent, this vulnerability of transferring access applies to relatively sophisticated access control measures using tokens (smart cards, swipe cards etc.) which rely on something 'owned' as well as something 'known'. Pub gossip, written passwords stuck to the keyboard, ill chosen passwords and even wilful *complicity* with an outside attacker are all threats to such systems. (It should be pointed out that complicity with an attacker will be the result of inadequate *personnel* security. This re-emphasises the truism that the proverbial 'chain' is only as strong as its weakest link.)

Biometric access control (something which the user 'is'), in the form of fingerprint and retina scanning (among other techniques), provides substantially increased security over simple passwords or token based systems because certain physical attributes are impossible to transfer or duplicate. Having said this, biometric methods are expensive and are rarely found in the PC environment.

A widely held belief pertaining to access control products is that they can provide guaranteed protection from computer viruses. This is a highly contentious issue. The basic function of access control is to restrict the system to a set number of users and further restrict those users to certain actions which may include inhibiting the import or execution of software.

Despite the inevitable shortcomings of logical access control it is recommended as a way of imposing control over multi-user PCs. Audit trails, in particular, are an effective way of making each user accountable for his/her actions at any given time which in turn tends to increase vigilance among a user group.

Secure Erasure

Insecure file deletion is the single most widespread risk to confidential data.

The DOS command 'DEL' only deletes the first character of the target file's *name* from the directory but does not remove its *contents*: it simply marks the clusters in which the deleted material resides as 'free' in the FAT. (The same is true of the Unix command 'RM'). DOS cannot invoke data or programs which have been deleted using the DEL command.

Many multi-tasking systems and even normal word-processing packages use temporary areas on disk which are simply abandoned when the program finishes executing. While not accessible using systems commands this data residue can be retrieved using disk utility programs such as Norton or PC Tools. This is obviously a threat to confidential data.

Positive erasure of a file can be achieved using specialist programs which change the polarity of each storage bit to be erased. However, laboratory studies have shown that it is possible to read information which has been overwritten once due to physical effects such as residual magnetism. Satisfactory security can thus only be achieved by positive erasure programs which can perform *multiple* overwrites. Positive erasure programs should also enable visual verification that overwriting has occurred.

Encryption

A contrasting approach to ensuring data confidentiality involves the use of encryption which is designed to deny effective use of disclosed information.

Good encryption, used properly, is the most secure way to protect confidential data against unauthorised disclosure. Encryption offers a more fundamental form of protection than access control systems which can be relatively simple for computer specialists to defeat. Encryption has an additional advantage of safeguarding confidential data from disclosure on *any* magnetic media, whether it is being processed, in storage or transit.

Cracking a properly implemented encryption code is an immense task, governed by powerful laws of mathematical intractability. Modern encryption methods are designed in such a way that it is not possible, even knowing the precise method used, to trace back the manipulations which have taken place and thus simply 'unravel' the encryption. This is because the basic tool, the encryption algorithm, works in conjunction with an unpredictable key.

Encryption products can protect against unauthorised disclosure and detect corruption or intentional modification to data stored on disk. Changes to encrypted data (cyphertext) implemented without the correct key will result in corrupted plaintext. However, encryption cannot prevent either data modification or destruction and critical data cannot be protected by encryption alone. Its function is to guarantee confidentiality and reveal corruption/modification should it occur.

Most packages enable the user to enter and change keys and encrypt and decrypt data, normally in the form of entire files. There are a variety of software and hardware encryption products available. Standard encryption involves the user preparing a file, running the encryption program (using a key known only to him) and producing unintelligible cyphertext. The original file should be positively overwritten - good encryption packages will do this automatically. The same key can be used to decrypt the file. Provided that these programs are designed for incorporation into batch files, file encryption is a highly suitable solution for professional use.

Other methods utilise bulk-file encryption; as data is written to disk it is encrypted and as it is read from disk it is decrypted and passed to the requesting program. Bulk file encryption usually necessitates the use of hardware devices which can speed the encrypt/decrypt process by a factor of between 10 and 100 over software implementations. Bulk file encryption often proves intolerably slow: some packages use 'fast algorithms' but their security is usually inferior to that provided by the Data Encryption Standard (DES). The choice between software and hardware implementation is dependent on individual specifications and requirements.

The problem with encryption is that it is critically dependent upon the reliability and integrity of the user.

Encrypted data is absolutely safe only if the key used to encrypt the data is generated in a secure way and is in the right 'trusted' hands. **All security is lost if the key is disclosed to an unauthorised person**.

The task of choosing, distributing and changing keys is known as **key management**. An encryption key is transferrable in much the same way as a password, which raises the spectre of an attacker impersonating a legitimate user. However, data stored on disk (whether hard disk or diskette) which has been encrypted using a secure tested algorithm (DES or RSA) is secure even in the face of a concerted attack (provided, of course, that the key is not disclosed). Products containing proprietary encryption algorithms should be regarded with caution: the algorithms used are often trivially weak.

Conclusions

High security software options on the PC are limited and costly in processing time. Embedded hardware can implement the necessary controls without detrimentally affecting the computer's processing and storage capacity.

Despite the inherent insecurity of the MS-DOS/PC-DOS operating system, security management can impose sufficient control over microcomputer systems for use in all but the most hostile environments.

It is important that the limitations of both access control and encryption are understood. Both techniques are designed to ensure *confidentiality*; they do not provide guaranteed protection against program or data *corruption* although both methods serve to reduce these risks.

Next Month

Cost-effective and proven methods to protect data and processing equipment will be examined. Computer security is often described as a balancing act between ***c**onfidentiality*, ***i**ntegrity* and ***a**vailability* (the acronym is, of course, CIA). In fact, data integrity and availability is the crucial concern to most commercial organisations. Enhancing PC security and safeguarding data need not be expensive and is as much a matter of common sense as installing 'high-tech wizardry'.

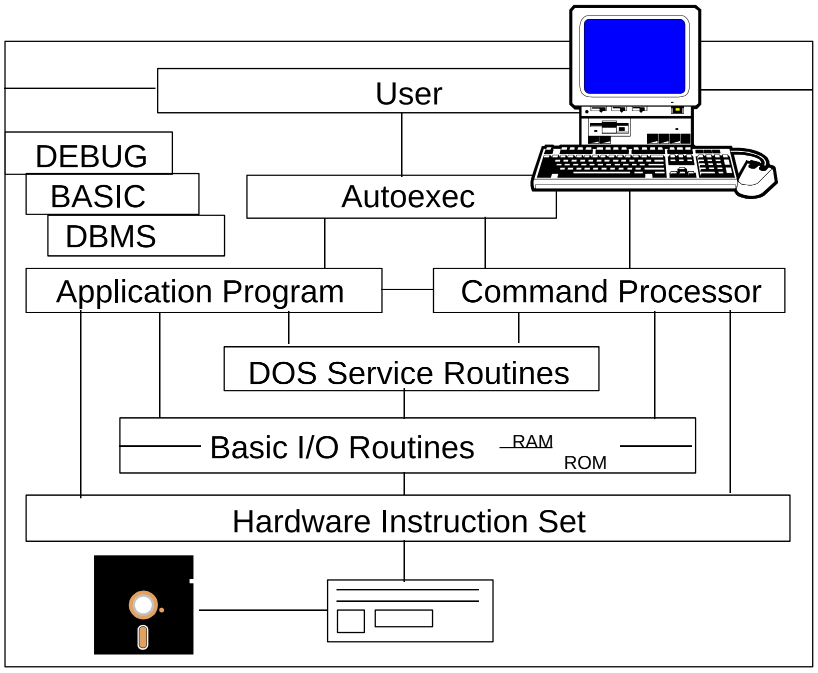

Figure 2. The many internal interfaces of the personal computer.

Securing the system depends upon controlling the paths by which users access system and program functions.

Controls implemented at any level can be circumvented by the determined intruder following an alternative path to his objective.

Although, it takes a degree of technical competence to exploit the inherent weakness of PC operating systems, many experienced users acquire the necessary knowledge to do so.

(Diagram reproduced from Data & Computer Security: Dictionary of Standards, Concepts and Terms, 1987, D. Longley and M. Shain, by kind permission of Macmillan Scientific Publishers Ltd.)

KNOWN IBM PC VIRUSES (UPDATES)

Updates and amendments to the *Virus Bulletin Table of Known IBM PC Viruses* as of October 26, 1990. The full table was last published in August 1990.

Entries consist of the virus' name, its aliases, if any, and the virus type. This is followed by a short description (if available) and a 10 to 16 byte hexadecimal pattern to detect the presence of the virus using the 'search' routine of disk utility programs or, preferably, by adding the pattern to the library of a virus scanning program. Offset (in hexadecimal) normally means the number of bytes from the virus entry point to the location at which the pattern commences.

Type Codes

C = Infects COM files	**D** = Infects DOS Boot Sector (logical sector 0 in each DOS partition)
E = Infects EXE files	**M** = Infects Master Boot Sector (track 0, head 0, sector 1 on disk)
N = Not memory-resident after infection	**R** = Memory-resident after infection
P = Companion virus	

Seen Viruses

1024-B - CER: A minor variant of the Bulgarian 1024 virus, detectable by the string published in *VB*, September 1990.

1226 - CR: This Bulgarian virus is related to Phoenix, Proud and Evil. As in the case of its relatives, no search pattern is possible. (See article on virus encryption techniques, pp.13-16.)

2100 - CER: This is a Bulgarian virus, related to the Eddie and Eddie-2 viruses and contains extensive segments of code common to both. The previously published pattern for Eddie-2 (*VB, August 1990*) can be found within this virus, but they can be easily differentiated on the basis of length.

Amstrad-852 - CN: Basically identical to the original 847 byte variant, only a text string has been changed. The Amstrad pattern in *VB*, August 1990, can be used to detect this variant.

Anthrax - CEMR: A multi-partite virus from Bulgaria, which infects the Master Boot Sector, as well as executable files. Infected files usually grow by 1000-1200 bytes.

```
Anthrax          0E1F 832E 1304 02CD 12B1 06D3 E08E C0BF ; Offset 0 in MBS
```

Anti-Pascal - CN: This is a family of 5 Bulgarian viruses which overwrite or delete .PAS or .BAK files, should they find no .COM files to infect. All five viruses are rare, even in Bulgaria, and fairly simple in structure. The length of the variants is in the range 400-605 bytes.

```
Anti-Pascal (1)  D1E0 D1E0 80E4 0380 C402 8AC4 8BD8 32FF ; Offset variable
Anti-Pascal (2)  21BE 0001 5A58 FFE6 50B4 0E8A D0CD 2158 ; Offset variable
```

Dir - CR: A 691 byte Bulgarian virus which only infects files when the DIR command is issued. No other effects have been found.

```
Dir              CD26 0E1F 580E 1FBE 0001 56C3 0E0E 1F07 ; Offset 04A
```

Evil - CR: This is a close relative of the Phoenix virus, but is shorter, 1701 bytes instead of 1704. It uses the same encryption method, which renders the extraction of a search pattern impossible. (See article on virus encryption techniques, pp.13-16.)

Internal, 1381 - EN: Infective length is 1381 bytes. Virus contains the string:

```
            INTERNAL ERROR 02CH.
            PLEASE CONTACT YOUR HARDWARE MANUFACTURER IMMEDIATELY !
            DO NOT FORGET TO REPORT THE ERROR CODE !
```

```
Internal         1E06 8CC8 8ED8 B840 008E C0FC E858 0480 ; Offset 0B1
```

Kamikaze - EN: This overwriting virus from Bulgaria is written in Turbo Pascal and is fairly large at 4031 bytes. Like other similar viruses it is not a serious threat (see technical note on overwriting viruses, p. 3).

Seen Viruses (contd.)

Korea - DR: A minor variant of the Korea virus probably compiledon a different assembler. It is not detected by the search pattern in *VB*, August 1990. The amended search pattern detects both known versions of the virus.

```
Korea           CO8E D88E D0BC F0FF FBBB 1304 8B07 4848 ; Offset 009
```

Microbes - DR: An Indian virus whose effects are not fully known, except that booting from an infected disk has been reported to cause some computers to “hang”.

```
Microbes        042D 0400 A313 04B1 06D3 E08E C006 C706 ; Offset 014
```

MG - CR: A simple, 500 byte Bulgarian virus.

```
MG              AA1F 1E07 585E 1EBB 0001 53CB 3D04 4B74 ; Offset 086
```

Nomenklatura - CER: Infective length is 1024 bytes and only files longer than 1024 bytes are infected. The virus infects on executing a program or opening a file which means that a virus scanning program will infect all files on the system if the virus is in memory. The virus seems to have no side-effects.

```
Nomenklatura    B8AA 4BCD 2173 785E 5606 33CO 8ED8 C41E ; Offset 2DD
```

Proud - CR: This 1302 byte virus is a member of a Bulgarian family of 4 viruses, which also includes 1226, Evil and Phoenix. As they all use the same encryption method, no search pattern is possible. (See article on encryption, pp. 13-16.)

Tiny Family - CR: This is a family of at least 10 Bulgarian viruses, which includes the shortest viruses now known. The viruses are not related to the Danish ‘Tiny’ virus, but just like it they do nothing but replicate. The length of the variants is from 198 down to 134 bytes.

```
Tiny Family (1) CD32 B43E CD32 071F 5F5A 595B 582E FF2E ; Offset variable
Tiny Family (2) 2687 85E0 FEAB E3F7 931E 07C3 3D00 4B75 ; Offset variable
```

Trackswap - DR: A small Bulgarian boot sector virus which is awaiting analysis.

```
Trackswap       FBA1 1304 48A3 1304 B106 D3E0 8EC0 06BD ; Offset 00E
```

VFSI - CN: A simple 437 byte Bulgarian virus.

```
VFSI            100E 1FB8 001A BA81 00CD 21BE 0001 FFE6 ; Offset 1A3
```

Reported Only

1605 - CER: This virus is reported to be related to the Jerusalem virus and to cause a slowdown of the system.

Black Monday - CER: This virus was reported in Fiji. It is 1055 bytes long, and contains the string “Black Monday 2/3/90 KV KL MAL”.

Christmas - CN: A 600 byte virus from Japan or Taiwan which will display the message “Merry Christmas to You!” on 25th December. Reported to be targeted at NEC PC-9800 computers. Programs greater than 30,720 bytes are destroyed.

Invader - DCER: This Taiwanese multi-partite virus is reported to be related to the Plastique virus. It will play a melody 30 minutes after activation.

Number One - CN: A primitive virus written three years ago and published in Burger’s *Computer Viruses: A High Tech Disease.*

Rat - ER: This Bulgarian virus has been reported, but the sample which is available for analysis does not replicate.

Saddam - This virus has been reported in Israel.

Scott’s Valley - CER: A 2131 byte virus, first reported in California.

Terror - CER: This Bulgarian virus has not been analysed yet, as it failed to replicate under testing conditions.

V2P2, V2P6, V2P6Z - CN: These three viruses are reportedly written by Mark Washburn, who is also the author of the 1260 virus, which is less complex than these three.

Westwood - CER: This is reported to be the Jerusalem virus, substantially altered in order to prevent it from being detected by anti-virus programs.

Wisconsin - CN: An 825 byte, .COM-infecting virus, which may delete .PAS files when it activates.

FOR PROGRAMMERS

Fridrik Skulason

Virus Encryption Techniques

Several of the viruses known today use encryption and this feature is likely to become even more prevalent in the future. An encrypted virus consists of two parts, a short decryption program and the encrypted main body of the virus. When an infected program is run, the decryption routine executes first. It decrypts the virus, possibly performing other tasks at the same time, such as computing a checksum for the decrypted code to check whether the virus has been tampered with.

Encryption is often used when sensitive or confidential information is transmitted or stored, but virus writers use it for different reasons, four of which are described here.

1 - Encryption - To Prevent Static Code Analysis

Static analysis of programs basically involves disassembling them and examining the disassembled code for suspicious instructions or blocks of code. Examples of suspicious code include statements such as:

JMP F000:XXXX	transfer control directly to the ROM BIOS
INT 26H	absolute disk write, bypassing the file system.

Several programs have been written which attempt to analyse code automatically and produce a warning when suspicious instructions are detected. With a proper definition of "suspicious instructions" they would indeed be able to detect most viruses and Trojans. The problem with this type of program, however, is the unacceptably large number of false positives and negatives they generate. A false positive occurs when a legitimate program happens to use one or more of the "suspicious" instructions. This may happen in the case of utility programs, such as the Norton Utilities, but also when cache programs, operating system extensions and other similar software are examined. A false negative, where a malicious program is given a clean "bill of health", is more serious. This happens because the author carefully disguised the suspicious instructions. Encryption is one of the methods used to disguise suspicious instructions and its presence in virus code often prohibits the use of static analysis programs.

2 - Encryption - To Prolong the Process of Dissection

Encryption makes analysis of the virus code more difficult, but it usually does not add more than a few minutes to the time required to analyse the virus. There is one notable exception to this - the Whale virus, which is described on page 17-20, where most of the code is dedicated to encryption in a convoluted attempt to confound disassembly. The term 'armour' describes code designed to confound disassembly.

3 - Encryption - To Prevent Tampering

It is common for new virus variants to arise as a result of minor changes to the original virus. The best example of this is the Payday virus - it was produced by changing only a single bit. Encrypting a virus makes it more difficult to change in this way, as anyone planning to modify the virus must first decrypt it, then make any necessary changes and re-encrypt it before reassembling it.

4 - Encryption - To Evade Detection

In the first encrypted viruses such as Cascade, the decryption code was identical in all files infected with the virus. A search pattern could therefore be selected from the decryption routine. Recent viruses make this impossible, by using self-modifying encryption, so that no two samples of the same virus have any usable hexadecimal search string in common.

A Note on Encryption

For any virus to function, regardless of encryption techniques, it must decrypt itself before processing. This means that any encryption technique used in virus code can always be broken regardless of the sophistication of the encryption algorithm used. This fact may explain why virus writers choose to adopt such trivial encryption methods - time expended on devising more sophisticated algorithms is effectively wasted.

The Encrypted Viruses

Having listed the various reasons for using encryption, we can now turn our attention to the encrypted viruses known today and describe them. Viruses generally use simple encryption algorithms, which are easily reversible. A standard method employs the XOR operation, where encryption and decryption are performed by the same program code.

Pretoria

The Pretoria (16th June) virus uses a very simple encryption algorithm, which consists of XOR-ing each byte with a fixed value. This algorithm is equivalent to a simple substitution algorithm. It is also easily reversible, as XOR-ing the encrypted byte with the same value as was used when encrypting will yield the original value.

```
again: lodsb                 ; get a byte to decrypt
       xor    al,0a5h        ; decrypt using key
       stosb                 ; and store it back
       dec    bx             ; finished ?
       jnz    again          ; if not, continue..
```

July 13th

This Spanish virus uses a method very similar to that used by the Pretoria virus, XOR-ing the virus code with a single value. As an example of this method, the decryption procedure is included here. It is identical in all infected files and long enough for a search pattern to be extracted from it.

```
         mov      al;cs:[label]              ; get the value used to encrypt...
         xor      al,90h                     ; ...this particular sample
         mov      si,offset label            ; where to start the decryption
         mov      cx,length_of_virus_body    ; number of bytes to translate
again:   xor      cs:[si],al                 ; perform the actual decoding
         inc      si                         ; and continue
         loop     again                      ; until all bytes have been decrypted
label:   db       5bh                        ; this will then be a NOP instruction
```

Slow

The Slow virus uses a similar method, XOR-ing each byte with a fixed value, which is changed in each infected file.

```
         mov      cx,length_of_virus_body    ; get length
again:   xor      [si],1bh                   ; decrypt one byte
         inc si                              ; increase pointer
         loop     again                      ; until all bytes have been decrypted
```

Cascade

Cascade was the first virus to use encryption. The algorithm used is somewhat more sophisticated than the simple substitution algorithm, as it consists of XOR-ing each byte twice with variable values, one of which depends on the length of the host program. Even though the decryption routine is short, a search pattern can comfortably be extracted from it.

```
         lea      si,[bx+start_of_virus]     ; where to start
         mov      sp,length_of_virus_body    ; SP used as counter to make tracing difficult
again:   xor      [si],si                    ; XOR with counter variable 1
         xor      [si],sp                    ; XOR with counter variable 2
         inc      si                         ; increase one counter
         dec      sp                         ; and decrease the other one
         jnz      again                      ; until all bytes have been decrypted.
```

Datacrime II

As mentioned in the August 1990 edition, the Datacrime II virus uses encryption. This virus was also the first to use self-modifying encryption, in the sense that the decryption/encryption routine modifies itself. This does not affect the extraction of a suitable search pattern. In fact, the purpose of this modification is to prevent tracing through the decryption process using DEBUG or a similar utility program. It could thus be described as an ‘armoured’ feature to prevent disassembly. The encryption method which is used is otherwise quite simple - each byte is XOR-ed with a key which is rotated by one bit each time.

```
again:   mov      al,cs:[bx]                 ; get next byte to be decrypted
         mov      cs:[di],22h                ; change the next instruction from xor al,dl to and al,dl
         xor      al,dl                      ; perform the decryption
         ror      dl,1                       ; rotate the key
         mov      cs:[bx],al                 ; store the decrypted byte
         inc      bx                         ; increment counter
         mov      cs:[di],32h                ; change the instruction back to an xor instruction.
         loop     again                      ; until all bytes have been decrypted
```

If the code is traced using DEBUG it will not work as intended, as the XOR instruction is changed to an AND instruction before it is executed. However, if the code is run normally, it will work, because when the instruction is changed it has already been fetched and the XOR instruction is waiting in the pipeline even if it has at that moment been replaced by an AND instruction in memory.

The 800 Virus

The 800 virus uses a simple encryption method, just XOR-ing with a constant value. However, one detail is unusual - the value is obtained by XOR-ing together all words in the virus body. The decryption routine is long enough to provide a usable search pattern.

```
         mov     di,si                     ; start of virus body
         xor     dx,dx                     ; set key to zero
         mov     cx,length_of_virus_body   ; length of virus (in words)
         push    cx                        ; and store it
again1:  lodsw                             ; get one word of data
         xor     dx,ax                     ; and compute a key
         loop    again1
         pop     cx                        ; recover length of virus
again2:  xor     [di],dx                   ; decrypt a word
         inc     di                        ; increment counter by 2
         inc     di                        ; (the size of a word)
         loop    again2                    ; until all words have been decrypted
```

Syslock

The encryption used by the Syslock virus is just a minor variation of the previous encryption methods. Like all the previous viruses described so far, the encryption procedure is of sufficient length for the extraction of a usable search pattern. Note that this works only when the number of words in the virus is even.

```
         mov     si,start_of_virus         ; get start of virus
         mov     cx,length_of_virus_body   ; and the length
         nop
         shr     cx,1                      ; convert from bytes to words
again:   mov     ah,cl                     ; get a key
         mov     al,cl                     ; which changes with every byte
         xor     ax,[key]                  ; and a constant
         xor     [si],ax                   ; and decrypt it
         inc     si                        ; increment counter
         inc     si
         loop    again                     ; until all bytes have been decrypted
```

1260 and Casper

The encryption routine used by these two viruses was described in the March 1990 edition of*VB*. As It represents a significant development in encryption techniques, most of that previously published information is repeated here.

The encryption algorithm itself is very similar to the other methods described in this article, but it contains the following instructions:

```
         mov     ax,key_1                  ; get the initial first encryption key
         mov     cx,key_2                  ; and the second one
         mov     di,start_of_virus         ; where to begin
label:   xor     [di],cx                   ; decrypt - step 1
         xor     [di],ax                   ; step 2
         inc     di                        ; point to next byte
         inc     ax                        ; increment the first key
         loop    label                     ; until all bytes have been decrypted
```

The significant advance consists of inserting various one-byte and two-byte non-functioning instructions between the instructions listed above. This ‘garbage’ has no effect on the decryption process, but makes the extraction of a search pattern impossible. The longest sequence present in all infected files contains just three bytes, which is far too short to be usable. To detect the virus some additional information must be used, such as that the virus only infects .COM files and infected files start with a JMP to a location 1260 (or 1200 in the case of Casper) bytes before the end of the file.

The 1260 virus uses an additional level of encryption as well, as described in the August 1990 edition.

The author of the 1260 virus, Mark Washburn of Minnesota, USA, is reported to have written several other viruses, which use even more complex encryption methods. In fact, the encryption method used by 1260 is just one of the possible methods which might be used by the simplest of his other viruses. Washburn has reportedly written three such samples, V2P2, V2P6 and V2P6Z.

Suomi

The Suomi virus also uses self-modifying encryption, but it is not as advanced as that used in 1260. The major difference is that additional instructions are not inserted at random, but only at fixed places, indicated by the question marks in the following code fragment. This makes the use of search pattern possible, provided that it is permitted to contain “wild-card” characters.

```
again:  xor      [bx+di],ax                  ; decrypt a word
        jmp      label                       ; skip over random data
        nop                                  ; a "padding" NOP inserted by MASM
        ?                                    ; a variable "random" byte
        ?                                    ; a variable "random" byte
label:  sub      di,2                        ; point to next word
        jnb      again                       ; until all words are decrypted.
```

Proud, 1226, Evil and Phoenix

This family of four viruses from Bulgaria uses a new method for preventing the selection of a search pattern. No instructions are added, but instead the instructions themselves change. The viruses all use the same decryption method, which is closely related to the one used by the 800 virus.

```
        mov      r2,r1                       ; beginning of virus code
        xor      r3,r3                       ; zero key
        mov      r4,length_of_virus          ; length of virus
        push     r4                          ; and store it for later use
again1: xor      r3,[r1+22h]                 ; obtain one word
        inc      r1                          ; and point to the next one
        inc r1
        dec r4
        jns/jge  again1                      ; until code has been xor-ed together
        pop      r5                          ; restore length
again2: xor      [r2+22h],r3                 ; decrypt one word
        inc      r2                          ; and point to the next one
        inc      r2
        dec      r5
        jns/jge  again2                      ; until all words have been decrypted
```

Here r1,r2, r3, r4, and r5 stand for ax,bx,cx,dx,si or di, the selection of which register is used and where varies from one virus to another. The conditional jumps can also be coded in more than one way.

Whale

The WHALE virus makes extensive use of encryption. Some parts of the code lie buried under several layers of encryption, and a number of different encryption methods are used. At the top layer, the encryption depends on the particular mutation of the virus. The methods include XOR-ing every byte, XOR-ing two bytes out of every three, adding a particular value to each byte, complementing each byte and so on. Underneath this layer there is extensive use of XOR-ing with random numbers obtained from the PC timer chip. In each case the random key used for a particular section is stored either as part of an XOR instruction or immediately following the call to the decryption routine. All the encryption used in WHALE is trivial from a cryptographic point of view.

VIRUS ANALYSIS

Jim Bates

WHALE... A Dinosaur Heading For Extinction

By far the largest virus that researchers have yet seen was recently uploaded to a bulletin board in the United States and comprises just under 10K of code.

The virus has been called THE WHALE since that is the "title" which appears within the code after the first level of code decryption has been executed. Disassembling this code has proven time-consuming and full analysis is incomplete due to the pressures of other work on the various researchers currently disassembling it. I am indebted to Dr. Peter Lammer of *Sophos* and Morgan and Igor of *MacAfee Associates* for access to their work on this and the report which follows collates results from all these sources although any errors in analysis or interpretation are entirely my own.

The FidoNet Message

Before describing the code in such detail as we have, mention should be made of the "motherfish" message which was posted anonymously to VIRUS ECHO on FidoNet and reported in full in last month's *Virus Bulletin*.

There are several discrepancies in this message which might suggest that the sender was either not familiar with the code or he was spreading disinformation. Since more than half the virus code is concerned with confusing and misinforming anyone trying to disassemble it, I incline to the latter theory.

The use of "motherfish" (which does not appear in the code) in preference to "whale" is strange, and the reference to the virus "learning" detection methods and being a "living, breathing entity" is fanciful in the extreme and inaccurate. That "the virus cannot be detected by present methods" is incorrect, despite concerted attempts on the part of the author(s) to make the virus undetectable. The use of the word "disavow" is interesting since text within the code suggests that the author comes from Hamburg where such a word seems unlikely to be common parlance. However, the suggestion that the code is modularly constructed is accurate, so unless this was a guess we must assume that the sender has some knowledge of the virus as a whole.

Heavyweight Confusion Coding

Following self-encrypting and "stealth" viruses, a new term has been coined by a member of the *Computer Crime Unit* at New Scotland Yard. "Armoured" virus code describes the deliberate disinformation and confusion techniques noted in FISH6 and WHALE. It is certainly appropriate in the case of WHALE since the "armour" outweighs the "stealth"!

The WHALE virus is characterised by large sections of code (estimated as at least 50 percent of the total) which involve extremely convoluted processing around and across the debug and single step interrupt handlers and accessing such hardware as the Programmable Interrupt Controller. There is no other reason for this than to confuse researchers trying to disassemble the code.

Paradoxically, the presence of this 'confusion' code has caused the research community to heave a sigh of relief. The reason for this is quite simply that such code is costly in processing time and when a machine becomes infected, processing speed slows by up to 50 percent - the WHALE is simply carrying so much programming weight (armour) that its very bulk is its giveaway. (*Rather like the dinosaurs, such viruses seem doomed to extinction, Ed.*)

A substantial amount of time and effort has been expended in writing this virus and it could well have been undertaken by more than one author. Program construction is modular and no effort has been spared to make the code difficult for scanning programs to detect.

"Virus authors have at last reached the predicted point at which their code has to carry so much protection that the parameters of invisibility and mobility can no longer be maintained."

Encryption Routines

Aside from the now accepted technique of self-encryption, this virus scrambles the order of its subroutines and varies the encryption algorithm used during file infection.

Also accepted as a "standard" technique now is the decryption/recryption process which is used to prevent detection of the virus code in memory. This technique consists of maintaining most of the resident virus code in memory in encrypted form and only decrypting it just prior to processing. Once a particular section has been executed a re-encryption routine is called which collects a new pseudo-random key value and reencrypts the code just executed before storing the new key and continuing to the next part of the code. The result is that only a small "window" around the code currently being executed is actually "in plain view", the remainder is variously and randomly encrypted. This is obviously to forestall the possibility of a recognition pattern being used to identify virus code in memory. The author(s) obviously likes this technique since it is used at least 96 times throughout the code. This is another part of the bulk that this unwieldy virus carries.

As with other recent viruses, there are several "undocumented" system calls (most of which are now well documented within the technical community) but two have been noted which may relate to specific software packages, possibly of an anti-virus nature.

General Structure

There is still much work to be done in analysing this code. However, we can say that this is a parasitic virus which infects executables with an infection length of around 9416 bytes. The actual appended length varies from infection to infection and this is probably due to the insertion of some random junk and alignment of code on paragraph boundaries.

No simple search pattern is possible because of the multiple encryption techniques and modular scrambling. There are considerable sections of self-modifying, self-checking and self-switching code within WHALE. This last technique consists of laboriously switching individual bytes within a specific subroutine using pre-calculated XOR values. The result is a sort of global XOR effect which can be used to switch between two different routines or as a decrypt/recrypt process.

The code appears to install itself as resident within the first available Memory Control Block and monitors system activity during normal DOS processing.

Stealth techniques are used to fool DOS into reporting original file sizes rather than the increased ones when files become infected. This is done by intercepting the DOS Get File Size function (23H) and checking whether the target file is infected before returning either a true or modified file size to the calling routine. (See also page 20.)

Infection Method

The virus' method of detecting infection is still being analysed but there is some evidence that several checks are made, failure of any one of which will indicate that a file is **not** infected.

The complexity of these checks means that a "sparse infection" method (i.e. not all files will be infected) may be employed. This makes external detection more difficult but it does reduce the virulence of the code and should mean that if this specimen does appear in the wild, it is unlikely to exist for long before detection and would therefore not spread too far.

One of the checks for infection seems to be that the hour field in the file time must be equal to or greater than 16 (i.e. 4pm or later) since the top bit of that field is modified within the Function 57H (Get/Set file Date/Time) handler. This too may limit the number of files suitable for infection.

Programming Style

There are several similarities with the FISH6 and 4K viruses and this might indicate either a distinct development cycle by the author(s) or simply that someone has copied useful code and ideas from the earlier specimens. I incline to the former view but whatever the truth of the matter, the similarity in file infection technique provides a useful method of identifying the presence of any of these three viruses.

However, it is reported from the United States that some generations of WHALE may not display this similarity and might therefore slip through this particular detection net.

The technique itself is discussed in the 4K data infection report on page 4 of this issue and with the exception of the differences in infected length (and the as yet unconfirmed U.S. reports), all three viruses show identical repetition of the original host header information.

Generation Code

The external results of running the WHALE have so far produced at least 27 different "generations" (*the total number of possible generations equals 30. Ed.*) and each generation appears to be the result of scrambling the order in which subroutines are written to the target file as well as changing both the encryption 'lock' and 'key'.

There is a counting mechanism fairly close to the beginning of the virus code which counts back from 0F0H (240 decimal) on the dissection copy but the significance of this has not yet become clear. Possibly sections of the virus yet to be dissected may be invoked when the counter reaches zero.

Infection apparently takes place during a Function 4BH call to DOS (Load and Execute) and thus affects COM, EXE, OVR and other executable code which is run in this way.

At various times, the interrupt vector addresses for Interrupts 1H, 2H, 3H, 9H, 13H, 24H and 2FH are accessed and may be modified for use by the virus code.

The main area of code subversion centres around the DOS Interrupt 21H and this is intercepted and passed through a function dispatcher routine. This dispatcher monitors 15 separate DOS functions including both types of Find First/Next (11H, 12H, 4EH and 4FH), Open and Close file operations (0FH, 3DH and 3EH) and various types of File Read and Seek calls (14H, 21H, 27H and 42H). Other functions handled are Get File Size (23H), Load and Execute (4BH) and Get/Set Date/Time (57H). As is now expected of this type of code, the DOS Critical Error vector is hooked during virus operation and appears to be correctly restored after use.

"The Computer Crime Unit at New Scotland Yard is now building a dossier on computer virus incidents and will seek to extradite any virus writer who causes damage to data, programs or processors in the United Kingdom."

Text Strings

As various layers of encryption are peeled back, two areas of plain text are revealed. The first of these is written to a hidden file in the root directory of the C: drive on a 1-in-4 random chance. This file is named FISH- 9.TBL and contains a copy of the boot sector of the drive, together with the following plain text:

```
FISH VIRUS 9 A Whale is no Fish! Mind her Mutant
Fish and the hidden Fish Eggs for they are
damaging. The sixth Fish mutates only if the Whale
is in her Cave.
```

No other reference is made to this file from within the virus code. The content indicates a juvenile mind at work.

The "sixth Fish" may refer to the FISH6 virus (and establish another definite link) but this has yet to be established. Since I haven't yet disassembled FISH6, I would be interested to know just how it got its name (why the '6' ?). It is also interesting to note that TBL is one of the data file extensions attacked by the 4K virus (see pages 4 - 5 and 20).

The second plain text section is displayed as a screen message if the system date is between 19th February and 20th March (*consistent with the astrological star sign of Pisces the fish. Ed*.) in any year except 1991. Subsequently the system hangs with a Divide Overflow message, necessitating a power down reboot. This is the only trigger point noted so far but there is a possibility that even these dates may be modified within differing generations, resulting in unpredictable trigger dates.

The message reads:

```
THE WHALE IN SEARCH OF THE 8 FISH
I AM '~knzyvo}' IN HAMBURG
```

This is exactly as the message appears on screen and the characters between the single quotes appear to be a name of some sort.

Elementary cryptanalysis suggests that this name is probably 'TADPOLES' (which ties in with the ichthyological theme) since this results from simply subtracting a value of 42 (decimal) from each character value. Whether the authors actually do come from Hamburg (*Chaos Computer Club? Ed*.) is not certain: since they are capable of producing this ludicrously silly code it is quite probable that they are pathological liars as well.

Many researchers have conjectured that WHALE might be designed to interact with other viruses (notably FISH6) but to date, no evidence of this has been found either within the virus code or by live testing with both viruses active on the same processor.

Possible Motives for the Virus

As knowledge currently stands on this virus, it may well be an extremely childish and malicious attempt to waste the time of virus researchers across the world. In rather the same way that the fire brigade can never ignore false alarms, the research community cannot ignore even the simplest virus code.

Any virus code is potentially destructive and the perpetrators should be aware that the *Computer Crime Unit* at New Scotland Yard is now building a dossier of computer virus incidents in the UK and will seek the extradition and prosecution of any virus writer who causes damage to data, programs or processing equipment within the United Kingdom. Under current legislation, conviction could carry a maximum five year prison sentence*. If 'TADPOLES' reads this, he/they might like to reflect on such a sentence.

The arrival of this virus caused initial consternation among knowledgeable researchers but preliminary examination has dispelled most of this concern. It is interesting to speculate that in the WHALE, virus writers have at last reached a predicted point where their code has to carry so much protection that the original parameters of invisibility and mobility can no longer be maintained with any reliability. Such bulky and processor intensive code will generally reveal itself long before any payload can be delivered.

Work will continue on disassembling and analysing this virus until all the fine details are known and further reports will appear as more information becomes available.

**Under the provisions of the United Kingdom Computer Misuse Act, 29th August, 1990. Ed.*

ADDENDUM

Dr. Peter Lammer

Jonah's Journey

The parasitic virus WHALE is not only the bulkiest but also the most convoluted specimen seen to date. WHALE uses several techniques to make itself not only difficult to find using anti-virus software, but also difficult to disassemble and analyse. However, disassembly is still a relatively straightforward process using DEBUG and the virus has now been disassembled in full.

WHALE includes several different anti-tamper measures. The 'active' obstacles range from disabling of the keyboard to exercising the single-step and breakpoint interrupts as an integral part of the code. The 'passive' traps include deeply buried routines which use checksums on the ROM BIOS data area and on WHALE's own code to detect any use of debuggers and breakpoints. If any sign of interference is found, WHALE attempts to erase itself from memory: a demure virus, which would die rather than be molested.

After removing some outer layers of active protection, one can disassemble the entire contents of the virus with relative ease, by invoking WHALE's own decryption routines in a controlled manner from DEBUG.

Many of the rumours regarding this virus are unfounded, including the claim that the virus was undetectable using conventional methods. While WHALE is distinctly slippery to detect in memory, due to the constant application of random-key de/re-cryption methods, it is relatively straightforward to find in executable files.

When WHALE infects a file, it first makes a 1-in-2 random choice whether or not to mutate and if appropriate then chooses one of its 30 possible mutations at random. Otherwise the virus replicates without mutating. Even when WHALE does not mutate, the virus constantly changes in appearance due to decrypt/recrypt routines in its code.

On disassembling WHALE's file infection routine one is reminded irresistibly of the legendary bird of paradise which, when attacked, flies in ever-decreasing circles until it disappears up its own fundament - from which position of safe refuge it is said to bombard its pursuers with abuse and excrement. WHALE performs a similar contortionist's act in memory in order to append itself to a COM or EXE file on the disk; it re-modifies all of its self-modifying code, mutates itself in memory and re-applies all of its various layers of encryption until, poised in an impossibly precarious position, it carries out a prearranged INT 21H function call to infect the target file. It then has to use its new mutated code to decrypt itself before it can return to its own depths and continue processing the file infection subroutine.

WHALE does not appear at present to do anything more significant than replicate, occasionally displaying fish-related messages. In addition to the Piscean activation dates reported on page 19, currently available copies contain a trigger date of 1st April 1991, after which no replication will take place. One of WHALE's confusion tactics is that on approximately one in ten infections, it appends a randomly chosen amount of garbage, up to 4 kilobytes, to the target file. Due to the way the virus is written, it is possible for files with extensions other than EXE and COM to become 'infected', exactly as described for 4K on page 5. This means that WHALE could inadvertantly have damaging effects on certain data or text files.

Another sign of sloppy programming is that when the virus 'forges' the lengths and time-stamps of files, it fails to distinguish between those which are genuinely infected and those which happened by chance to have a time-stamp 'hours' value larger than 15. If any file has a time-stamp hours value of 16 or more, WHALE will subtract 16 from this value (for example when a DIR command is processed), regardless even of whether the file is a program, let alone whether it is genuinely infected. If the file is of type COM or EXE, WHALE also subtracts 9216 from the reported length, again regardless of whether the file genuinely has been infected. On infecting a file, furthermore, WHALE sets the top bit of the hours field high without checking whether this would set the value to greater than 23 hours.

An interesting aspect of WHALE's programming is that one piece of self-modifying code makes the execution flow of identical copies of the virus vary from one processor to another. This depends on the length of the instruction queue. The sequence is shown here in simplified form as:

```
        mov     bx, offset retpt
        mov     al,c3           ;opcode for 'ret'
        mov     cs:[bx],al
        add     ax,020C
retpt:  int     3
```

It does not follow the same execution path on an 8088-based PC as on an 8086 based one; the 8088 chip has a 4-byte instruction queue, whereas the 8086 has a 6-byte queue. On an 8086 the 'INT 3H' instruction will already be in the queue before it is modified to read 'RET', and will therefore execute as 'INT 3H'. On an 8088, by contrast, the instruction will be modified before entering the queue and will therefore execute as 'RET'. It is unlikely that the author of the virus was aware of this particular feature of his code.

All in all, it is improbable that WHALE will pose a practical threat as a virus; it is too large and slows down performance of any PC much too noticeably to spread undetected. It is also relatively easy to detect using normal methods.

PRODUCT REVIEW

Dr. Keith Jackson

HyperACCESS/5 - A Virus Filtering Communications Package

HyperACCESS/5 is a communications software package which incorporates virus screening. This review will concentrate on the package's anti-virus features rather than try to evaluate the operation of the communications features. The review copy of HyperACCESS/5 was provided on two 3.5 inch diskettes, containing both MS-DOS and OS/2 versions.

Documentation

At 338 pages, the manual provided with HyperACCESS/5 is by no means light reading, but it does cover most aspects of operation in some depth. If specific information is required then the manual probably contains a reference to it somewhere. A four page table of contents, and a twenty six page index (both very thorough) are included.

The HyperACCESS/5 anti-virus features were explained on printed A4 sheets and in a README file. Detecting viruses during file transfer is the subject of a patent application in the USA by Hilgraeve Inc. (the developers of HyperACCESS/5) under the title "In-transit detection of computer viruses with safeguard".

Installation

The manual states that HyperACCESS/5 can be installed either on a floppy disk (of at least 720K capacity), or on a hard disk. This is not true. If you attempt to install HyperACCESS/5 on to a 720K disk, the message "Insufficient disk space. Approximately 56096 bytes short" is displayed. I assume that the HyperACCESS/5 files have grown much larger in recent times, and nobody has checked that the floppy disk installation process still works.

Even though I chose only a minimal system (no OS/2 files, no scripts or other non-essential files), the installation process executed very slowly, and took a leisurely 19 minutes 15 seconds on a humble PC. Many HyperACCESS/5 files are stored in compressed form, and have to be decompressed during the installation process. This seems to take an inordinate amount of time.

Once installed, it proved very easy to set up HyperACCESS/5 for communication via the telephone system. The menu structure is very clear, and permits any desired set of communications parameters to be specified. I had no trouble in using HyperACCESS/5 to dial several different computer systems.

Communications Facilities

As a quick summary of the communications facilities:

HyperACCESS/5 offers a dialling directory with up to 250 entries, and can transfer files using any of the Xmodem, Ymodem, Ymodem-G, Zmodem, Kermit, Compuserve-B, HyperProtocol (developed specifically for HyperACCESS), or Text (ASCII) protocols. HyperACCESS/5 can access a system automatically by learning the correct keystrokes from user activity at the keyboard, has a powerful script language, a full screen editor, and built-in anti-virus features.

The above is inevitably only a cursory description of what I found to be a very full-featured communications program.

Anti-Virus Features

The anti-virus features offered by HyperACCESS/5 are currently twofold:

1) While HyperACCESS/5 is receiving a file using one of the file transfer protocols specified above, checks can be invoked which detect viruses in the incoming file. If a virus is detected in this way, a message appears in a window on the screen of the computer receiving the file, offering to terminate the file transfer. If no action is taken when this message is displayed, then the default action is to terminate the transfer anyway. This is an eminently sensible default that permits safe unattended file transfer.

2) A utility is provided with HyperACCESS/5 which copies files in exactly the same manner as the COPY command provided with MS-DOS. This utility monitors the files which are being copied, and tests for the presence of a virus. If a virus is detected, then the copying process can be terminated, and (optionally) the virus erased.

Testing Procedure

The obvious way to test the efficacy of virus detection during file transfer is to set up HyperACCESS/5 on two computers (each with a modem) and transfer files between them. However, after some thought, transferring virus infected files through the telephone system, out of my direct control, did not seem to be a very good idea. As an alternative, HyperACCESS/5 can be setup to transfer files directly to another computer via an RS-232 cable, and this was used as an inherently safer way to transmit virus infected files. I used the Zmodem communications protocol to transfer virus infected files between two PCs. As far as checking for viruses is concerned, the actual communications protocol used seems to be irrelevant, as data is only checked once it is known to have been received correctly.

Detection Rate and Performance

HyperACCESS/5 proved to work as claimed, and samples of the 1701, 1704, Jerusalem, South African, Valert and Vacsina viruses were successfully detected. However, samples of the Kennedy and PSQR viruses were not detected, and these viruses were successfully transferred from one computer to another. At this point I realised that although the vendor of HyperACCESS/5 had sent a disk which supposedly contained the latest virus checking files, the file for checking viruses during file transfer was identical to the original version. Somebody had sent the wrong version for review!

I did encounter technical problems while transferring files between the two computers. On three separate occasions, the computer using HyperACCESS/5 (which was acting as the receiver) locked up to the extent that a power down was needed to restart the computer. The other end of this file transfer operation was using the Odyssey communications package (from *Micropack* in Aberdeen). Odyssey was required because HyperACCESS/5 would not operate from a 720K floppy disk (see above). At no time did Odyssey lock up, therefore the fault lies firmly with HyperACCESS/5. Given this problem, and the aforementioned limited range of viruses that could be detected, it is only fair to conclude that detection of viruses during file transfer works, but needs some problems ironing out, not the least of which involves distributing the correct files.

The developers of HyperACCESS/5 use the virus pattern file compiled for IBM's scanning program (*incorporated with the full cooperation of IBM. Ed.*). The version with which I was provided was dated 20th April 1990 which rendered the pattern file obsolescent. I decided that it would be fairer to test the efficacy of virus detection using the copy utility provided with HyperACCESS/5. This was dated 24th August 1990, and provided a much more comprehensive list of 73 viruses (60 unique viruses and 13 variants) which it claimed could be detected during file copying. (*The most recent virus detection files can be downloaded directly from Hilgraeve's BBS in the USA. Ed.*)

The copy utility was tested by copying the *VB* test set of 97 parasitic viruses on a file by file basis from one floppy disk to another. The results were extremely impressive. The only viruses which were not detected were: AIDS, 1260, three of the five variants of the Yankee virus, and five of the ten variants of the Vienna virus. Apart from the usual problems of nomenclature, all other viruses were detected correctly.

The naming system used by HyperACCESS/5 is heavily biased towards numeric names for viruses e.g. the Valert (Tenbyte) virus was called 9800:000, and the Fu Manchu, Jerusalem, Kennedy, Perfume, Vcomm and Zero Bug viruses were all referred to by their infective lengths (2086, 1813, 333, 765, 637 and 1536 bytes respectively).

When the HyperACCESS/5 copy utility detects a virus, the user must specify whether or not to abort the copying process, and whether or not to delete the original infected file. All this happens on a simple question and answer basis.

The above quoted results are impressive in that HyperACCESS/5 successfully detected 43 out of 47 parasitic viruses, and 89 out of 99 variants of these viruses.

It should be noted that transferring a pure boot sector virus via a modem is very difficult, as the infected boot sector has to be extracted, transmitted, and then replaced on top of the boot sector in the recipient computer. This constraint does not apply to the newer multi-partite viruses which infect programs and the boot sector (*see VB, September 1990, p.3*).

Checking for the presence of a virus during the transfer of a file between two computers seems to have very little effect on the rate at which files are transferred. So much so that given the very short virus infected files that were being transferred in the above tests, there was no perceptible slowdown. This is not surprising, as the processor lets the serial port controller chip(s) do most of the work during file transfers. Therefore, further checks while waiting for data to be received will not slow the operation down. The HyperACCESS/5 documentation claims that up to 2400bps transfer speed, checking for viruses has no effect on even the slowest of PCs. My tests confirm that this is true. It is also claimed that on 286 and 386 PCs, checking for viruses has no effect up to 19200bps transfer speed. I have no means of testing this claim.

The utility which checks for viruses while they are being copied took 24.5 seconds to copy 8 files from one part of a hard disk to another. The MS-DOS COPY command copied the same set of files in 5.7 seconds. The difference in speed between the two transfer methods is therefore more than a factor of 4. This means that the virus checking copy utility should only really be used in circumstances where a virus is suspected: e.g. copying files from a newly received floppy disk. The virus checking copy utility is too slow for routine use. Given the successful virus detection rate (see above), this slow execution speed is a shame. It mars an otherwise excellent product. (*Obviously, faster processors such as the 286 or 386 will significantly improve file transfer rates. Due to the risks involved, sacrificial machines and those used for evaluating anti-virus software tend be the humblest of 'work horses'. Ed.*)

Compression

All of the above assumes that infected files will be transmitted/copied in their standard form. Unfortunately, this is not always true. To save time during transfer, files are often compressed before transmission, and decompressed after transmission is complete. The HyperACCESS/5 documentation warns that virus infected files which have been compressed in this way will not be detected and that compressed file(s) must be checked separately after decompression. This applies regardless of which compression system is used (ARC, ZIP, LZH, PAK and ZOO etc.). It would be possible to check files as part of the transmission process, but this would necessitate "unzipping on the fly". (*No comment. Ed.*).

These problems are gradually being alleviated by the use of MNP error correction/compression protocols at the modem level. From level 5 upwards, MNP adds data compression at the data transfer level, thus making external compression unnecessary. HA/5's own file transfer protocol, HyperProtocol, has just been placed in the public domain as a free-standing DOS module. This utility provides file compression on the fly and error detection. Whether it will be adopted as widely as MNP error correction is debatable.

Assessment

Assuming that the level of virus detection exhibited by the copy utility can be repeated in virus detection during file transfer, HyperACCESS/5 lives up to its claim to be a full-featured communications package with built-in detection of virus infected files.

Technical Details

Product: HyperACCESS/5

Vendor: Firefox Communications Ltd., Aspen House, 9 Coventry Road, Colehill, West Midlands B46 3BB, UK. Tel 0675 467244, Fax 0675 463504.

Developer: Hilgraeve Inc., Genesis Centre, 111 Conant Avenue, Suite A, Monroe, Michigan 48161, USA, Tel: +1 (313) 243 0576.

Availability: IBM PC, PS/2, or 100 compatible with either a built-in modem or an available RS-232 serial port, running under MS-DOS or OS/2. A hard disk is not mandatory. Many different modems are supported, including (but not exclusive to) those conforming to the Hayes command set.

Version Evaluated: 1.1

Price: £175 for both DOS and OS/2 versions.

Hardware Used: An Amstrad PPC640 with a V30 processor, and two 3.5 inch (720K) floppy disk drives, running under MS-DOS v3.30. Also an ITT XTRA (PC compatible) with a 4.77MHz 8088 processor, one 3.5 inch (1.44M) floppy disk drive, two 5.25 inch floppy disk drives, and a 32Mbyte Western Digital hard card, running under MS-DOS v3.30. The modem used was a British Telecom PC424X (a badged Dowty Quattro on a plug-in card), Hayes compatible, and operable up to V22bis (2400bps).

Virus Test Suite: This set of 49 unique viruses (according to the virus naming convention employed by *VB*), spread across 101 individual virus samples, is the standard *VB* test set. It comprises two boot viruses (Brain and Italian), and 99 parasitic viruses. The actual viruses used for testing are listed below. Where more than one variant of a virus is available, the number of examples of each virus is shown in brackets. For a complete explanation of each virus, and the nomenclature used, please refer to the list of PC viruses published regularly in *VB*:

405(2), 4K(2), AIDS, Alabama, Amstrad(2), Anarkia, Brain, Cascade(10), Dark Avenger(2), Datacrime(3), dBASE, December 24th, Devils Dance, Eddie(2), FuManchu(3), GhostBalls, Hallochen, Icelandic(2), Italian, Jerusalem(6), Kennedy, Lehigh, Macho-Soft, MIX1(2), Number of the Beast, Oropax, Perfume, Prudents, PSQR, South African(2), 1260, Suriv(8), Sylvia, Syslock(2), Taiwan, Traceback(4), Typo, Vacsina, Valert, Vcomm, Vienna(10), Virdem, Virus-90, Virus-B(2), VP, W13(2), XA-1, Yankee(5), Zero Bug,

Transmission of Executable Files - A Hazardous Business

The hazard of downloading computer viruses or Trojans from Bulletin Boards has forced many organisations to forbid the use of BBS software. In fact, the systems operators (SysOps) of reputable Bulletin Boards take great care to screen software before it is made available for download. In a recent incident on the CIX BBS, a handful of people downloaded a Cascade infected program; system monitoring enabled the recipients to be traced and warned by telephone.

A far more serious incident occurred last February when the Tenbyte virus was posted by accident to the V-ALERT electronic mailing list. The virus was clearly labelled as such and was intentionally downloaded by more than 500 people within the space of a few hours. As *VB* predicted, the Tenbyte virus has now appeared in the wild in the United States. (*VB, April 1990, pp.4-5.*)

In the face of the threat of accidentally downloading or receiving virus infected files, the use of virus 'filtering' provides a first line of defence, particularly for regular users of BBSs or those who often transmit and receive executable images. HyperACCESS/5 is the only product of which *VB* is aware that claims to provide virus filtering. The product successfully detected all of the parasitic viruses which are a current threat in the UK.

However, communications packages which incorporate filtering are restricted by certain inherent limitations. Nearly all software which is destined for transmission is compressed. There is a range of commercial and share-ware compression utilities*. Searching for a virus in a compressed file is possible, but becomes an impractible proposition when the multiplicity of possible compression techniques is accounted for. Furthermore, while filtering packages may readily detect conventional 'first genera-tion' viruses (by searching for hexadecimal patterns), they are not suitable for detecting any form of virus which uses self-modifying encryption.

Defence in-depth is simple commonsense. The use of a filtering package followed by scanning using more conventional anti-virus software would appear to provide security - for the moment.

**Dynamic decompression, as offered by LZEXE, is unsuitable for the transmission of untrusted software. It's use should be restricted to compressing clean EXE files for storage on disk. (See VB, June 1990, p.12).*

END-NOTES & NEWS

The *Virus Bulletin* is hosting a two day **conference on combating computer viruses**, September 12-13th 1991. The venue will be the Hotel de France, St. Helier, Jersey. The conference will be chaired by Edward Wilding (UK) and Fridrik Skulason (Iceland) and speakers include Jim Bates (UK), Vesselin Bontchev (Bulgaria), David Ferbrache (UK), Ross Greenberg (USA), Jan Hruska (UK), John Norstad (USA), Yisrael Radai (Israel), Ken van Wyk (USA) and Gene Spafford (USA). Several additional speakers will be confirmed in the final programme. Information from Petra Duffield, *Virus Bulletin Conference*, UK. Tel 0235 531889.

CERT Advisory CA-9007 dated October 25th, 1990 warns of a **VMS VAX system vulnerability** (versions 4.0 to 5.4). It describes how non-privileged users can acquire system privileges through the ANALYZE/PROCESS DUMP routine. *CERT* e-mail to cert@edu.cmu.sei.cert or telephone USA 412 268 7090 (24 hour hotline).

The US *National Institute of Standards & Technology* (NIST) is considering the development of a **government-industry consortium** to combat computer viruses. Contact Dennis Steinauer, NIST, Rm. A216, Technology Building, Gaithersburg, MD 20899, USA. E-mail to steinauer=ecf.ncsl.nist.gov or telephone 301 975 3359.

Heriot-Watt University, Edinburgh, UK, hosts a one-day seminar '**Computer Viruses: Protect IT**' on November 23rd, 1990. The programme will address the appearance of stealth and armoured viruses and current software developments. Contact David Ferbrache, Department of Computer Science, Heriot-Watt. E-mail (Internet)<david@cs.hw.ac.uk>, (Janet)<david@uk.ac.hw.cs>. Tel 031 225 6465 ext 553.

RG Software Systems, developer of Vi-Spy (*VB*, May 1990) is moving to a **new headquarters**. The company's new address is RG Software Systems, Inc., 6900 E. Camelback, Suite 630, Scottsdale AZ 85251, USA. Tel 602 423 8000, Fax 602 423 8389.

Briefing on Computer Viruses by Fred Cohen, London, November 13th 1990. Contact *IBC Technical Services*, UK. Tel 071 236 4080.

S & S Consulting Group, UK, is holding a two-day seminar on '**The Virus Threat**', February 13th-14th 1991. Tel 0494 791900.

The following appeared in the UK's *Daily Star* newspaper on 12th October 1990:

> *Speech goes to pot! RED-FACED diplomats are trying to find out who inserted a plea to legalise cannabis into a speech by Hong Kong Governor Sir David Wilson. His commentary on the future of the British colony, which was delivered to newspapers on computer disks, contained the surprise announcements: "Your PC is now stoned. Legalise marijuana."*

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