Computer Viruses
An Introduction

Jeffrey Horton
Department of Computer Science
University of Wollongong
Northfields Avenue, Wollongong
jeffh@cs.uow.edu.au

Jennifer Seberry
Department of Computer Science
University of Wollongong
Northfields Avenue, Wollongong
j.seberry@cs.uow.edu.au

Abstract

Computer viruses pose a considerable problem for users of personal computers. The recent emergence of macro viruses as a problem of some importance may heighten virus awareness in general. Yet most people have little or no understanding of common anti-virus measures, the varieties of viruses that exist today, and the strategies which they use to accomplish infection and to defeat anti-viruses. It is well-known that the virus problem is most severe for users of IBM PCs and compatibles; however, users of other platforms, such as the Macintosh, should not become complacent — viruses exist for many platforms in varying numbers. The ease with which macro viruses may be written is discussed, and a new virus attack for the Macintosh is presented which closely resembles an attack under DOS for the PC.

Keywords Computer science, computer virus, computer viruses, macro viruses, companion viruses.

1 What is a Computer Virus?

There is some difficulty in producing a definition for the term "computer virus". Dr. Cohen has presented a mathematical definition of a computer virus, which may be roughly expressed as:

> A virus is a program that can 'infect' other programs by modifying them to include a possibly evolved version of itself. [4]

However, this definition classifies as viruses many things which would not be considered viruses by those working in the anti-virus field. At the same time this definition would not consider as viruses programs that infect another without modifying the target program itself [2] (an example of such a virus are the **companion viruses**, discussed in Section 3).

Proceedings of the 20th Australasian Computer Science Conference, Sydney, Australia, February 5–7 1997.

Additionally, the above definition does not convey any need for the "virus" to be able to replicate further once it has infected some other program [2] — and replication is viewed as an important characteristic of a true "computer virus".

A definition which is felt to be more practically useful when dealing with "real" computer viruses than Dr. Cohen's mathematical model is:

> We define a computer 'virus' as a self-replicating program that can 'infect' other programs by modifying them or their environment such that a call to an 'infected' program implies a call to a possibly evolved, and in most cases, functionally similar copy of the 'virus'. [21, 2]

The term 'infect' is used with respect to computer viruses in the sense of the definition above throughout the remainder of this document.

It is important to note that a virus is not necessarily malicious — although it may have side effects (as a result of the virus clashing with the operating system, user programs and extensions to the standard operating system installed by the user) which are deemed to be undesirable.

Some researchers have been considering the question of viruses that perform useful actions — so-called **benevolent viruses**. Cohen [4, pp. 15–21] considers briefly the topic of benevolent viruses. He considers as examples:

Compression viruses — little-used files are compressed by the virus and uncompressed when required.

Maintenance viruses — any virus which would perform maintenance tasks in a computer system, such as updating installed programs.

Distributed Databases with viruses — viruses would reproduce on networked computers, performing searches for the virus' originator. Results would be reported back by mail, and the virus would clean itself up after a certain time.

The use of viruses in covert distributed data processing (specifically, key cracking on encrypted messages) has been proposed by White [13]. It may be argued, however, that this would not be particularly beneficial to the user whose resources are used by the virus.

Many arguments against the idea of benevolent viruses are presented by Bontchev [2].

2 Worms, Trojan Horses, Droppers and Logic Bombs

A **worm** is an independent program that is able to spread copies of itself or of parts of itself to other computers, commonly across network connections, and these copies are themselves fully functional independent programs, which are capable either of spreading further and/or of communicating with the parent worm (to report back results of some computation, for example).

There is often confusion over the distinction between a **worm** and a **virus**. For example, the program that negatively affected the Internet in November 1988 is referred to as a virus ("the Internet Virus") by some [5] and as a worm ("the Internet Worm") by others [11, 12, 10]. Spafford [12] argues that referring to the infection as a "worm" rather than a "virus" is most appropriate.

A notable difference between worm programs (such as the Internet Worm) and viruses is that while a virus may take advantage of network connections to infect other programs (some local area networks are particularly susceptible, as the user is able to interact with programs and data stored on a remote machine as if they were available locally), it is not capable of causing its code to execute on a remote machine. Clearly the well-known worms have been able to cause their programs to be executed on the remote machine which was the worm's target.

The Internet Worm affected Sun 3 and VAX systems running variants of BSD UNIX [11]. Other worms have been created with other networks in mind, such as DECnet [7].

A **trojan horse** is a program which possesses various intentional undocumented features[1] whose effects few users of the software would appreciate were these undocumented features to manifest themselves. Unlike a computer virus, which attaches itself to some other program using any of a number of methods, a trojan horse is a self-contained program. A trojan horse may have functions of use to the user.

Some definitions of "trojan horse" define a computer virus as a replicating trojan horse which inserts a copy of itself into some other program [1].

[1] As opposed to bugs in the program, otherwise known as "unintentional undocumented features".

A trojan horse might install a virus as its intentional undocumented action. Some feel that a program which installs a virus as a result of having been previously infected with the virus is also a trojan horse (having been converted into a trojan horse by the virus) [18]. However, this seems incompatible with the notion of a "trojan horse" being a program which was initially produced with an intentional undocumented feature in place.

A **dropper** is a program which acts as a carrier for a computer virus. A dropper is not the result of a normal infection of some program by a virus — it exists only to spread the virus. The virus is usually kept by the dropper in a form which will not be detected by anti-virus software [18]. Some **macro viruses** (see Section 3) attempt to act as a dropper for more conventional varieties of viruses [15], in addition to whatever other actions they perform.

3 Varieties of Viruses

There are a number of different ways that viruses use to infect a computer system. The two main types of viruses are:

File Infectors: These are viruses that attach themselves to some form of executable code. There is a variety of ways in which a virus might attempt to infect a file. On a DOS-based system, file infectors will commonly attach themselves to `.COM` or `.EXE` files, although there are many other kinds of infectable objects.

Boot Sector Infectors: Only discussed in the context of a PC-compatible system. These kinds of viruses infect executable code which is loaded from disk and called when a computer is starting up. There are a number of different pieces of code which may be modified by a virus to infect a system, such as:

- DOS boot sector [floppy disks and hard disks].
- Master Boot Record (MBR) [hard disks only].
- Partition table [hard disks only].

A virus that is capable of spreading by infecting files and by infecting via any code executed at boot time is known as a **multipartite** virus.

Boot sector viruses are extremely widespread; as a group they are easily the most commonly found variety of virus on PC-compatible systems.

A virus may be **direct-action** or **resident** [18]. A direct-action virus is one that when initially executed in the course of normal use of a computer system identifies executable objects for infection and exits once infection has been accomplished.

Direct-action viruses may also be referred to as **non-resident** viruses.

A resident virus is one which installs itself somewhere in memory, and makes arrangements for the virus body in memory to be executed at some future time; the virus may infect files or take other action (to conceal its presence, for example) at the time it is next executed. For example, some Macintosh viruses if resident in memory will infect an application when that application commences running and performs certain system calls that initialise the Macintosh Toolbox, which consists of a set of utility functions available to all applications.

Some programs useful to the user are also resident programs — this includes some antivirus programs that monitor computer system operations for actions which may indicate the presence of a virus.

There are some other types of viruses which should be mentioned:

Macro Viruses: These are explained in detail in Section 4. The ease with which such a virus may be written is discussed in Section 9.2.

File System or **Cluster Viruses:** Rather than infecting files directly, such a virus modifies directory table information so that the virus is executed first. It would then pass control to the program that the caller really wants so as to avoid rapid detection. "Dir-II" is an example of this variety of virus.[18]

Kernel Viruses: These are viruses that target a specific feature of an operating system's kernel (the program(s) that represent the heart of an operating system). [18]

Companion Viruses: Companion viruses occur in several varieties [3, 8], the most notable being:

Regular Companion: Creates a file in the same directory as the target of infection but with a filename extension which the operating system chooses to execute before the original file (for example, under DOS a `.COM` file is executed before a `.EXE` file with the same name). This would appear to be an attack which is highly specific to PCs running DOS.

PATH Companion: Create a file with any executable extension in a directory that is searched for executable files before the directory containing the target of infection (named after the PATH environment variables found in operating systems such as DOS and UNIX).

Obviously, not all of these infection strategies are available on some platforms and operating systems. For example, Macintosh computers do not suffer from companion viruses as described above in any form, and also don't appear to be afflicted with viruses of the boot sector variety.

All viruses, with the exception of macro viruses, are platform dependent.

4 Macro Viruses

The anti-virus community has been aware for some time now of the potential for virus-writing provided by the scripting (or macro) languages of large software packages such as Microsoft Word and Excel The idea of macro viruses was first introduced by Highland [6].

However, it is only recently that such viruses have become a problem. These viruses are remarkable for the fact that they infect what are usually thought of as documents. A macro virus may also be platform independent, being capable of spreading on any computer platform supported by the host application. The most well-known and widespread are Microsoft Word viruses written in WordBasic (two examples are **Concept** and **Nuclear**).

Microsoft Word recognises the existence of two different forms of user file — an ordinary document, and a template. A template is very much like an ordinary document, with the addition that templates may also contain macros written in WordBasic. Ordinary documents may be converted easily into templates, but the reverse is not the case. Word has a "global template", the so-called "Normal template", which is used by every document created, and is important for the spread of macro viruses — the macros that make up the virus are copied into the Normal template where they are subsequently available to other documents.

Word macros may be marked as *ExecuteOnly*, which means that the macros cannot be easily edited or inspected by a Word user but can only be executed. Some viruses, for example the **Nuclear** virus, use this method to hinder casual analysis of the viral macros.

Macro viruses in Microsoft Word exploit the existence of several varieties of macros within WordBasic [15]:

- The **AutoExec** macro, stored within some global template such as the Normal template, which is executed automatically whenever Word is started.
- "Auto" macros, which run whenever certain user actions take place within Word:
  - **AutoNew** runs when a new document is created.

- **AutoOpen** runs when an existing document is opened.
- **AutoClose** runs when an open document is closed.
- **AutoExit** runs before Word exits when the user quits.

- Macros named for Word menu options, which are run when the menu option is selected.

A basic Word virus is not difficult to create, requiring just one macro. The macro virus **DMV** features a single macro **AutoClose**. The Normal template will be infected when a document containing the DMV **AutoClose** macro is closed. Subsequently, documents are infected as they are closed [15]. **Concept**, a more complicated macro virus, signals its initial infection of the Normal template, and subsequently will infect any document with the viral macros when the document is saved using the "Save as ..." option of the "File" menu (an example of a virus using a macro named for a Word menu option).

Some macro viruses, such as **DMV** and **Concept** do nothing but spread. Some would argue that even this is damaging, in terms of the time required to remove the virus macros from whatever documents have been infected. More damaging actions are certainly possible, however. For example, a virus might delete paragraphs from a document, or rearrange or insert words (the **Nuclear** virus will append some lines of text to documents when printed at a certain time). More sophisticated macro viruses attempt to infect the user's computer with an ordinary variety of virus which infects executable files (the **Nuclear** virus unsuccessfully attempts to do this), however behaviour such as this is platform dependent.

A virus can also give itself some (limited) protection against being removed by implementing a **ToolsMacro** macro, which is then executed in place of the corresponding menu option should it be selected by the user. This menu command offers one possible way for macros to be removed from a document, if the environment has not already been infected by a virus.

Disabling of the features of Microsoft Word which allow automatic execution of certain macros when documents are opened and closed offers some limited protection, but as macro viruses can be written with macros that mask menu options (macros which cannot be disabled in the same way that automatic macros may), this is hardly a complete solution.

5 Virus Occurrences

There have been a great many viruses created for many different computer platforms. The PC has by far the largest share of all the viruses in existence (the producers of *Dr. Solomon's Anti-Virus Toolkit*, a leading anti-virus package, claim to detect 9417 PC viruses [20]).

Many of these PC viruses are closely related (once a virus becomes available, it is not uncommon to find a number of copycat viruses that differ only slightly from the original appearing, perhaps written by less-skilled virus writers using virus source code; alternatively, the virus author might release a number of viruses with different payloads but sharing common code for infection and anti-anti-virus measures).

Some information has been gathered by *Virus Bulletin* about PC viruses that have been reported as found over the course of a month for some months. The percentage of the reports made up by the various virus classes for several months of 1996 are shown in Table 1 [22].

	Jun.	Jul.	Aug.	Sep.
Macro	21.8	18.4	20.2	34.3
Boot	62.5	64.5	59.2	53.7
Multipartite	7.3	10.6	13.9	6.6
File	8.4	4.8	6.4	4.8
Other	0.0	0.3	0.0	0.0
Unclassified	0.0	1.3	0.4	0.6

Table 1: % of reports to/collected by *Virus Bulletin* made up by various classes of virus.

Only a very few of the total number of known viruses are responsible for the majority of virus incidents. Some on-access anti-virus products use this fact to help restrict the number of viruses that a file must be checked for when accessed by a user.

Viruses exist for a number of other personal computer platforms, such as the Amiga and the Macintosh, but the numbers of these viruses are a small fraction of the numbers of viruses available for the PC. The Macintosh, for example, is afflicted with only a few dozen viruses. Viruses written to target UNIX are especially uncommon.

6 Virus Anti-Detection/Anti-Analysis Strategies

There are a variety of strategies which a virus might employ to hinder detection of the virus, and analysis once it has been discovered:

Stealth: While the virus is active in memory, it can intercept disk reads, and when it detects an attempt to read a section of the disk (such as the boot sector, partition table or master boot record) or a file that has been infected by the virus, can conceal its presence by returning as a result of the call data with no signs of infection. Most stealth viruses are boot sector viruses (it being much easier to detect reads

of these areas of the disk). Some examples of stealth viruses are [17]:

- Members of the "Brain" strain of viruses are stealth floppy boot sector viruses.
- The "512" virus (also known as "Number of the Beast") is a stealth file infecting virus.

Polymorphic Viruses: A polymorphic virus is one that is capable of varying many aspects of its appearance in the hope of avoiding detection. Cohen refers to viruses of this type as "evolutionary" viruses [4, p. 73]. The strategies which might be employed by a polymorphic virus are (many of these strategies for code 'evolution' are explained in greater detail by Cohen [4, pp. 199–215]):

Encryption: Encrypt the body of the virus not only with a variety of different keys but also with a variety of different encryption strategies (each requiring a different decryptor, of course). The encryption approach is a particularly common polymorphic trick — its wide use was facilitated by the distribution of a variety of object code modules (such as **MtE** and **TPE**) which could make any virus polymorphic. Additionally, this is the most straightforward polymorphic strategy which a virus might employ. One of the many viruses which employs a strategy of this sort is the "Tequila" virus [17].

Instruction Equivalence: Some machine instructions achieve equivalent effects. Substitute for these equivalent instructions in the virus code.

Equivalent Instruction Sequences: Replace one sequence of machine instructions with another which achieves the same final result.

Instruction Reordering: Sometimes it is possible to reorder a sequence of instructions in a variety of different ways and still achieve the same result. The "1260 virus" is one which alters the order of instructions in its decryption routine from infection to infection, and additionally inserts irrelevant instructions [17].

Variable Substitutions: Alter which memory locations contain each of the variables used by the program. Especially effective if the variables are dispersed about the program.

Add or Remove Jumps: Add unnecessary jump/branch instructions to a program while still preserving its function (this makes the program longer, of course), or remove existing jump/branch instructions by moving the program code which was the destination of the jump/branch.

Add or Remove Calls: Replace calls to subroutines with a copy of the subroutine; replace a sequence of code with a subroutine call.

Garbage Insertion: For example, might add NOP instructions, or other instructions which do not otherwise achieve any useful purpose apart from obscuring the code's functions. The "Tequila" virus employs a strategy of this sort [17].

Simulation: Replace an instruction sequence with an alternative sequence which achieves the same effect when interpreted (or simulated) appropriately.

Build and Execute: 'Build' instructions somewhere in memory (one byte or a sequence of bytes at a time) and then execute them. This obscures the instructions being constructed from observation until their execution.

Intermixing Programs: Interleave the instructions for separate programs or blocks of code in such a way that the interleaved code achieves the same result as the separate blocks of code.

Tunnelling: A technique sometimes used by viruses that attempts to bypass activity-monitoring virus detectors (for example, by bypassing operating system disk access functions and interpreting the contents of the disk itself, or calling the operating system's functions directly to avoid any trap the activity monitor may have set in place) or otherwise subvert anti-virus techniques and strategies.

Cavity Virus: A variety of file infecting virus which overwrites a portion of the file which is filled with the same value, such as a region filled with zero bytes. Such a virus will not alter the total length of the file.

Antidebugger Mutations: Various tricks can be employed to make disassembling and tracing the program code difficult. For example, the program code might be arranged in such a way that an instruction or set of instructions is hidden from casual inspection,

so that the function of the code is no longer readily apparent from its disassembly. The debugger itself could be manipulated in the course of tracing a program by tampering with the debugger's address space.

The speed with which a virus reproduces and infects other files is also important. Some viruses spread particularly quickly. For example, a virus might be programmed to infect any executable file when the file is opened, for whatever reason. These are the so-called **fast infectors**.

A **slow infector** is a virus which only infects files as they are created, immediately prior to or following a legitimate change (that is, some change that the user wants to have happen), or as files are copied, perhaps onto a floppy disk [3, 9, 18] [4, p. 91].

7 Defenses Against and Detection For Viruses

There are a variety of defenses against viruses, and ways of detecting their presence. A natural first question is: "Is it possible to detect *all* viruses?". Unfortunately not — it is mentioned in Cohen [4, pp. 64–68] that it is not possible to construct some program which correctly determines whether or not some other program is a virus. In fact, given a known virus, it isn't even possible to systematically determine using a computer program if another program is infected with a virus derived from the original known virus in some way.

However, there are a variety of imperfect ways to detect the (possible) presence of computer viruses. Most forms of virus detection involve *false positives*, which occur when an object is identified as being infected by a virus when in fact it is clean, and *false negatives*, which occur when an object is passed as clean when in fact it has been infected with a virus. Ideally, false positives and false negatives will occur very infrequently.

Some techniques, such as scanning for viruses, often lead to a positive identification of a virus. In many cases, the infection (and some damage) caused by the virus is reversible.

Stealth techniques mean that attempts at virus detection that involve manipulating file objects on disk will not necessarily be effective if the virus is present and active in memory. For this reason, it is recommended that before attempting to detect a virus, the computer in question be rebooted from an uninfected, locked, floppy disk containing a clean version of whatever anti-virus software is needed to search for viruses.

Known-Virus Scanning: Attempts to identify viruses by scanning files for certain strings of bytes known to occur in particular viruses. Simple scanners which perform only such searches are easily defeatable by a well-written and sophisticated polymorphic virus, so many also employ some more advanced techniques (such as *heuristic analysis*, to detect suspicious code fragments, or *algorithmic analysis*, to detect complex polymorphic viruses). Scanners often have difficulty detecting new viruses, and they require frequent updating.

Heuristic Analysis: Attempts to identify a possible virus by looking for code that performs functions which in combination with each other are deemed to be suspicious. An example of a suspicious code fragment would be one that alters the first few bytes of an executable file in memory — this would be required so that an infected executable could run normally when infected with a virus.

Behaviour Blocker/Monitor: Attempts to detect viruses based on patterns of virus activity. This approach has problems because many of the actions performed by a virus are perfectly legitimate under other circumstances. These methods are also sometimes rendered ineffective if a tunnelling virus is infecting the system — these viruses are frequently able to bypass the methods used by a monitor to detect virus activity.

Integrity Checker/Integrity Shell: Integrity checking involves collecting a database of signatures for each file which is likely to be the target of a virus infection (such as application programs). If at a later date this signature can be determined to have changed, then it is possible that a virus infection has taken place. This method will not detect viruses before infection takes place. There are a variety of enhancements to the basic method outlined here which must be implemented (for example, a **companion virus** does not necessarily modify the item it "infects". So integrity checkers must attempt to identify the presence of a companion virus by other means). An integrity shell [4, pp. 83–93] is a more sophisticated approach which involves checking every object on which some object X (which the user wishes to use in some way) depends.

An integrity checker is an example of a generic anti-virus program — it is not targeted at a specific virus or class of viruses and so will rarely need updating. Integrity checking issues are extensively discussed in papers by Bontchev [3] and Radai [9].

There are a variety of other ways to help prevent virus infections. For example, as the great major-

ity of PC virus infections are boot sector viruses, changing the order in which the computer searches its disk drives for a bootable disk is an effective defense in many cases (an apparently common setting is to attempt to boot a floppy disk before attempting to boot the hard disk). Precautions will still have to be taken to deal with multipartite and other non-boot sector viruses, of course.

Cohen outlines a number of strategies that will prevent or hinder computer viruses spreading throughout a computer system or computer network [4, pp. 57–64]. The most interesting of these approaches is that of **limited sharing**. The best that can be done to limit sharing in a transitive information network that implements sharing is to base the structure on a "partially ordered set", or POset. This means that information can flow in only one direction, for example, from Host A to Host B but not from Host B to Host A. This effectively limits the possible range of a viral infection, and also helps to trace the origin of any suspected infection, as the source of the infection would be one of a limited number of machines which had access to all of the machines on which the infection was ultimately detected.

The use of a "vaccine" against certain computer viruses is a technique no longer widely practiced. Most viruses check a potential infection target to make sure it is not infected by that particular type of virus (to prevent multiple infections), so infection by a particular virus could be prevented by marking executables so that they appeared to be already infected (hence the name "vaccine"). The large number of viruses and the fact that some virus' identification techniques are mutually contradictory means that this technique is no longer workable.

8 Problems with and Attacks Against Anti-Virus Measures

Virus scanners need updating with great frequency because of the speed with which new viruses are created and released. They are popular for a number of reasons:

- A scanner is usually straightforward to use. Whether or not it is in fact used in the recommended manner is another matter.
- When a virus is detected it can often be positively identified, and many scanners include facilities for "disinfecting" infected files.
- A scanner is the most reliable means of detecting a known virus in a new file; other techniques are not necessarily applicable (for example, an integrity checker cannot be used to check a newly-obtained program for viruses, because there is no way of determining what the signature of an uninfected program should be).

A scanner will sometimes perform poorly when attempting to detect unknown viruses.

Polymorphism was at one time an effective attack against scanners which merely searched for strings of bytes known to characterise certain viruses. Cohen states that "until several years after the MtE was spreading in the world, no scanner was able to pick up over 95% of infections" [4]. The situation has improved greatly in recent times — most good scanners are capable of detecting the majority of polymorphic viruses.

As mentioned in Section 7, stealth viruses may cause problems if the virus is present in memory when using a virus scanner or integrity checker, as files presented for inspection may appear clean when in fact they are not. Furthermore, a variety of fast infector stealth virus may take the opportunity presented by the opening of numerous files for checking to infect those files; this represents a serious cleanup problem in an environment with hundreds (or possibly even thousands) of executable files. An attempt must be made to identify such viruses in memory.

Activity monitoring programs require updating as well, to cope with new virus behaviours. There are some varieties of virus behaviour which are not readily detectable by a monitor — such as infecting only files which are about to be modified in any case. One particular virus, the "Darth Vader" virus, was designed to avoid alerting an activity monitor program by attaching itself only to certain files as they were copied [3]. This also presented an effective attack against integrity checkers at that time.

Activity monitors have the additional problems in that they may flag legitimate actions as suspicious, since the functions used by computer viruses commonly have legitimate uses as well. They might also be bypassed by a tunnelling virus or disabled in memory [3, 9].

Slow infectors are a concern for integrity checking software. The virus infection may go unnoticed, because the integrity checker doesn't have any signatures in its database with which to compare that of the (new) file. An example of a common legitimate change to an existing executable file is the addition of patches to the file by an updater program. Radai [9] suggests that it would be possible to detect the presence of a slow virus by creating a series of small executable files in the hope that one will be infected by a slow virus. Copies of the created executable could be created for easy comparison with the first. This will not establish which other newly-created or modified objects are infected with the slow virus, however. Tracking down the source of the infection could be problematic.

Programs which cause modifications to their own executable code will cause problems for integrity checkers. It is difficult to know how widely such bad programming habits are practiced at this time.

Integrity checkers will not be effective against all types of viruses. As integrity checking is usually applied only to hard disks (its application to floppy disks is not practical, as the contents of a floppy disk are frequently modified) a virus which infects floppy disks only and ignores hard disks would go unnoticed [9, 3]. The "Brain" virus, an early DOS virus, is an example of a virus which ignores hard disks.

Finally, integrity checkers need to be carefully constructed so that the database of checksums is not easily compromised by a virus. For example, a virus might attempt to forge an entry in the integrity checker's signature database for a program which is a target of infection.

9 Recent Work

9.1 Companion Viruses and the Macintosh

Macintosh viruses infect application files, the System file (a file present on every bootable Macintosh disk which holds many resources used by the operating system) or system extensions (small files containing executable code which load every time the Macintosh is started up and which add extra functionality to the operating system). A very few viruses which no longer work under recent releases of the operating system infected the Macintosh by more unusual means.

The Macintosh does not have any features which correspond to the preference of DOS to execute .COM files before .EXE files of the same name (when the selection is not made explicit). Nor does it feature the concept of a "path" along which the operating system searches for applications to execute if the desired application is not in the current directory, as DOS does. So a companion virus which does not actually alter the target application is not implementable in the same manner as under DOS.

However, it is possible to produce a virus with many of the same characteristics as a companion virus by manipulating a Macintosh disk's Desktop Database. This attack seems not to have been explored to date.

Macintosh files have both a **file type** (for example, 'TEXT' denotes a plain text file) and a **creator**. Each application should have a unique creator code, with which the files that the application creates are marked. The "Finder" (the part of the Macintosh operating system which is responsible for presenting the graphical user interface and managing user interactions) stores information about file creators which allows it to determine what application should be launched when a document icon is double-clicked, and what icon should be displayed for a file of a certain type and creator.

When there is more than one application present with the same creator, then the Finder launches the application with the most recent creation date when documents are double-clicked. Applications which are launched as the result of a user double-clicking a document icon are sent by the operating system what is referred to as a **high level event** or **Apple Event**, detailing which document or documents are to be manipulated.

A viral application, with a more recent creation date than the infected application, would be launched by the Finder before the target application, and would then have an opportunity to infect further applications (for example, any application currently running which has not already been infected, or perhaps by scanning the directory tree for uninfected applications, processing only a few directories at a time so as to avoid detection), or to perform other actions. The viral application would then pass control to the infected application. The Apple Event which details the documents to be processed can easily be passed on to the target application, so that there is little outward evidence that anything unusual has taken place.

This method of infection seems to function well on hard drives with single and multiple partitions, and should infect Macintosh file servers as well.

The ability of such a virus to conceal itself is limited, being restricted to setting the position of its document icon displayed by the Finder to some point off the edge of the display window.

Like companion viruses of DOS, this attack avoids altering any existing executable code or system resources.

9.2 Macro Viruses

A traditional virus can be quite difficult to write, as there are many factors which must be considered if the virus is to work successfully and be able to avoid detection on a wide range of systems running a variety of software products.

Simple macro viruses, however, are not hampered by many of the same considerations. As an exercise, a number of very simple macro viruses were implemented by Horton. A first attempt required only a single macro, **AutoClose**. This attempt was very crude and would be easily detected by an alert user, as infecting a document when it was closed (causing the **AutoClose** macro to run) required that the document be saved to disk with the viral macro attached.

A later, slightly more sophisticated offering was implemented using two macros, **AutoOpen** and **FileSaveAs**. The **AutoOpen** macro would be activated whenever an infected document is opened in Word. If Word's Normal template is not already infected, the virus then infects the Normal template, which is a global macro file visible by all documents, by copying the viral macros to the template.

From this point on, whenever the user selects **Save As ...** from the **File** menu, the **FileSaveAs** macro (now attached to the Normal template) is activated instead. It performs the same tasks as the usual menu option, with the exception that before the document is saved it is converted to a template to which the viral macros are then attached.

Neither of the two macro viruses created implemented any "payload" macros, although the addition of a simple payload would be trivial. The viruses are not remarkable for the techniques they employ, which are found in other macro viruses.

The first implementation attempt of a macro virus was accomplished in approximately eight hours by a computer science graduate previously unfamiliar with Microsoft Word and WordBasic. No documentation was available other than the "Help" system of Microsoft Word and a freely available description of some common macro viruses [15].

The second attempt required another hour, and was sufficiently sophisticated that it would be able to replicate unnoticed. A greater familiarity with WordBasic would be required to write anything more sophisticated, but the exercise illustrates the fact that macro viruses, because of their ease of construction, permit a much wider pool of computer users to write computer viruses in a very short space of time.

Macro viruses are quite widespread and so samples are often easy to obtain. Furthermore, understanding the code of a macro virus (assuming that the macros are not marked as *ExecuteOnly*) is very much less difficult than interpreting the assembly language in which non-macro viruses are typically implemented. The study of techniques used to implement other macro viruses is one straightforward method of improving virus writing.

More sophisticated macro viruses might target specific computer systems by attempting to "drop" a non-macro filesystem virus into the target system, or perform other operating system dependent actions. This would require greater familiarity with WordBasic or other macro languages than are demonstrated in a simple macro virus, such as the ones created as a part of this exercise.

10 Summary

This paper has presented a definition of computer viruses as the term is commonly used, as well as attempting to explain the meaning of some other terms, such as "worm", which are often closely associated with viruses. The various types of viruses that currently exist and various methods used by those viruses for infection have been explained, and the types and numbers of viruses likely to be found in the wild have been considered.

The various ways viruses currently use to hinder detection and delay analysis once they have been detected were discussed. Commonly available anti-virus measures were explained, as well as faults and problems that these methods are known to have.

Finally, some recent work on the ease of implementation of simple macro viruses and the implementation of a companion virus-type attack for the Macintosh was discussed.

References

[1] M. Bishop. An Overview of Computer Viruses in a Research Environment. Obtained from *ftp://ftp.informatik.uni-hamburg.de/pub/virus/texts/viruses/mallogic.zip.*

[2] V. Bontchev. Are Good Computer Viruses Still a Bad Idea? Obtained from *ftp://ftp.informatik.uni-hamburg.de/pub/virus/texts/viruses/goodvir.zip.*

[3] V. Bontchev. Possible Virus Attacks Against Integrity Programs and how to prevent them. In *Proc. Second International Virus Bulletin Conf.*, pages 131–141, September 1992. Obtained from *ftp://ftp.informatik.uni-hamburg.de/pub/virus/texts/viruses/attacks.zip.*

[4] F. B. Cohen. *A Short Course on Computer Viruses*, John Wiley & Sons, Inc., second edition, 1994.

[5] M. W. Eichin and J. A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. Obtained from *ftp://athena-dist.mit.edu:21/pub/virus/mit.PS.*

[6] H. J. Highland. A Macro Virus. *Computers & Security*, May 1989, pp. 178–182.

[7] T. A. Longstaff and E. E. Schultz. Beyond preliminary analysis of the WANK and OILZ worms: a case study of malicious code. *Computers & Security*, Volume 12, Number 1, pages 61–77, 1993.

[8] S. Magruder. High-level language computer viruses — a new threat? *Computers & Security*, Volume 13, Number 3, pages 263–269, 1994.

[9] Y. Radai. Integrity Checking for Anti-Viral Purposes Theory and Practice. December 1994. Obtained from *http://shum.cc.huji.ac.il/ radai/virus.htm*; an improved version of "Checksumming Techniques for Anti-Viral Purposes", *Proc. First International Virus Bulletin Conference*, September 1991.

[10] D. Seeley. A Tour of the Worm. Obtained from *http://www.ja.net/newsfiles/janinfo/cert/Seeley/tour.ps.*

[11] E. H. Spafford. The Internet Worm Program: An Analysis. Purdue Technical Report CSD-TR-823. Obtained from *http://www.ja.net/newsfiles/janinfo/cert/Spafford/Internet_worm.ps*

[12] E. H. Spafford. The Internet Worm Incident. Purdue Technical Report CSD-TR-933. Obtained from *http://www.ja.net/newsfiles/janinfo/cert/Spafford/IWorm2.ps*

[13] S. R. White. Covert Distributed Processing With Computer Viruses. In *Advances in Cryptology — Crypto '89 Proceedings*, pages 616–619, Springer-Verlag, 1990.

[14] A. Young. Cryptovirology: Extortion-Based Security Threats and Countermeasures. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy.* Obtained from *http://www.cs.columbia.edu:80/~ayoung/-iee96.ps.gz.*

[15] U.S. Department Of Energy's Computer Incident Advisory Capability Information Bulletin. G-10a: Winword Macro Viruses. Obtained from *http://ciac.llnl.gov/ciac/bulletins/g-10a.shtml.*

[16] Computer Virus Catalog (Macintosh Viruses). Obtained from *ftp://ftp.informatik.uni-hamburg.de/pub/virus/texts/catalog/macvir.zip.*

[17] Computer Virus Catalog (PC/MSDOS Viruses). Obtained from *ftp://ftp.informatik.uni-hamburg.de/pub/virus/texts/catalog/msdos.zip.*

[18] Various Contributors. *Virus-L/Comp.virus FAQ.* Obtained from *http://www.bocklabs.wisc.edu/~janda/virl_faq.html.*

[19] Various Contributors. *Alt.comp.virus FAQ.* Obtained from *http://www.bocklabs.wisc.edu/~janda/acv_faq.html.*

[20] Posting by Graham Cluley on 23 Jul 1996 in the USENET newsgroup `alt.comp.virus`.

[21] Posting by Brian Seborg on 15 August 1994 in the USENET newsgroup `comp.virus`. VIRUS-L Digest, Volume 7, Number 68. Obtained from *http://csrc.ncsl.nist.gov/virus/virusl/*)

[22] Virus Prevalence Table compiled using reports sent to and collected by *Virus Bulletin*. Found at *http://www.virusbtn.com/Prevalence/*