symantec.
press

The Art of Computer

Virus Research and Defense

"Of all the computer-related books I've read recently, this one influenced my thoughts about security the most. There is very little trustworthy information about computer viruses. Peter Szor is one of the best virus analysts in the world and has the perfect credentials to write this book."

—Halvar Flake, Reverse Engineer, SABRE Security GmbH

Peter Szor

Symantec Press Publisher: Linda McCarthy
Editor in Chief: Karen Gettman
Acquisitions Editor: Jessica Goldstein
Cover Designer: Alan Clements
Managing Editor: Gina Kanouse
Senior Project Editor: Kristy Hart
Copy Editor: Christal Andry
Indexers: Cheryl Lenser and Larry Sweazy
Compositor: Stickman Studio
Manufacturing Buyer: Dan Uhrig

Library of Congress Number: 2004114972

ISBN 0-32-130454-3

Text printed in the United States on recycled paper at Phoenix BookTech in Hagerstown, Maryland.
First printing, February, 2005

Preface

Who Should Read This Book

Over the last two decades, several publications appeared on the subject of computer viruses, but only a few have been written by professionals ("insiders") of computer virus research. Although many books exist that discuss the computer virus problem, they usually target a novice audience and are simply not too interesting for the technical professionals. There are only a few works that have no worries going into the technical details, necessary to understand, to effectively defend against computer viruses.

Part of the problem is that existing books have little—if any—information about the current complexity of computer viruses. For example, they lack serious technical information on fast-spreading computer worms that exploit vulnerabilities to invade target systems, or they do not discuss recent code evolution techniques such as code metamorphism. If you wanted to get all the information I have in this book, you would need to spend a lot of time reading articles and papers that are often hidden somewhere deep inside computer virus and security conference proceedings, and perhaps you would need to dig into malicious code for years to extract the relevant details.

I believe that this book is most useful for IT and security professionals who fight against computer viruses on a daily basis. Nowadays, system administrators as well as individual home users often need to deal with computer worms and other malicious programs on their networks. Unfortunately, security courses have very little training on computer virus protection, and the general public knows very little about how to analyze and defend their network from such attacks. To make things more difficult, computer virus analysis techniques have not been discussed in any existing works in sufficient length before.

I also think that, for anybody interested in information security, being aware of what the computer virus writers have "achieved" so far is an important thing to know.

For years, computer virus researchers used to be "file" or "infected object" oriented. To the contrary, security professionals were excited about suspicious events only on the network level. In addition, threats such as CodeRed worm appeared to inject their code into the memory of vulnerable processes over the network, but did not "infect" objects on the disk. Today, it is important to understand all of these major perspectives—the file (storage), in-memory, and network views—and correlate the events using malicious code analysis techniques.

During the years, I have trained many computer virus and security analysts to effectively analyze and respond to malicious code threats. In this book, I have included information about anything that I ever had to deal with. For example, I have relevant examples of ancient threats, such as 8-bit viruses on the Commodore 64. You will see that techniques such as stealth technology appeared in the earliest computer viruses, and on a variety of platforms. Thus, you will be able to realize that current rootkits do not represent anything new! You will find sufficient coverage on 32-bit Windows worm threats with in-depth exploit discussions, as well as 64-bit viruses and "pocket monsters" on mobile devices. All along the way, my goal is to illustrate how old techniques "reincarnate" in new threats and demonstrate up-to-date attacks with just enough technical details.

I am sure that many of you are interested in joining the fight against malicious code, and perhaps, just like me, some of you will become inventors of defense techniques. All of you should, however, be aware of the pitfalls and the challenges of this field!

That is what this book is all about.

What I Cover

The purpose of this book is to demonstrate the current state of the art of computer virus and antivirus developments and to teach you the methodology of computer virus analysis and protection. I discuss infection techniques of computer viruses from all possible perspectives: file (on storage), in-memory, and network. I classify and tell you all about the dirty little tricks of computer viruses that bad guys developed over the last two decades and tell you what has been done to deal with complexities such as code polymorphism and exploits.

The easiest way to read this book is, well, to read it from chapter to chapter. However, some of the attack chapters have content that can be more relevant after understanding techniques presented in the defense chapters. If you feel that any of the chapters are not your taste, or are too difficult or lengthy, you can always jump to the next chapter. I am sure that everybody will find some parts of this book very difficult and other parts very simple, depending on individual experience.

I expect my readers to be familiar with technology and some level of programming. There are so many things discussed in this book that it is simply impossible to cover everything in sufficient length. However, you will know exactly what you might need to learn from elsewhere to be absolutely successful against malicious threats. To help you, I have created an extensive reference list for each chapter that leads you to the necessary background information.

Indeed, this book could easily have been over 1,000 pages. However, as you can tell, I am not Shakespeare. My knowledge of computer viruses is great, not my English. Most likely, you would have no benefit of my work if this were the other way around.

What I Do Not Cover

I do not cover Trojan horse programs or backdoors in great length. This book is primarily about self-replicating malicious code. There are plenty of great books available on regular malicious programs, but not on computer viruses.

I do not present any virus code in the book that you could directly use to build another virus. This book is not a "virus writing" class. My understanding, however, is that the bad guys already know about most of the techniques that I discuss in this book. So, the good guys need to learn more and start to think (but not act) like a real attacker to develop their defense!

Interestingly, many universities attempt to teach computer virus research courses by offering classes on writing viruses. Would it really help if a student could write a virus to infect millions of systems around the world? Will such students know more about how to develop defense better? Simply, the answer is no...

Instead, classes should focus on the analysis of existing malicious threats. There are so many threats out there waiting for somebody to understand them—and do something against them.

Of course, the knowledge of computer viruses is like the "Force" in *Star Wars*. Depending on the user of the "Force," the knowledge can turn to good or evil. I cannot force you to stay away from the "Dark Side," but I urge you to do so.

CHAPTER 3

Malicious Code Environments

"In all things of nature there is something of the marvelous."
—Aristotle

One of the most important steps toward understanding computer viruses is learning about the particular execution environments in which they operate. In theory, for any given sequence of symbols we could define an environment in which that sequence could replicate itself. In practice, we need to be able to find the environment in which the sequence of symbols operates and prove that it uses code explicitly to make copies of itself and does so recursively[1].

A successful penetration of the system by viral code occurs only if the various dependencies of malicious code match a potential environment. Figure 3.1 is an imperfect illustration of common environments for malicious code. A perfect diagram like this is difficult to draw in 2D form.

The figure shows that Microsoft Office itself creates a homogeneous environment for malicious code across Mac and the PC. However, not all macro viruses[2] that can multiply on the PC will be able to multiply on the Mac because of further dependencies. Each layer might create new dependencies (such as vulnerabilities) for malicious code. It is also interesting to see how possible developments of .NET on further operating systems, such as Linux, might change these dependency points and allow computer viruses to jump across operating systems easily. Imagine that each ring in Figure 3.1 has tiny penetration holes in it. When the holes on all the rings match the viral code and all the dependencies are resolved, the viral code successfully infects the system.

Figure 3.1 suggests how difficult virus research has become over the years. With many platforms already invaded by viruses, the fight against malicious code gets more and more difficult.

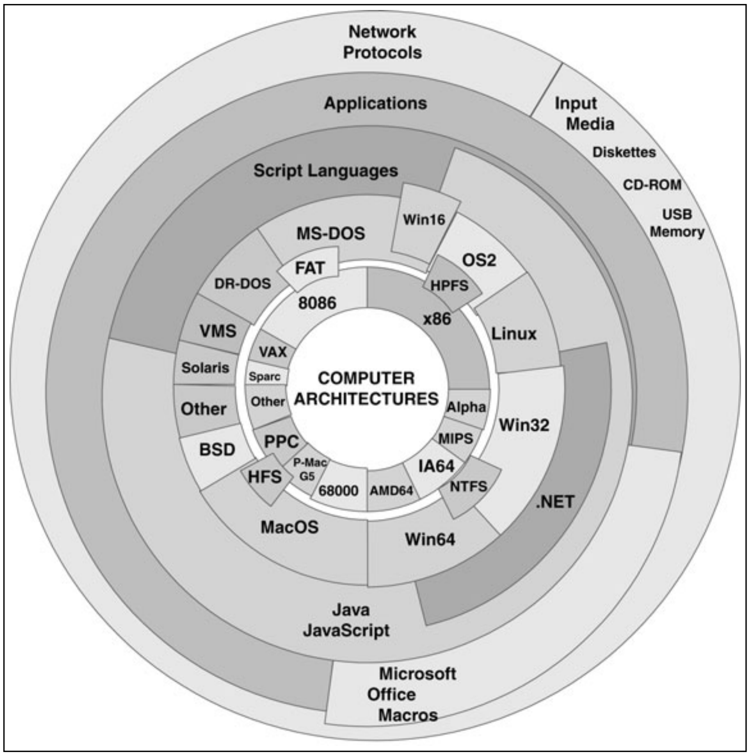


Figure 3.1 Common environments of malicious code.

Please note that I am not suggesting that viruses would need to exploit systems. An exploitable vulnerability is just one possible dependency out of many examples.

Automation of malicious code analysis has also become increasingly more difficult because of diverse environment dependency issues. It is not uncommon to spend many hours with a virus in a lab environment, attempting natural replication, but without success, while the virus is being reported from hundreds or perhaps even thousands of systems around the world.

Another set of viruses could be so unsuccessful that a researcher could never manage to replicate them. Steve White of IBM Research once said that he could give a copy of the Whale virus ("the mother of all viruses") to everybody in the audience, and it would still not replicate[3]. However, it turns out that Whale has an interesting dependency on early 8088 architectures[4] on which it works perfectly. Even more interestingly, this dependency disappears on Pentium and above processors[5]. Thus Whale, "the dinosaur heading for extinction,"[6] is able to return, theoretically, in a *Jurassic Park*–like fashion.

One of the greatest challenges facing virus researchers is the need to be able to recognize the types, formats, and sequences of code and to find its environment. A researcher can only analyze the code according to the rules of its environment and prove that the sequence of code is malicious in that environment.

Over the years, viruses have appeared on many platforms, including Apple II, C64, Atari ST, Amiga, PC, and Macintosh, as well as mainframe systems and handheld systems such as the PalmPilot[7], Symbian phones, and the Pocket PC. However, the largest set of computer viruses exists on the IBM PC and its clones.

In this chapter, I will discuss the most important dependency factors that computer viruses rely on to replicate. I will also demonstrate how computer viruses unexpectedly evolve, devolve, and mutate, caused by the interaction of virus code with its environment.

3.1 Computer Architecture Dependency

Most computer viruses do spread in executable, binary form (also called compiled form). For instance, a boot virus will replicate itself as a single or couple of sectors of code and takes advantage of the computer's boot sequence. Among the very first documented virus incidents was Elk Cloner on the Apple II, which is also a boot virus. Elk Cloner modified the loaded operating system with a hook to itself so that it could intercept disk access and infect newly inserted disks by overwriting their system boot sectors with a copy of its own code and so on. Brain, the oldest known PC computer virus, was a boot sector virus as well, written in 1986. Although the boot sequences of the two systems as well as the structures of these viruses show similarities, viruses are highly dependent on the particularities of the architecture itself (such as the CPU dependency described later on in this chapter) and on the exact load procedure and memory layout. Thus, binary viruses typically depend on the computer architecture. This explains why one computer virus for an Apple II is generally unable to infect an IBM PC and vice versa.

In theory, it would be feasible to create a multi-architecture binary virus, but this is no simple task. It is especially hard to find ways to execute the code made for one architecture to run on another. However, it is relatively easy to code to two independent architectures, inserting the code for both in the same virus. Then the virus must make sure that the proper code gets control on the proper architecture. In March of 2001, the PeElf virus proved that it was possible to create a cross-platform binary virus.

Virus writers found another way to solve the multi-architecture and operating system issue by translating the virus code to a pseudoformat and then translating it to a new architecture. The Simile.D virus (also known as Etap.D) of Mental Driller uses this strategy to spread itself on Windows and Linux systems on 32-bit Intel (and compatible) architectures.

It is interesting to note that some viruses refrain from replication in particular environments. Such an attempt was first seen in the Cascade virus, written by a German programmer in 1987. Cascade was supposed to look at the BIOS of the system, and if it found an IBM copyright, it would refrain from infecting. This part of the virus had a minor bug, so the virus infected all kinds of systems. Its author repeatedly released new versions of the virus to fix this bug, but the newer variants also had bugs in this part of the code[8].

Another kind of computer virus is dependent on the nature of BIOS updating. On so-called flashable or upgradeable BIOS systems, BIOS infection is feasible. There have been published attempts to do this by the infamous Australian virus-writer group called VLAD.

3.2 CPU Dependency

CPU dependency affects binary computer viruses. The source code of programs is compiled to object code, which is linked in a binary format such as an EXE (executable) file format. The actual executable contains the "genome" of a program as a sequence of instructions. The instructions consist of opcodes. For instance, the instruction NOP (no operation) has a different opcode on an Intel x86 than on a VAX or a Macintosh. On Intel CPUs, the opcode is defined as 0x90. On the VAX, this opcode would be 0x01.

Thus the sequences of bytes most likely translate to garbage code from one CPU to another because of the differences between the opcode table and the operation of the actual CPU. However, there are some opcodes that might be used as meaningful code on both systems, and some viruses might take advantage of this. Most computer viruses that are compiled to binary format will be CPU-dependent and unable to replicate on a different CPU architecture.

There is yet another form of CPU dependency that occurs when a particular processor is not 100% backward compatible with a previous generation and does not support the features of another perfectly or at all. For example, the Finnpoly virus fails to work on 386 processors because the processor incorrectly executes the instruction CALL SP (make a call according to the Stack Pointer). Because the virus transfers control to its decrypted code on the stack using this instruction, it hangs the machine when an infected file is executed on a 386 processor. In addition, a similar error appeared in Pentium processors as well[9]. Another example is the Cyrix 486 clones, which have a bug in their single-stepping code[10]. Single-stepping is used by tunneling viruses (see Chapter 6, "Basic Self-Protection Strategies") such as Yankee_Doodle, thus they fail to work correctly on the bogus processors.

Note

It is not an everyday discovery to find a computer virus that fails because of a bug in the processor.

Some viruses use instructions that are simply no longer supported on a newer CPU. For instance, the 8086 Intel CPU supported a POP CS instruction, although Intel did not document it. Later, the instruction opcode (0x0f) was used to trap into multibyte opcode tables. A similar example of this kind of dependency is the MOV CS, AX instruction used by some early computer viruses, such as the Italian boot virus, Ping Pong:

```
Opcode      Assembly     Instruction
8EC8        MOV          CS,AX
0E          PUSH         CS
1F          POP          DS
```

Other computer viruses might use the coprocessor or MMX (Multimedia Extensions) or some other extension, which causes them to fail when they execute on a machine that does not support them.

Some viruses use analytical defense techniques based on altering the processor's prefetch queue. The size of the prefetch queue is different from processor to processor. Viruses try to overwrite code in the next instruction slot, hoping that such code is already in the processor prefetch queue. Such modification occurs during debugging of the virus code; thus, novice virus code analysts are often unable to analyze such viruses. This technique is also effective against early code emulation–based heuristics scanners. However, the disadvantage of such virus code is that it might become incompatible with certain kinds of processors or even operating systems.

3.3 Operating System Dependency

Traditionally, operating systems were hard-coded to a particular CPU architecture. Microsoft's first operating systems, such as MS-DOS, supported Intel processors only. Even Microsoft Windows supported only Intel-compatible hardware. However, in the '90s the need to support more CPU architectures with the same operating system was increasing. Windows NT was Microsoft's first operating system that supported multiple CPU architectures.

Most computer viruses can operate only on a single operating system. However, cross-compatibility between DOS, Windows, Windows 95/98, and Windows NT/2000/XP still exists on the Intel platforms even today. Thus, some of the viruses that were written for DOS can still replicate on newer systems. We tend to use less and less old, "authentic" software, however, thus reducing the risk of such infections. Furthermore, some of the older tricks of computer viruses will not work in the newer environments. On Windows NT, for example, port commands cannot be used directly to access the hardware from DOS programs. As a result, all DOS viruses that use direct port commands will fail at some point because the operating system generates an error. This might prevent the replication of the virus altogether if the port commands (IN/OUT operations) occur before the virus multiplies itself.

A 32-bit Windows virus that will infect only portable executable (PE) files will not be able to replicate itself on DOS because PE is not a native file format of DOS and thus will not execute on it. However, so-called multipartite viruses are able to infect several different file formats or system areas, enabling them to jump from one operating environment to another. The most important environmental dependency of binary computer viruses is the operating system itself.

3.4 Operating System Version Dependency

Some computer viruses depend not only on a particular operating system, but also on an actual system version. Young virus researchers often struggle to analyze such a virus. After a few minutes of unsuccessful test infections on their research systems, they might believe that a particular virus does not work at all. Especially at the beginning of a particular computer virus era, we can see a flurry of computer viruses repeating the same mistakes that make them dependent on some flavor of Windows. For example, the W95/Boza virus does not work on non-English releases of Windows 95, such as the Hungarian release of the operating system.

This leads to the discovery that computer viruses might be used to target the computers of one particular nation more than others. For example, Russian Windows systems can be different enough from U.S. versions to become recognizable, enabling the author of a virus, intentionally or unintentionally, to target only a subset of computer users. In general, however, after a virus has been created, its author has very little or no control over exactly where his or her creation will travel.

3.5 File System Dependency

Computer viruses also have file system dependencies. For most viruses, it does not matter whether the targeted files reside on a File Allocation Table (FAT), originally used by DOS; the New Technology File System (NTFS), used by Windows NT; or a remote file system shared across network connections. For such viruses, as long as they are compatible with the operating environment's high-level file system interface, they work. They will simply infect the file or store new files on the disk without paying attention to the actual storage format. However, other kinds of viruses depend strongly on the actual file system.

3.5.1 Cluster Viruses

Some successful viruses can spread only on a specific file system. For instance, the Bulgarian virus, DIR-II, is a so-called cluster virus, written in 1991. DIR-II has features specific to certain DOS versions but, even more importantly, spreads itself by manipulating key structures of FAT-based file systems. On FAT on a DOS system, direct disk access can be used to overwrite the pointer (stored in the directory entry) to the first cluster on which the beginning of a file is stored.

Files are stored on the disk as clusters, and the FAT is used by DOS to put the puzzle pieces together. The DIR-II virus overwrites the pointer in the directory entry that points to the first cluster of a file with a value that directs the disk-read to the virus body, which has been stored at the end of the disk. The virus stores the pointer to the real first cluster of each host program in an encrypted form, in an unused part of the directory entry structure. This is used later to execute the real host from the disk after the virus has been loaded in memory. In fact, when the virus is active in memory, the disk looks normal and files execute normally.

Such viruses infect programs extremely quickly because they only manipulate a few bytes in the directory entries on the disk. These viruses are often called *"super fast" infectors*[1]. It is important to understand that there is only one copy of

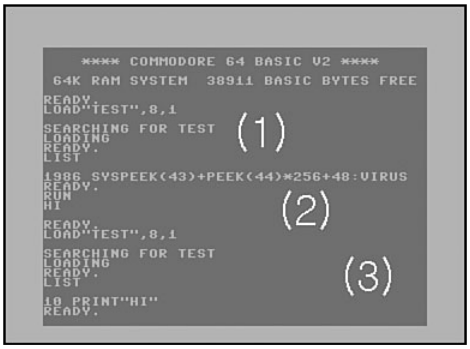
DIR-II on each infected disk. Consequently, when DIR-II is not active in memory, the file system appears "cross-linked" because all infected files point to the same start cluster: the virus code.

A similar cluster infection technique appeared in the BHP virus on the Commodore 64 in Germany, written by "DR. DR. STROBE & PAPA HACKER" in circa 1986[11]. This virus manipulates with the block entries of host programs stored on Commodore floppy diskettes. I decided to call this special infection technique the *cluster prepender* method. Let me tell you a little bit more about this ancient creature.

Normally, the Commodore 1541 floppy drive can store up to 166KB on each side of a diskette. The storage capacity of each diskette side is split into 664 "blocks" that are 256 bytes each. When BHP infects a program on the diskette, the virus will attempt to occupy eight free blocks for itself. Next, it replaces the "block" pointer in the first block of the host program to point to the virus code instead. Except for the first block, the host program's code will not be moved on the diskette. Instead, the virus will link its own "blocks" with the "blocks" of the host program as a single cluster of blocks. The infected host program will be loaded with the virus in front. Unlike the DIR-II virus, the BHP virus has multiple copies per diskettes. In each infection, eight blocks of free space will be lost on the diskette, but the infected files will not appear to be larger in a directory listing even if the virus is not active in memory.

Figure 3.2(1) shows when a BHP-infected program called TEST is loaded for the first time with a LOAD command. When I list the content of the loaded program with the LIST command, a BASIC command line appears as shown in Figure 3.2(2). This SYS command triggers the binary virus code. When I execute the infected program with the RUN command, the 6502 Assembly-written virus gets control. On execution of the virus code, BHP becomes active in memory. Finally, the virus runs the original host program. Figure 3.2(2) shows that a "HI" message is displayed when the loaded virus is executed. This message is displayed by the host program.

When BHP virus is active in memory it becomes stealth just like the DIR-II virus. As shown in Figure 3.2(3), I load the infected TEST program a second time. When I list the content of the program, I see the original host program, a single PRINT command that displays "HI." Thus, the virus is already stealth; as long as the virus code is active in memory, the original content of the program is shown instead of the infected program. In addition, the BHP virus implements a set of basic self-protection tricks. For example, the virus disables restart and reset attempts to stay active in memory. Moreover, BHP uses a self checksum function to check if its binary code was modified or corrupted. As a result, a trivially modified or corrupted virus code will intentionally fail to run.



```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
LOAD"TEST",8,1

SEARCHING FOR TEST
LOADING
READY.
LIST

1986 SYSPEEK(43)+PEEK(44)*256+48:VIRUS
READY.
RUN
HI

READY.
LOAD"TEST",8,1

SEARCHING FOR TEST
LOADING
READY.
LIST

10 PRINT"HI"
READY.
```

Figure 3.2 The BHP virus on Commodore 64.

3.5.2 NTFS Stream Viruses

FAT file systems are simple but very inefficient for larger hard disks (in FAT terms, a drive of several Gigabytes is considered very large). Operating systems such as Windows NT demanded modern file systems that would be fast and efficient on large disks and, more importantly, on the large disk arrays that span many Terabytes, as used in commercial databases.

To meet this need, the NTFS (NT file system) was introduced. A little-known feature of NTFS is primarily intended to support the multiple-fork concept of Apple's Hierarchical File System (HPS). Windows NT had to support multiple-fork files because the server version was intended to service Macintosh computers. On NTFS, a file can contain multiple streams on the disk. The "main stream" is the actual file itself. For instance, notepad.exe's code can be found in the main stream of the file. Someone could store additional named streams in the same file; for instance, the notepad.exe:test stream name can be used to create a stream name called *test*. When the WNT/Stream[12] virus infects a file, it will overwrite the file's main stream with its own code, but first it stores the original code of the host in a named stream called *STR*. Thus WNT/Stream has an NTFS file system dependency in storing the host program.

Malicious hackers often leave their tools behind in NTFS streams on the disk. Alternate streams are not visible from the command line or the graphical file manager, Explorer. They generally do not increment the file size in the directory entries, although disk space lost to them might be noticed. Furthermore, the content of the alternate streams can be executed directly without storing the file content in a main stream. This allows the potential for sophisticated NTFS worms in the future.

3.5.3 NTFS Compression Viruses

Some viruses attempt to use the compression feature of the NTFS to compress the host program and the virus. Such viruses use the DeviceIoControl() API of Windows and set the FSCTL_SET_COMPRESSION control mode on them. Obviously, this feature depends on an NTFS and will not work without it. For example, the W32/HIV virus, by the Czech virus writer, Benny, depends on this. Some viruses also use NTFS compression as an infection marker, such as the WNT/Stream virus.

3.5.4 ISO Image Infection

Although it is not a common technique, viruses also attack image file formats of CD-ROMs, such as the ISO 9660, which defines a standard file system. Viruses can infect an ISO image before it is burnt onto a CD. In fact, several viruses got wild spread from CD-R disks, which cannot be easily disinfected afterwards. ISO images often have an AUTORUN.INF file on them to automatically lunch an executable when the CD-ROM is used on Windows. Viruses can take advantage of this file within the image and modify it to run an infected executable. This technique was developed by the Russian virus writer, Zombie, in early 2002.

3.6 File Format Dependency

Viruses can be classified according to the file objects they can infect. This short section is an introduction to binary format infectors. Many of the techniques are detailed further in Chapter 4, "Classification of Infection Strategies."

3.6.1 COM Viruses on DOS

Viruses such as Virdem and Cascade only infect DOS binary files that have the COM extension. COM files do not have a specific structure; therefore, they are easy targets of viruses. Dozens of variations of techniques exist to infect COM files.

3.6.2 EXE Viruses on DOS

Other viruses can infect DOS EXE files. EXE files start with a small header structure that holds the entry point of the program among other fields. EXE infector viruses often modify the entry point field of the host and append themselves to the end of the file. There are more techniques for infecting EXE files than for infecting COM files because of the format itself.

EXE files start with an MZ identifier, a monogram of the Microsoft engineer, Mark Zbikowski, who designed the file format. Interestingly, some DOS versions accept either MZ or ZM at the front of the file. This is why some of the early Bulgarian DOS EXE viruses infect files with both signatures in the front. If a scanner recognizes EXE files based on the MZ signature alone, it might have a problem detecting a virus with a ZM signature. Some tricky DOS viruses replace the MZ mark with ZM to avoid detection by antivirus programs, and yet others have used ZM as an infection marker to avoid infecting the file a second time.

Disinfecting EXE files is typically more complicated than disinfecting a COM file. In principle, however, the techniques are similar. The header information, just like the rest of the executable, must be restored, and the file must be truncated properly (whenever needed).

3.6.3 NE (New Executable) Viruses on 16-bit Windows and OS/2

One of the first viruses on Windows was W16/Winvir. Winvir uses DOS interrupt calls to infect files in the Windows NE file format. This is because early versions of Windows use DOS behind the scene. NE files are more complicated in their structure than EXE files. Such NE files start with an old DOS EXE header at the front of the file, followed by the new EXE header, which starts with an NE identifier.

One of the most interesting NE virus infection techniques was developed in the W16/Tentacle_II family, which was found in the wild in June 1996 in the U.S., U.K., Australia, Norway, and New Zealand. Not only was Tentacle_II in the wild, but it was also rather difficult to detect and repair because it took advantage of the complexity of the NE file format. This virus is discussed further in Chapter 4.

3.6.4 LX Viruses on OS/2

Linear eXecutables (LXs) were also introduced in later versions of OS/2. Not many viruses were ever implemented in them, but there are a few such creations. For instance, OS2/Myname is a very simple overwriting virus.

Myname uses a couple of system calls, such as DosFindFirst(), DosFindNext(), DosOpen(), DosRead(), and DosWrite(), to locate executables and then overwrites them with itself. The virus searches for files with executable extensions in the current directory. It does not attempt to identify OS/2 LX files for infection; it simply overwrites any files with its own copy. Nonetheless, OS2/Myname is dependent on the LX file format and OS/2 environment for execution given that the virus itself is an LX executable.

The OS2/Jiskefet version of the virus also overwrites files to spread itself. This virus looks specifically for files with a New Executable header that starts with the LX mark:

```
cmp      word ptr [si], 'XL'
jnz      NO
```

The header of the file is loaded by the virus, and the si (source index) register is used as an index to check for the mark. If the marker is missing, the virus will not overwrite the file. As a result, Jiskefet is more dependent on the LX file format than Myname.

3.6.5 PE (Portable Executable) Viruses on 32-bit Windows

The first virus known to infect PE files was W95/Boza, written by members of the Australian virus-writing group, VLAD, for the beta version of Windows 95.

The virus was named Bizatch by its authors but got its current name, Boza, from Vesselin Bontchev. He called the virus Boza, referring to a bizarre Bulgarian drink with color and consistency of mud that is disliked by most non-Bulgarians. Bontchev picked the name not only because Boza sounds similar to "Bizatch," but also because the virus was "buggy and messily written." The Bulgarian idiom, "This is a big boza," means "this is extremely messy and unclear."

Quantum, the virus writer, was unhappy about this, which was Bontchev's intention in choosing the name. In fact, other viruses attacked antivirus software databases to change the name of Boza to Bizatch so that the original name would be displayed when an antivirus program detected it. This illustrates the psychological battle waged between virus writers and antivirus researchers.

Because PE file infection is currently one of the most common infection techniques, I will provide more information about it in Chapter 4. Many binary programs use the PE file format, including standard system components, regular applications, screen-saver files, device drivers, native applications, dynamic link libraries, and ActiveX controls.

The new 64-bit PE+ files are already supported by 64-bit architectures, such as IA64, AMD64, and EM64T. Computer virus researchers expected that 64-bit

Windows viruses will appear to infect this format correctly with native 64-bit virus code.

The W64/Rugrat.3344[13] virus appeared in May 2004, written by the virus writer "roy g biv." Rugrat is written in IA64 Assembly. The virus is very compact—about 800 lines. Rugrat utilizes modern features of the Itanium processor, such as code predication. In addition, roy g biv released the W64/Shruggle virus during the summer of 2004. W64/Shruggle infects PE+ files that run on the upcoming 64-bit Windows on AMD64.

3.6.5.1 Dynamic Link Library Viruses

The W95/Lorez virus was one of the first 32-bit Windows viruses that could infect a dynamic link library (DLL). A Windows DLL uses the same basic file format as regular PE executables. Dynamic linked libraries export functions that other applications can use.

The interface between applications and dynamic link libraries is facilitated by exports from DLLs and imports into the executables. Lorez simply infects the user mode KERNEL client component, KERNEL32.DLL. By modifying the DLL's export directory, such viruses can hook an API interface easily.

DLL infection became increasingly successful with the appearance of the Happy99 worm (also known as W32/SKA.A, the worm's CARO name), written by Spanska in early 1999. Figure 3.3 is a capture of Happy99's fireworks payload.

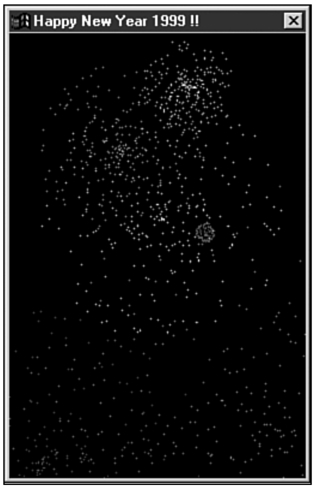


Figure 3.3 The Happy99 worm's payload.

Just as many other worms are linked to holidays, this worm took advantage of the New Year's period by mimicking an attractive New Year's card application.

Happy99 injected a set of hooks into the WSOCK32.DLL library, hooking the connect() and send() APIs to monitor access to mail and newsgroups.

Happy99 started a debate about computer malware classifications by carrying the following message for researchers:

```
Is it a virus, a worm, a trojan? MOUT-MOUT Hybrid (c) Spanska 1999.
```

3.6.5.2 Native Viruses

Recently, a new kind of 32-bit Windows virus is on the rise: native infectors. The first such virus, W32/Chiton, was created by the virus writer, roy g biv, in late 2001. Unlike most Win32 viruses, which depend on calling into the Win32 subsystem to access API functions to replicate, W32/Chiton can also replicate outside of the Win32 subsystem.

A PE file can be loaded as a device driver, a GUI Windows application, a console application, or a native application. Native applications, such as autochk.exe, load during boot time. Because they load before subsystems are available, they are responsible for their own memory management. In their file headers, the PE.OptionalHeader.Subsystem value is set to 0001 (Native).

The HKLM\System\CurrentControlset\Control\Session Manager\BootExecute value contains the names and arguments of native applications that are executed by the Session Manager at boot time. The Session Manager looks for such applications in the Windows\System32 directory, with the native executable names specified.

Native applications use the NTDLL.DLL (Native API), where hundreds of APIs are stored and remain largely undocumented by Microsoft. Native applications do not rely on the subsystem DLLs, such as KERNEL32.DLL, as these DLLs are not yet loaded when native applications load. There are only a handful of APIs that a computer virus needs to be able to call from NTDLL.DLL, and virus writers have already discovered the interface for these functions and their parameters.

W32/Chiton relies on the following NTDLL.DLL APIs for memory, directory, and file management:

1. Memory management:

```
RtlAllocateHeap()
RtlFreeHeap()
```

2. Directory and file search:

```
RtlSetCurrentDirectory_U()
RtlDosPathNameToNtPathName_U()
NtQueryDirectoryFile()
```

3. File management:

```
NtOpenFile()
NtClose()
NtMapViewOfSection()
NtUnmapViewOfSection()
NtSetInformationFile()
NtCreateSection()
```

Native viruses can load very early in the boot process, which gives them great flexibility in infecting applications. Such viruses are similar in structure to kernel-mode viruses. Therefore, it is expected that kernel-mode and native infection techniques will be combined in the future.

3.6.6 ELF (Executable and Linking Format) Viruses on UNIX

Viruses are not unknown on UNIX and UNIX-like operating systems, which generally use the ELF executable file format[14]. Typically, ELF files do not have any file extensions, but they can be identified based on their internal structure.

Just like PE files, ELF files can support more than one CPU platform. Moreover, ELF files can properly support 32-bit as well as 64-bit CPUs in their original design, unlike PE files, which needed some minor updates to make them compatible with 64-bit environments (resulting in the PE+ file format).

ELF files contain a short header, and the file is divided into logical sections. Viruses that spread on Linux systems typically target this format. Most Linux viruses are relatively simple[15]. For instance, the Linux/Jac.8759 virus can only infect files in the current folder.

One of the most complex Linux viruses is {W32,Linux}/Simile.D (also known as Etap.D), which was the first entry-point obscuring Linux virus (more on this in Chapter 4). Of course, Simile.D's success will depend on how well security settings are used in the file system. Writeable files will be infected; however, the virus does not elevate privileges to infect files.

It seems likely that future computer worm attacks (such as Linux/Slapper) will be combined with ELF infection on Linux. The elevated privileges often gained by exploiting network services can result in better access to binary files.

The main problem for ELF viruses is the missing binary compatibility between various flavors of UNIX systems. The diversification of the binaries on various

CPUs introduces library dependency. Because of this, many ELF-infecting viruses suffer serious problems and crash with core dumps rather than causing infections.

3.6.7 Device Driver Viruses

Device driver infectors were not very common in the DOS days, although virus writer magazines such as *40Hex* dedicated early articles to the subject. Device drivers for popular operating systems tend to have their own binary format, but as these are special forms of the more general executable formats for those platforms, all can be infected with known virus infection techniques. For example, 16-bit Windows drivers must be in the LE (linear executable) format. LE is very similar to the OS/2 LX file format. Of course, viruses can infect such files, too.

On Windows 9x, the VxD (virtual device driver) file format was never officially documented by Microsoft for the general public's use. As a result, only a few viruses were created that could infect VxD files. For example, W95/WG can infect VxD files and modify their entry point to run an external file each time the infected VxD is loaded. Consequently, only the entry-point code of the VxD is modified to load the virus code from the external source.

Other viruses, such as the W95/Opera family, infect VxD files by appending the virus code to the end of the file and modifying the real mode entry point of the VxD to run themselves from it.

Recently, device driver infectors appeared on Windows XP systems. On NT-based systems, device drivers are PE files that are linked to NT kernel functions. The few such viruses that exist today hook the INT 2E (System Service on IA32-based NT systems) interrupt handler directly in kernel mode to infect files on the fly. For example, WNT/Infis and W2K/Infis families can infect directly in Windows NT and Windows 2000 kernel mode. The W32/Kick virus was created by the Czech virus writer, Ratter, in 2003. W32/Kick infects only SYS files in the PE device driver format. The virus loads itself into kernel mode memory but runs its infection routine in user mode to infect files through the standard Win32 API.

Note

More information about in-memory strategies of computer viruses is available in Chapter 5, "Classification of In-Memory Strategies."

3.6.8 Object Code and LIB Viruses

Object and LIB infections are not very common. There are only about a dozen such viruses because they tend to be dependent on developer environments.

Source code is first compiled to object code, and then it is linked to an executable format:

```
Source Code - Object code / Library code - Executable.
```

Viruses that attack objects or libraries can parse the object or library format. For instance, the Shifter virus[16] can infect object files. Such viruses spread in a couple of stages as shown in Figure 3.4.

Stage 1: Infected executable is run on the host.

Stage 2: Virus code locates new object files and infects them.

Stage 3: The object files or libraries are linked by the user as part of a new project. (Repeat Stage 1.)

Figure 3.4 The infection stages of the Shifter virus.

Shifter was written by Stormbringer in 1993. The virus carefully checks whether an object file is ready to be linked to a COM, DOS executable. This is done by checking the Data Record Entry offset of object files. If this is 0x100, the virus attempts to infect the object in such a way that once the object is linked, it will be in the front of the COM executable.

3.7 Interpreted Environment Dependency

Several virus classes depend on some sort of interpreted environment. Almost every major application supports users with programmability. For example, Microsoft Office products provide a rich programmable macro environment that uses Visual Basic for Applications. (Older versions of Word, specifically Word 6.0/Word 95, use WordBasic.) Such interpreted environments often enhance viruses with multi-platform capabilities.

3.7.1 Macro Viruses in Microsoft Products

Today there are thousands of macro viruses, and many of them are in the wild. Users often exchange documents that were created with a Microsoft Office product, such as Word, Excel, PowerPoint, Visio, or even Access or Project. The first wild-spread macro virus, WM/Concept.A[17], appeared in late 1995. Within a couple of months, only a few dozen such viruses were found, but by 1997 there were thousands of similar creations. The XM/Laroux[18], discovered in 1996, was the first wild-spread macro virus to infect Excel spreadsheets. The first known Word macro virus was WM/DMV, written in 1994. The author of the WM/DMV virus also created a nearly functional Excel macro (XM) virus at the same time.

Figure 3.5 illustrates a high-level view of an OLE2 file used by Microsoft products. Microsoft does not officially document the file structure for the public.

Please note that Microsoft products do not directly work with OLE2 files. As a result, technically a macro virus in any such Microsoft environments does not directly infect an OLE2 file because Microsoft products access these objects through the OLE2 API. Also note that different versions of such Microsoft programs use different languages or different versions of such languages.

In the front of OLE2 files, you can find an identifier, a sequence of hex bytes "D0 CF 11 E0," which looks like the word DOCFILE in hex bytes (with a lowercase L). These bytes can appear in both big-endian and little-endian formats. Other values are supported by various beta versions of Microsoft Office products. The header information block contains pointers to important data structures in the file. Among many important fields, it contains pointers to the FAT and the Directory. Indeed, the OLE2 file is analogous to MS-DOS FAT-based storage. The problem is that OLE2 files have an extremely complex structure. They are essentially file systems in a file with their own clusters, file allocation table, root directory, subdirectories (called "storages"), files (called "streams"), and so on.

The basic sector size is 512 bytes, but larger values are also allowed. (In some implementations, a mini-FAT[19] allows even shorter "sector" sizes.) Office products locate macros by looking in the Directory of an OLE2 file for the VBA storage folder. The macros appear as streams inside the document. Obviously, any objects can get fragmented, as in a real file system—corruptions of all kinds are also possible, including circular FAT or Directory entries, and so on. Unfortunately, even macros can get corrupted; as you will see, this fact contributes to the natural creation of new macro virus variants.

In addition, documents have a special bit inside, the so-called template bit. WinWord 6/7 does not look for macros if the template bit is off[20].

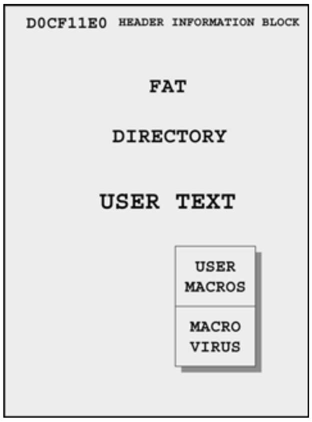


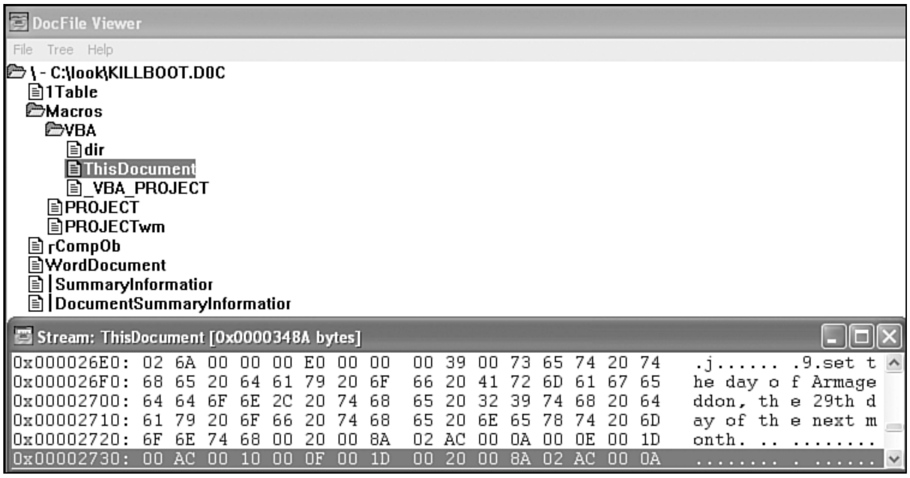
Figure 3.5 A high-level view of the OLE2 file format.

Macro viruses are stored inside the document instead of at the front or at the very end of the file. Even worse, the macros are buried inside some of the streams, and the streams themselves have a very complex structure. When looking at the physical OLE2 document, without understanding its structure, the (otherwise logically continuous) body of a macro of a macro virus could be split into chunks—some of them as small as 64 bytes.

A major challenge is the protection of user macros in the documents during the removal of virulent macros. In some cases, it is simply impossible to remove a macro virus safely without also removing the user macros. Obviously, users prefer to keep their own macros and remove the viruses from them, but such acrobatics are not always possible.

Macro viruses are much easier to create than other kinds of file infectors. Furthermore, the source of the virus code is available to anybody with the actual infection. Although this greatly simplifies the analysis of macro viruses, it also helps attackers because the virus source code can be accessed and modified easily.

To understand the internal structure of OLE2 documents better, look at a comment fraction of the W97M/Killboot.A virus in Microsoft's DocFile Viewer application, shown in Figure 3.6. DocFile Viewer is available as part of Microsoft Visual C++ 6.0. This tool can be used to browse the document storage and find the "ThisDocument" stream in the Macros\VBA directory.



```
DocFile Viewer
File  Tree  Help
\ - C:\look\KILLBOOT.DOC
  1Table
  Macros
    VBA
      dir
      ThisDocument
      _VBA_PROJECT
    PROJECT
    PROJECTwm
  rCompOb
  WordDocument
  |SummaryInformatior
  |DocumentSummaryInformatior

Stream: ThisDocument [0x0000348A bytes]
0x000026E0: 02 6A 00 00 00 E0 00 00  00 39 00 73 65 74 20 74  .j...... .9.set t
0x000026F0: 68 65 20 64 61 79 20 6F  66 20 41 72 6D 61 67 65  he day o f Armage
0x00002700: 64 64 6F 6E 2C 20 74 68  65 20 32 39 74 68 20 64  ddon, th e 29th d
0x00002710: 61 79 20 6F 66 20 74 68  65 20 6E 65 78 74 20 6D  ay of th e next m
0x00002720: 6F 6E 74 68 00 20 00 8A  02 AC 00 0A 00 0E 00 1D  onth. .. ........
0x00002730: 00 AC 00 10 00 0F 00 1D  00 20 00 8A 02 AC 00 0A  ........ . ......
```

Figure 3.6 The W97M/Killboot.A virus in DocFile Viewer.

The ThisDocument stream can be further browsed to find the virus code. In Figure 3.6, a comment by the virus writer can be seen encoded as VBA code:

```
E0 00 00 00 39 00 73 65 74 20 74 68 65 20 64 61 ....9.set the da
79 20 6F 66 20 41 72 6D 61 67 65 64 64 6F 6E 2C y of Armageddon,
20 74 68 65 20 32 39 74 68 20 64 61 79 20 6F 66  the 29th day of
20 74 68 65 20 6E 65 78 74 20 6D 6F 6E 74 68 00  the next month.
```

The 0xE0 opcode is used for comments. The 0x39 represents the size of the comment. Thus the preceding line translates to

```
'set the day of Armageddon, the 29th day of the next month
```

The opcode itself is VBA version-specific, so the 0xE0 byte can change to other values, resulting in Word up-conversion and down-conversion issues[21].

One of the most interesting aspects of macro viruses is that they introduced a new set of problems not previously seen in such quantities with any other type of computer virus.

3.7.1.1 Macro Corruption

Many macro viruses copy themselves to new files using macro copy commands. A macro virus can copy itself into a new document in this way, often attacking the global template called NORMAL.DOT first and then copying itself from the global template back to user documents.

A natural mutation often occurs in Microsoft Word environments[22]. The real reason for the corruption was never found, but it is believed to be connected to saving documents on floppy disks. Some users simply did not wait until the document was written perfectly to disk, which can result in a couple of bytes of corruption in the macro body. Because Word interprets the VBA code line by line, it will not generate an error message unless the faulty code is about to be executed[1].

As demonstrated earlier, macros are stored as binary data in Word documents. When the binary of the macro body gets corrupted, the virus code often can survive and work at least partially. The problem is that such corruptions are so common that often hundreds of minor variants of a single macro virus family are created by the "mutation engine" of Microsoft Word itself! For instance, the WM/Npad family has many members that are simply natural corruptions, which are not created intentionally.

Corrupted macro viruses can often work after corruption. There are several reasons for this commonly observed behavior:

- The VBA code necessary to copy a macro to another document is very short.
- Even a single working macro can copy dozens of corrupted macros.
- The corruption's side effects might only appear in conditional cases.
- The corruption happens after the replication of viral code.
- The virus supports an "On Error Resume Next" handler.

Consider the example shown in Listing 3.1.

Listing 3.1
A Corrupted Macro Example

```
Sub MAIN
    SourceMacro$= FileName$()+ "Foobar"
    DestinationMacro$ = "Global:Foobar"

    MacroCopy(SourceMacro$, DestinationMacro$)

    // Corruption here //
End Sub
```

Because most macro viruses include an error handler at the beginning of their code, macro virus compilation and execution has tended to be resilient to all but the most traumatic corruptions.

Because many AV products use checksums to detect and identify macro viruses, the antivirus software can get confused by the corrupted macro virus variants.

Using checksums is the only way to exactly identify each different variant.

Other types of viruses, such as Assembly-written viruses on DOS, most often fail immediately when the slightest corruption occurs in them. However, macro viruses often survive the corruption because the actual replicating instructions are so short in the macro body.

3.7.1.2 Macro Up-Conversion and Down-Conversion

When creating Word 97 and additional support for VBA, Microsoft decided to create new document formats and started to use a different, even richer macro language. To solve compatibility problems for customers, they decided to automatically convert old macros to the new formats. As a result, when a macro virus in the Word 95 WordBasic format was opened with the newer editions of Word, the virus might be converted to the new environment, creating a new virus. As a result, WM viruses are often converted to W97M format, and so on.

The macro up-conversion issue generated many problems for antivirus researchers that went beyond simple technicalities. Some researchers believed it was not ethical to up-convert all old macro viruses to the new format, while others believed it was the only choice to protect customers. Today, techniques are available[23] to convert the different macro formats to a canonical form; thus, detection can be done on the canonical form using a single definition. This greatly simplifies the macro detection problems and reduces the antivirus scanner's database growth because less data need to be stored to detect the viruses, and the virus code no longer needs to be replicated on more than one Office platform.

3.7.1.3 Language Dependency

Given that Microsoft translated basic macro commands, such as FileOpen, into different language versions for Office products, most viruses that use these commands to infect files cannot spread to another language version of Microsoft Office, such as the German edition.

Table 3.1 lists some of the most common macro names in Microsoft Word in various localized versions.

Table 3.1

Common Macro Names in Microsoft Word in Some Localized Versions

English	Finnish	German
FileNew	TiedostoUusi	DateiNeu
FileOpen	TiedostoAvaa	DateiOffnen
FileClose	TiedostoSulje	DateiSchliesen

continues

Table 3.1 continued

Common Macro Names in Microsoft Word in Some Localized Versions

English	Finnish	German
FileSave	TiedostoTallenna	DateiSpeichern
FileSaveAs	TiedostoTallennaNimmellä	DateiSpeichernUnter
FileTemplates	TiedostoMallit	DateiDokVorlagen
ToolsMacro	TyökalutMacro	ExtrasMakro
Spanish	**French**	**Italian**
ArchivoNuevo	FichierNouveau	FileNuovo
ArchivoAbrir	FichierOuvrir	FileApri
ArchivoCerrar	FichierFermer	FileChiudi
ArchivoGuardar	FichierEnregister	FileSalva
ArchivoGuardarComo	FichierEnregisterSous	FileSalvaConNome
ArchivoPlantillas	FichierModules	FileModelli
HerramMacro	OutilsMacro	StrumMacro

Various Office products use different versions of these macro names. A few common examples can be found in Table 3.2 for English Microsoft Office products.

Table 3.2

Differences in Macro Names Between Word and Excel

Microsoft Word	Microsoft Excel
AutoClose	Auto_Close
AutoOpen	Auto_Open

The WM/CAP.A[24] virus is an example of language independence, because it uses menu indexes. Using menu indexes was strongly recommended to macro developers by the Microsoft Access team. Of course, menu indexes only work reliably if the host environment has not been customized.

The WM/CAP.A virus also fools users to believe that they are saving their files in RTF (Rich Text Format) when, in fact, they are saving them as infected DOC files instead. Users would prefer to save files as RTF to avoid saving active macros into documents. The virus takes over the File/SaveAs... operation for this trick[25].

3.7.1.4 Platform Dependency of Macro Viruses

Although most macro viruses are not platform-dependent, several have introduced some sort of dependency on the actual platform. Microsoft Office products are used not only on Windows but also on Macintosh systems. Not all macro viruses, however, are able to work on both platforms because of the following common reasons.

Win32 function calls

A few macro viruses define API function calls for their own use from the Win32 set of Windows. Such viruses might fail to replicate on the Mac because the API is not implemented on it. For instance, the virus WM/Hot.A used the GetWindowsDirectory() API calls in January, 1996[26].

```
Declare Function GetWindowsDirectory Lib "KERNEL.EXE" \
        (Buffer As String, Size As Integer) As Integer
        :
        :
GetWindowsDirectory(WinPath$, SizeBuf)
```

Tricky macro viruses use Win32 callback functions to run code outside of the context of the macro interpreter. For instance, a simple string variable is defined that has encoded Assembly code. Often the chr() function is used to build larger strings that contain code. Then the callback routine is used to run the string directly as code. This way, the macro virus jumps out of the context of the macro interpreter and becomes CPU and platform -dependent.

For example, the {W32, W97M}/Heathen.12888 virus uses the CallBack12(), CallBack24(), and CreateThread() APIs of KERNEL32.DLL to achieve infection and dropping mechanism of both documents and 32-bit executables.

Location of files in storage

Another key difference among operating system platforms is the location of files on the disk. Some macro viruses use hard-coded path names, such as the location of the NORMAL.DOT template on the C: drive. Obviously, they cannot work on the Mac.

In addition, viruses often assume a Windows-style file system, even if they use the "correct" VBA methods to get the configured folder locations.

Registry modifications

Some macro viruses modify Registry keys on Windows systems to introduce extra tricks or store variables. Such viruses introduce OS dependencies as a result.

3.7.1.5 Macro Evolution and Devolution

Macro viruses consist of a single macro or set of macros. Because these individual macros must be recognized by the antivirus programs on a macro-to-macro basis, a set of interesting problems occurs.

Some macro viruses will copy more than their own set of macros. They can snatch macros from the documents they had infected previously. This way, the virus might evolve into new forms naturally. Some viruses will lose macros from their sets and thus will naturally devolve[27] to other forms. There are also sandwiches[28], which are created when more than one macro or script virus shares a macro name or script file.

A set of dangerous situations was introduced because of antivirus detection and disinfection, one of which was found by Richard Ford[29]. The problem occurs when an antivirus product detects a subset of known macros ("macro virus remnants") from a set of macros in a newer virus that has at least one new macro among the other, older known macros. If the antivirus product removes the known macros, it could create a new virus by leaving a macro or set of macros in the document that is still part of the virus and often remains viral itself.
This problem can be avoided in several different ways, one of which is to remove all macros from infected documents (although this means removing user macros from the documents also). Researchers also suggested defining a minimal set of macros from a known virus to "safely remove" a set of viral macros from a document. However, there is a natural extension of Richard Ford's problem, which was found by Igor Muttik, described in a scientific paper of Vesselin Bontchev in detail[30]. This is known as "Igor's problem."

Suppose there is a virus known as Foobar that consists of a single macro called *M*. The antivirus program identifies M in an infected document, but when it attempts to disinfect the document, a problem occurs. This happens because there is a variant of the Foobar virus in the document. This variant of Foobar consists of {M, P} macros. Unfortunately, macro P is not known to the antivirus program; thus, whenever the antivirus removes macro M, it will leave P behind. The major problem is that P could be a fully functional virus on its own. Consequently, an antivirus program, even with exact identification for Foobar, would create a new virus by accident when repairing a document in such situation. Indeed, sometimes, it is dangerous to remove a macro virus without removing all macros from the document.

The environment of the malicious programs and agents within the environment of the programs can make changes in computer viruses that result in newly evolved or devolved creatures. In addition, multiple infections of different macro viruses in the same document can lead to "crossed" threats and behavior. Indeed, viruses can become "sexual" by accident: They can exchange their macros ("genes") and evolve and devolve accordingly.

3.7.1.6 Life Finds a Way—Source, P-code, and Execode

Microsoft file formats had to be reverse-engineered by AV companies to be able to detect computer viruses in them. Although Microsoft offered information to AV developers about certain file formats under NDA, the information received often contained major bugs or was incomplete[31].

Some AV companies were more successful in their reverse-engineering efforts than others. As a result, a new kind of expert quickly emerged at AV companies: the file format expert. Among the best file format experts are Vesselin Bontchev, Darren Chi, Peter Ferrie, Andrew Krukov ("Crackov"), Igor Muttik, and Costin Raiu to just name a few.

Starting with VBA5 (Office 97), documents contain the compressed source of the macros, as well as their precompiled code, called p-code (pseudocode), and execode. Execode is a further optimization of p-code that simply runs without any further checks because its state is self-contained. A problem appears because under the right circumstances, any of these three forms can run.

Unfortunately, some AV companies produced products that occasionally corrupted the documents they repaired. In other cases, the products removed any of the three forms, without removing at least one of the other two. For example, some antivirus programs might remove the p-code, but they leave the source behind. Normally the p-code would run first. The VBA Editor also displays decompiled p-code as "a source" for macros, instead of using the actual source code of macros which are saved in the documents. Given the right circumstances, however, when the p-code is removed but the source is not, the virus might be revived. This happens when the document is created in Office 97 but is opened with Office 2000.

Most viruses break without the source because they often use a function such as MacroCopy() that copies the source. In other cases such as worms, however, the macro will continue to function properly because it does not refer to its source.

In some other cases, the execode might run on its own without source and p-code in the document. If the VBA project does contain execode and is opened by the same version of the Office application as the one that created it, the execode runs, and everything else is ignored. In fact, antivirus researchers experienced a case with the X97M/Jini.A virus where both the p-code and the source were removed from a document, but the execode was left behind when an antivirus program "cleaned" the document. The virus runs from the execode when the infected document is opened in the same version of Office that created it[26]; thus, some of the "half-cooked repaired" viruses can still function and infect further. Life finds a way, so to speak! Indeed, not all viruses will survive, but those that do don't need to refer to their sources or modules. Jini survives because it does not copy any modules. Instead, it copies the victim's data sheets to the workbook where it resides and then overwrites the file of the victim with the file in which it resides. Of course these tricky cases introduce major problems for the antivirus programs. Viruses that exist only in execode form are especially hard to detect. (So far, Microsoft has not provided information about this format to AV developers even under NDA[26].)

3.7.1.7 Macro Viruses in the Form of the Multipartite Infection Strategy

There are a couple of binary viruses that attempt to infect documents. These viruses are not primarily dependent on the interpreted environments.

For instance, the multipartite virus, W32/Coke, drops a specially infected global template with a little loader code. This loader will fetch polymorphic macro code (as discussed in Chapter 7) from a text file into the global template. As a result, Coke is one of the most polymorphic binary viruses, as well as a macro virus. Polymorphic macro viruses are usually very slow because of many iterations required to run their code. However, it is normally the polymorphic engine that is slow. Because Coke generates polymorphic macro virus code in a text file using its Win32 code, the polymorphic macro of Coke is not as slow as most polymorphic macro viruses based on macro polymorphic engines.

Other viruses do not need Word to infect Office documents. These viruses are very rare and usually very buggy. Even the Word 6 file format is complicated enough to parse and modify it in such a way that a macro is inserted in the file. The W95/Navrhar virus injects macro code to load a binary file from the end of the Word document. Thus, Navrhar can infect documents without Word installed on the system.

3.7.1.8 New Formula

Another set of problems occurred because Excel not only supported standard macros, but formula macros as well. As you might expect, formulas are not stored with macros; therefore, their locations had to be identified.

Viruses that need the Microsoft Excel Formula language to replicate are predicated with the XF/ tag. Excel macros are stored in the Excel macro module area, but Excel formulas are stored in the Excel 4 macro area instead. Therefore, these viruses are not visible via the Tools/Macro menu, and users must create a special macro to find them. The first such virus, known as XF/Paix[32], was of French origin.

3.7.1.9 Infection of User Macros

Most macro viruses replicate their own set of macros to other documents. However, infection is also possible by modifying existing user macros to spread the virus code, similar to the techniques of binary infectors. In practice, very few macro viruses use these parasitic techniques. This is because most of the documents do not contain user macros, and thus the spreadability of such parasitic macro viruses is seriously limited. (In addition, macro viruses often delete any existing macros in the objects they are infecting.) This kind of macro virus is very difficult to detect and remove with precision.

3.7.1.10 New File Formats: XML (Extensible Markup Language)

Microsoft Office 2003 introduced the ability to save documents in XML, textual format. This caused a major headache for antivirus developers, who must parse the entire file to find the embedded, encoded OLE2 files within such documents and then locate the possible macros within them. Currently, Word and Visio 2003 support the XML format with embedded macros[33]. Initially, such documents did not have any fields in their headers that would indicate whether or not macros were stored in them. Microsoft changed the file format of Word slightly in the release of this version due to pressure from the AV community.

Visio 2003, however, was released without any such flags, leaving no choice for AV software but to parse the entire XML file to figure out whether there are macros in it. Thus, the overhead of scanning increases dramatically and is particularly noticeable when files are scanned over the network.

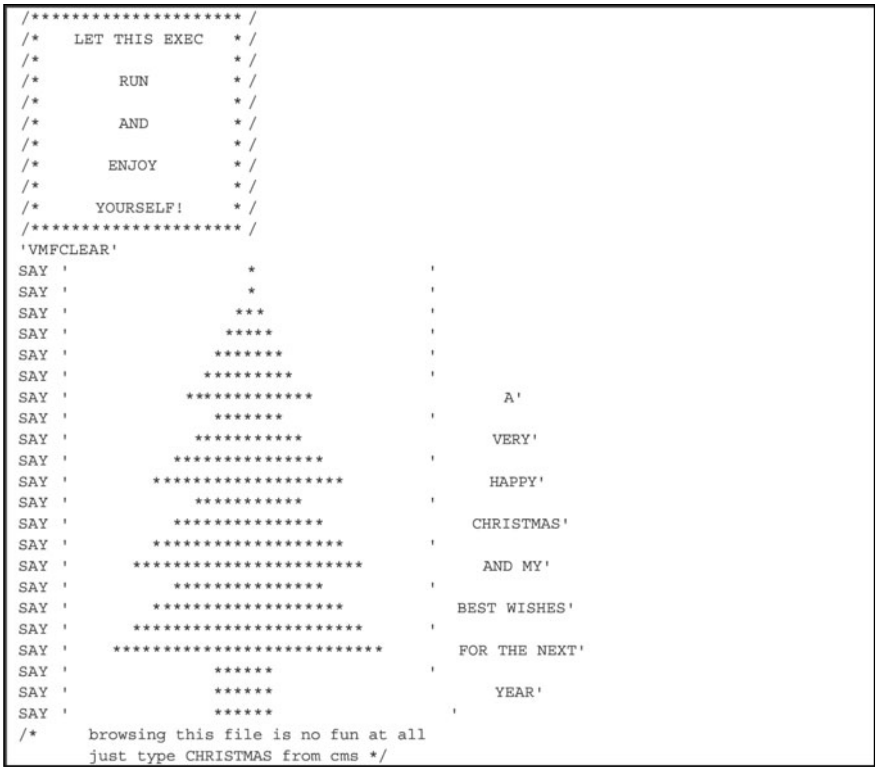
Note

XML infection was considered by virus writers years ago using VBS (Visual Basic Script) code. The idea was that an XML file can contain a Web link to reference code that is stored in an XSL (Extensible Stylesheet Language) file. This technique was first proposed by the virus writer, Rajaat, and was later introduced by the W32/Press virus of Benny.

3.7.2 REXX Viruses on IBM Systems

IBM has a long tradition of implementing interpreted language environments. Examples include the powerful Job Control languages on mainframe systems. IBM also introduced the REXX command script language to better support both large batch-like installations and simple menu-based installation programs. Not surprisingly, virus writers used REXX to create new script viruses. In fact, some of the first mass-mailer script viruses, such as the infamous CHRISTMA EXEC[34] worm, were written in REXX. The worm could execute on machines that supported the REXX interpreter on an IBM VM/CMS system and were also connected to a network. This worm was created by a German Informatics student[35] in 1987.

CHRISTMA EXEC displayed the Christmas tree and message shown in Figure 3.7 when the REXX script was executed by the user. Obviously, such viruses rely on social engineering for their execution on remote systems. However, users were happy to follow the instructions in the source of the script. The worm looked around for user IDs on the system and used the CMS command SENDFILE (or SF in short form) to send CHRISTMA EXEC files to other users.



```
/********************* /
/*   LET THIS EXEC   * /
/*                   * /
/*        RUN        * /
/*                   * /
/*        AND        * /
/*                   * /
/*       ENJOY       * /
/*                   * /
/*     YOURSELF!     * /
/********************* /
'VMFCLEAR'
SAY '                  *                 '
SAY '                  *                 '
SAY '                 ***                '
SAY '                *****               '
SAY '               *******              '
SAY '              *********             '
SAY '            *************                   A'
SAY '               *******              '
SAY '             ***********                   VERY'
SAY '           ***************          '
SAY '         *******************              HAPPY'
SAY '             ***********            '
SAY '           ***************               CHRISTMAS'
SAY '         *******************        '
SAY '       ***********************            AND MY'
SAY '           ***************          '
SAY '         *******************           BEST WISHES'
SAY '       ***********************      '
SAY '     ***************************       FOR THE NEXT'
SAY '               ******               '
SAY '               ******                      YEAR'
SAY '               ******                  '
/*     browsing this file is no fun at all
       just type CHRISTMAS from cms */
```

Figure 3.7 A snippet of the CHRISTMA EXEC worm.

At one point, such viruses were so common that IBM had to introduce a simple form of content filtering on its gateways to remove them.

REXX interpreters were made available on other IBM operating systems, such as OS/2, as well; thus, a few REXX viruses appeared on OS/2.

3.7.3 DCL (DEC Command Language) Viruses on DEC/VMS

The Father Christmas worm was released in 1988. This worm attacked VAX/VMS systems on SPAN and HEPNET. It utilized DECNET protocols instead of Internet TCP/IP protocols and exploited TASK0, which allows outsiders to perform tasks on the system.

This worm made copies of itself as HI.COM. Although DOS COM files have a binary format, the DCL files with COM extensions are simple text files. The worm sent mail from the infected nodes; however, it did not use e-mail to propagate itself. In fact, this worm could not infect the Internet at all. It attacked remote machines using the default user account and password and copied itself line by line (151 lines) to the remote machine.

Then the worm exploited TASK0 to execute its own copy remotely. It used the SET PROCESS/NAME command to run itself as a MAIL_178DC process on the remote node[36]. Father Christmas mailed users on other nodes the following funny message:

```
$ MAILLINE0 = "HI,"
$ MAILLINE1 = ""
$ MAILLINE2 = " HOW ARE YA ? I HAD A HARD TIME PREPARING ALL THE PRESENTS."
$ MAILLINE3 = " IT ISN'T QUITE AN EASY JOB. I'M GETTING MORE AND MORE"
$ MAILLINE4 = " LETTERS FROM THE CHILDREN EVERY YEAR AND IT'S NOT SO EASY"
$ MAILLINE5 = " TO GET THE TERRIBLE RAMBO-GUNS, TANKS AND SPACE SHIPS UP HERE AT"
$ MAILLINE6 = " THE NORTHPOLE. BUT NOW THE GOOD PART IS COMING."
$ MAILLINE7 = " DISTRIBUTING ALL THE PRESENTS WITH MY SLEIGH AND THE"
$ MAILLINE8 = " DEERS IS REAL FUN. WHEN I SLIDE DOWN THE CHIMNEYS"
$ MAILLINE9 = " I OFTEN FIND A LITTLE PRESENT OFFERED BY THE CHILDREN,"
$ MAILLINE10 = " OR EVEN A LITTLE BRANDY FROM THE FATHER. (YEAH!)"
$ MAILLINE11 = " ANYHOW THE CHIMNEYS ARE GETTING TIGHTER AND TIGHTER"
$ MAILLINE12 = " EVERY YEAR. I THINK I'LL HAVE TO PUT MY DIET ON AGAIN."
$ MAILLINE13 = " AND AFTER CHRISTMAS I'VE GOT MY BIG HOLIDAYS :-)."
$ MAILLINE14 = ""
$ MAILLINE15 = " NOW STOP COMPUTING AND HAVE A GOOD TIME AT HOME !!!!"
$ MAILLINE16 = ""
$ MAILLINE17 = "    MERRY CHRISTMAS"
$ MAILLINE18 = "       AND A HAPPY NEW YEAR"
$ MAILLINE19 = ""
$ MAILLINE20 = "             YOUR  FATHER CHRISTMAS"
```

3.7.4 Shell Scripts on UNIX (csh, ksh, and bash)

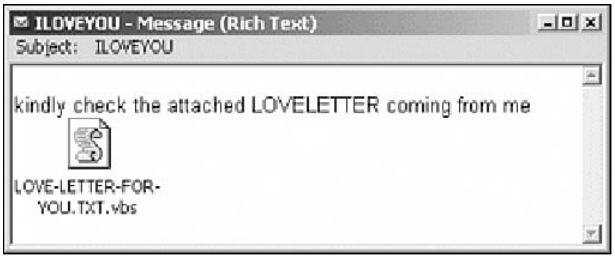
Most UNIX systems also support script languages, commonly called *shell scripts*. These are used for installation purposes and batch processing. Naturally, computer worms on UNIX platforms often use shell scripts to install themselves. Shell scripts have the advantage of being able to run equivalently on different flavors of UNIX. Although binary compatibility between most UNIX systems is not provided, shell scripts can be used by attackers to circumvent this problem. Shell scripts can use standard tools on the systems, such as GREP, that greatly enhances the functionality of the viruses.

Shell scripts can implement most of the known infection techniques, such as the overwriter, appender, and prepender techniques. In 2004 some new worms appeared such as SH/Renepo.A that use bash script to copy themselves into the StartupItems folders of mounted drives on MAC OS X. This indicates a renewed interest of worm developments on MAC OS X. In addition, threats like Renepo exposes MAC OS X systems to a flurry of attacks by turning the firewall off, run the popular password cracker tool John The Ripper, and create new user accounts for the attackers. However, current attacks require root privileges.

(It is expected that MAC OS X will be the target of future remote exploitation attacks as well.)

3.7.5 VBScript (Visual Basic Script) Viruses on Windows Systems

Windows script viruses appeared after the initial macro virus attack period was over. The VBS/LoveLetter.A@mm worm spread very rapidly around the world in May of 2000. LoveLetter arrived with a simple message with the subject ILOVEY-OU, as shown in Figure 3.8. The actual attachment has a "double extension." The "second" extension is VBS, which is necessary to run the attachment as a Visual Basic Script. This "second" extension is not visible unless the Windows Explorer Folder option Hide File Extensions for Known File Types is disabled. By default, this option is enabled. As a result, many novice users believed they were clicking a harmless text file, a "love letter."


ILOVEYOU - Message (Rich Text)
Subject: ILOVEYOU

kindly check the attached LOVELETTER coming from me

LOVE-LETTER-FOR-YOU.TXT.vbs

Figure 3.8 Receiving a "love letter."

On execution of the attachment, the VBS file runs with the script interpreter WSCRIPT.EXE. Mass-mailer VBS script worms typically use Outlook MAPI functions via CreateObject ("Outlook.Application") followed by the NameSpace ("MAPI") method to harvest e-mail addresses with AddressLists(), and then they mass-mail themselves as an attachment to recipients via the Send() method. In this way, many users receive e-mail from people they know. As a result, many recipients are curious enough to run the attachment—often on more than one occasion.

VBS viruses can use extended functionality via ActiveX objects. They have access to file system objects, other e-mail applications, and locally installed ActiveX objects.

3.7.6 BATCH Viruses

BATCH viruses were not particularly successful in the DOS years. Several unsuccessful attempts were made to develop in-the-wild BATCH viruses, none of which actually became wild. Nevertheless, common infection types, such as the prepender, appender, and overwriting techniques, were all developed as successful demonstrations. For example, BATCH files can be attacked with the appender technique by placing a goto label instruction at the front of the file and appending the extra lines of virus code to the end after the label.

BATCH viruses are also combined with binary attacks. BATVIR uses the technique of redirecting echo output to a DEBUG script; thus, the virus is a textual BATCH command starting with

```
rem [BATVIR] '94 (c) Stormbringer [P/S]
```

This is followed by a set of echo commands to create a batvir.94 file with the DEBUG script. The DEBUG command receives a G – GO command via the script and runs the binary virus without ever creating it in a new file.

BAT/Hexvir uses a similar technique, but it simply echoes binary code into a file and runs that as a DOS COM executable to locate and infect other files.

Some other tricky BATCH viruses use the FOR % IN () commands to look for files with the BAT extension and insert themselves into the new files in packed form using PKZIP. BAT/Zipbat uses PKUNZIP on execution of the infected BATCH files to extract a new file called *V.BAT*, which will infect other files by placing itself in them, again in zipped form. Members of the BAT/Batalia family use the compressor, ARJ, instead. Batalia, however, uses random passwords to pack itself into BATCH files.

Similar to BAT/Zipbat, the BAT/Polybat family also uses the PKZIP and PKUNZIP applications to pack and unpack itself at the ends of files. Polybat is practically a polymorphic virus. The virus inserts garbage patterns of percent signs (%) and ampersands (&) that are ignored during normal interpretation. For instance, the ECHO OFF command is represented in some way similar to the following:

```
@ec%&%h%&%o o%&%f%&%f
```

```
@e%&%ch%o&% %&o%f%f&%
```

BATCH viruses, or at least multi-component viruses with a significant BATCH part, are becoming a bigger threat on Windows systems. For instance, the BAT/Mumu family got especially lucky in corporate environments by using a set of binary shareware tools (such as PSEXEC) in combination with the BAT file–driven virus code.

Several custom versions of BATCH languages do exist, such as the BTM files in 4DOS and 4NT products—just to name a few—which also have been used by malicious attackers.

3.7.7 Instant Messaging Viruses in mIRC, PIRCH scripts

Instant messaging software, such as mIRC, supports script files to define user actions and simplify communications with others. The script language allows the definition of commands whenever a new member joins a conference and is often stored in script.ini in the system's mIRC folder.

IRC worms attempt to create or overwrite this file with an INI file that sends copies of the worm to others on IRC. The command script supports the /dcc send command. This command can be used to send a file to a recipient on a connected channel.

3.7.8 SuperLogo Viruses

In April of 2001, a new LOGO worm was created and mass-mailed to some antivirus companies. It never became wild, though, and there is definitely more than one reason for that. Its author calls herself Gigabyte. Gigabyte has a background of creating other malware and has authored mIRC worms. As you will see, she tried to use her existing mIRC knowledge to create the Logic worm[37]. The actual worm is created in Super Logo, a reincarnation of the old Logo language for Windows platforms, which claims to be "the Windows platform for kids."

In 1984, I came across several Logo implementations for various 8-bit computers. Our 8-bit school computer, the HT 1080Z—a Z80-based TRS-80 clone built in Hungary—had a top screen resolution of 128x48 dots in black and white. Although we had not paid too much attention to the fact at the time, the built-in Basic of HT 1080Z was created by Microsoft in 1980.

The Logo language's primary purpose is to provide drawing with the "Turtle." The Turtle is the pen, and its head can be turned and instructed to draw. For instance, in Super Logo, the following commands are common: HIDETURTLE, FORWARD, PENUP, PENDOWN, WAIT, and so on.

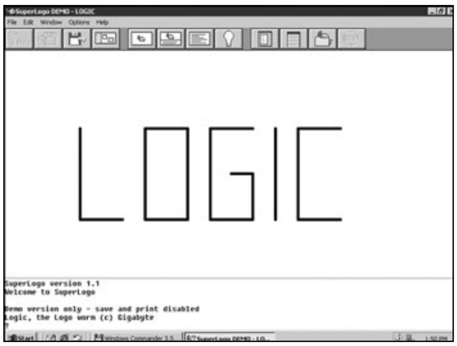
The set of commands can be formed as subroutines and saved in a Logo project file with an LGP extension. The actual project file is a pretokenized binary format, but commands and variable names remain easily readable and stored as Pascal-style strings. The project file can be loaded and executed with the Super Logo interpreter. The original Logo language is well extended in Super Logo to compete with other existing implementations. It can deal simultaneously with multiple graphical objects (see the cute Turtle as an example in Figure 3.9) and move them around the screen with complete mouse support.

Figure 3.9 Main Turtle ICON.

We can easily determine, however, that the Super Logo language does not support mailing or embedded executables; neither does it support spawning of other executables or scripts—yet Super Logo does support a PRINTTO "XYZ" command. XYZ can be a complete path to a file. With that statement, a Logo program might modify any file, such as winstart.bat, overwriting its content with something like the following:

```
@cls
@echo You think Logo worms don't exist? Think again!
```

Get the idea? When the logic.lgp project is loaded and executed, the worm will draw *LOGIC* on screen with a short message, as shown in Figure 3.10.



```
SuperLogo version 1.1
Welcome to SuperLogo

Demo version only - save and print disabled
Logic, the Logo worm (c) Gigabyte
?
```

Figure 3.10 The payload of the Logic worm.

The worm will make sure that a STARTUP.VBS file is created in one of the Windows startup folders and, as such, will be executed automatically the next time Windows is booted. The worm also tries to modify the shortcuts (if any) of some common Windows applications, such as notepad.exe, to start the VBS file without a reboot.

This VBS file propagates the 4175-byte logic.lgp worm project file to the first 80 entries of the Outlook address book. This is a very standard VBS mail propagation that has a set of minor bugs. In 2004, Gigabyte was arrested by Belgian authorities. She is facing criminal prosecution; the penalty might include imprisonment and large fines.

3.7.9 JScript Viruses

One of the reasons to turn off JScript support in a Web browser such as Internet Explorer has to do with JScript viruses. JScript viruses typically use functions via ActiveX communication objects. They can access such objects in a way similar to VBS scripts. For instance, the very first overwriting JScript viruses accessed the file system object via the CreateObject ("Scripting.FileSystemObject") method. This kind of virus was first created by jacky of the Metaphase virus-writing group around 1999.

The File System Object provides great flexibility to attackers. For example, an attacker can use the CopyFile() method to overwrite files. This is how overwriting JScript viruses work. Of course, more advanced attacks have been implemented by the attackers using the OpenTextFile(), Read(), Write(), ReadAll(), and Close() functions. Thus JScript viruses can carry out complex file infection functionality similar to VBS viruses, using a slightly different syntax.

3.7.10 Perl Viruses

Perl is an extremely popular script language. Perl interpreters are commonly installed on various operating systems, including Win32 systems. The virus writer, SnakeByte, wrote many Perl viruses in this script language.

Perl scripts can be very short, but they have a lot of functionality in a very compact form. Attackers can use Perl to develop not only encrypted and metamorphic viruses, but also entry point obscuring ones. The open(), print, and close() functions are used to move newly created content to a target file located in storage with the foreach() function.

For example, the following Perl sequence reads its source to the CurrentContent variable:

```
open(File,$0);
@CurrentContent=<File>;
close(File);
```

Perl viruses are especially easy to write because Perl is such a powerful script language to process file content.

3.7.11 WebTV Worms in JellyScript Embedded in HTML Mail

Microsoft WebTV is a special embedded device that allows users to browse the Web over their televisions. In July 2002, a new, malicious WebTV worm appeared, which at first glance was believed to be a Trojan horse. The payload of the worm reconfigured the access number (dial-up number) for the WebTV network to call 911 (the phone emergency center of the U.S.) instead, to perform a DoS attack.

WebTV HTML (Hypertext Markup Language) files can run HREF (hyperlink reference) within the <script> </noscript> tags using WebTV's Internet Explorer. The HREF would normally link a page to another location on the World Wide Web; however, in WebTV JellyScript, these special commands were used to set up the WebTV. Obviously, these commands have not been documented officially, though many people tried to figure out something more about WebTV and published detailed information about the available commands.

This malicious program, NEAT, was later identified as a worm that used the sendpage commands to send HTML mail that contained the worm to others on the WebTV network. The mail was sent by various fake "from" addresses, such as Owner_, minimoo, masonman, and so on.

The worm also introduced many pop-up advertising messages on the recipient's machine before it used the ConfirmPhoneSetup?AccessNumber command to reconfigure the dial-up number to 911 to overload the emergency network with a DoS attack.

3.7.12 Python Viruses

Python is an extremely handy programming language. Unlike shell script, which can be rather limited in functionality because of speed issues, Python is fast and modular. Because of its more general data types, Python can solve a larger problem. It has built-in modules to support I/O, system calls, sockets, and even interfaces to graphical user interface toolkits.

Although Python viruses are not extremely common, a few concept viruses written in Python scripts exist. They typically combine the open(), close(), read(), and write() functions to locate files with listdir() to replicate themselves to other files. However, this virus type is probably the simplest imaginable form for a Python virus, which could utilize much more on the system to implement a variety of infection strategies.

3.7.13 VIM Viruses

A successor of the VI UNIX editor is VIM (VI IMproved). Unlike VI, VIM works on Windows, Macintosh, Amiga, OS/2, VMS, QNX, and other systems. VIM is a text editor that includes almost all VI commands and a lot of new ones.

Among its many new features, VIM supports a very powerful scripting language that has already been used by virus writers to create worms. (The known example of such a worm is an intended worm, which will not replicate.)

3.7.14 EMACS Viruses

Just like VIM, newer versions of the EMACS editor also support scripting. This kind of virus is not common, but proof-of-concept creations exist for the environment.

3.7.15 TCL Viruses

TCL (Tool Command Language) is a portable script language that can run on systems such as HP-UX, Linux, Solaris, MAC, and even Windows. TCL is very similar language to Perl. TCL scripts are executed by the tclsh interpreter.

The first virus implemented in TCL (pronounced "tickle") was Darkness, a very simple virus written by Gigabyte in 2003. TCL supports foreach(), open(), close(), gets(), and puts() functions, which are all TCL script viruses need to replicate themselves.

3.7.16 PHP Viruses

PHP (a recursive acronym for PHP: hypertext preprocessor) is an open-source, general-purpose scripting language. It is well suited to Web development and can be embedded into HTML. PHP is different from client-side scripting, such as JScript, because PHP runs on the server instead of on the local machine. However, PHP also can be used in command-line mode without any server or browser.

PHP/Caracula was introduced in 2001 by the virus writer, Xmorfic, of the BCVG virus-writing group. The virus spreads as an overwriter and creates mIRC scripts to spread as a worm.

PHP viruses typically use the fopen(), fread(), fputs(), fclose() sequence to write themselves to new files, which they locate with direct action infection techniques using the opendir(), readdir(), closedir() sequence in combination with the file_exists() function.

There are examples of polymorphic PHP viruses, such as PHP/Feast, written by the virus writer, Kefi, in 2003. Feast looks for files to overwrite, but it overwrites them with an evolved copy of itself. In particular, each variable in the body of the virus will mutate to random character sequences.

3.7.17 MapInfo Viruses

MapInfo, developed by Geo-Information Systems, is not a widely used application. It is used for mapping and geographical analysis. The MPB/Kynel[38] virus demonstrated that it is possible to make this platform virulent. Kynel was created by Russian virus writers in late 2003.

MapInfo Professional has it own development environment called *MapBasic*, which is a Basic-like language. MapBasic is very powerful and, as expected, supports Open, Close, Read, and Write to both ASCII and binary files. It also supports API calls from other DLLs, dynamic data exchange (DDE), and object linking and embedding (OLE). When these programs are compiled, a new executable, MBX, is created, called *MapBasic eXecutable*. As expected, however, these files can be only executed by MapInfo.

The MPB/Kynel virus infects new tables. It enumerates for new tables each time the function WinChangedHandler() is called. WinChangedHandler() is triggered whenever the user changes something in a document. The virus hooks this function and uses this moment to create a copy of itself in the newly enumerated tables, as tablename.mif. It then inserts a Run Application line to this MBX executable into the TAB file of MapInfo documents. In this way, the MBX file will be run whenever the infected document is opened.

MapInfo is available on both Windows and Macintosh platforms. It is not very common, but like the SuperLogo virus threats, it demonstrates virus writers' interest in all platforms as possible targets.

3.7.18 ABAP Viruses on SAP

The first virus known to attempt to infect SAP was ABAP/Rivpas, written in April 2002. It is a proof-of-concept virus that is based on the Advanced Business Application Programming scripting language. This creation had a few intentional bugs and did not have a chance to replicate. However, other variants with the fix appeared quickly—that were real viruses. In about 20 lines of script, the virus replicates in databases by copying itself from one database to another.

3.7.19 Help File Viruses on Windows—When You Press F1…

A very powerful but surprisingly unpopular virus infection target is Windows Help files. Windows Help files are in binary format and contain a script section. The scripts have access to Windows API calls. Most Help viruses inject a little script into the SYSTEM directory of HLP files. This script section will be executed the next time the Help file is loaded. As a result, such a virus is triggered simply by pressing the F1 button in an application that is associated with an infected HLP file.

The major trick of such viruses is to define functions for their use, such as EnumWindows() of the USER32.DLL. For example, the Dream virus uses this technique to infect Windows Help files.

The RR ('USER32.DLL','EnumWindows','SU') script line will define an EnumWindows() callback for use. Then an EnumWindows(virusbody) call is made by the script, which will execute the "string," the virus body, via the callback. Thus execution can continue in native code, getting out of its script context.

The first virus to infect Windows Help files was the 32-bit polymorphic virus, W95/SK[39], written in Russia. Unlike Demo, SK uses WinExec() functions to execute a set of command.com /c echo commands to print code into a binary for execution outside of the HLP file in the root directory. The first native Help infector, the HLP/Demo virus, also appeared to replicate from one Windows Help file to another.

3.7.20 JScript Threats in Adobe PDF

The PDF format is used by Adobe Acrobat products. In 2003, the {W32,PDF}/Yourde virus infected PDF files using an executable that is dropped by a JScript exploit (a PDF form is also dropped). The binary is executed by the form when the form is loaded. The complete version of the Adobe Acrobat installation is required to infect files because the virus relies on the user's saving the infected file. (Saving the infected file cannot be forced externally with Adobe Acrobat. Additionally, the reader-only version cannot save PDFs at all.)

The JScript runs automatically by Acrobat itself, without relying on an external interpreter such as Windows Scripting Host; thus, the vulnerability is Acrobat version-specific.

3.7.21 AppleScript Dependency

AppleScript is used on Macintosh systems to support local scripting. Not surprisingly, some threats can replicate only if AppleScript is installed. For example, the AplS/Simpsons@mm worm is written in AppleScript. After it is executed, it utilizes Outlook Express or Entourage to send a copy of itself to everybody in the address book.

This particular worm was not reported frequently from the wild; however, AppleScript threats expose Mac users to similar security problems as those of other powerful script languages, such as VBS on Windows.

3.7.22 ANSI Dependency

IBM PCs introduced ANSI.SYS drivers that fulfill the needs of many users by providing the ability to reconfigure certain key functions via escape (ESC) sequences. These sequences are usually stored in a file with an ANS extension. ESC sequences can start with a special escape code (accessible via holding the Alt key and typing[40] on the numeric keypad).

Whenever the line DEVICE=ANSI.SYS is included in the CONFIG.SYS file, the support to execute ESC sequences is available. For example, a simple ANSI sequence can redefine the *N* key to *Y* and the *n* key to *y*. Consequently, the user would give the wrong answer to confirmation questions asked by applications. This would be done the following way:

```
ESC [78;89;13p ESC [110;121;13p
```

This kind of redefinition might be desirable for other keys; the Enter key also can be redefined, and del *.* or format c: might be displayed when Enter is pressed.

ANSI sequences also can be used to redefine entire commands. Thus, the wrong command name is displayed when a different command is typed.

3.7.23 Macromedia Flash ActionScript Threats

A newcomer on the malicious scene is ActionScript malware. The LFM virus uses the ActionScript of Flash files to create and run a DOS COM executable. Such threats, then, are fairly limited because they introduce several other dependencies.

For instance, LFM[41] needs to be downloaded to the local machine from a Web page. It can only infect files if it is downloaded to a folder that contains other clean files and only as long as the external file V.COM can run properly.

3.7.24 HyperTalk Script Threats

"An excellent beginning tool to teach average people, from 5th grade, on how to control their computers as masters rather than slaves."

—Steve Wozniak

HyperCard is a versatile environment that supports a scripting language called *HyperTalk*. Created by Bill Atkinson, HyperTalk is one of the most linguistic script languages available. Not surprisingly, some of the oldest computer viruses were written in HyperTalk. The first HyperTalk script virus was Dukakis, written around 1988.

HyperTalk scripts activate based on event handlers associated with a name in the stack. The scripts are stored in HyperCard data files, called *stacks*, which are in binary format. But the script code itself is purely textual inside the stacks.

For instance, upon opening a HyperCard stack, the openStack event handler can be invoked. This is fairly similar to how Microsoft Office products work with macros, though HyperCard is much more than a scripted text editor. It can be used to create many different projects with menus and database front-ends for cards (records in the database), and different stacks can share their functions with each other. HyperCard extended the promise of easy-to-use systems to easy-to-program environments.

HyperTalk script code is interpreted between the event handler tags of the keywords *on* and *end*. Here is an example:

```
on openStack
   ask "What is your name?"
   put it * it into field "Name"
end openStack
```

HyperCard was developed well before Microsoft's Visual Basic. Like Microsoft Office products' global templates (or should I say, the other way around?), HyperCard supports a so-called Home stack, which contains an arsenal of useful scripts. Most HyperCard viruses infect the Home stack by copying themselves into it with the help of *put* keywords. After this, they can copy themselves to the newly opened stacks. Any stack can be a Home stack, as long as its name is *home*.

The Dukakis virus uses the following lines to select its script body for a new copy:

```
put the script of stack "home" into temp2
get offset (""-** The HyperAvenger **-,"temp2)
put char it to it+2426 of temp2 into theCode
```

This script snippet looks for the offset where the virus code starts in the home stack and copies the virus script (2426 bytes) from that location to the variable, theCode. The virus then only needs to copy theCode into another stack later. The *this stack* is a reference to the currently opened stack. Its content can be accessed with yet another put command.

Several other HyperCard viruses exist on the Mac; the most famous ones are the Merry Xmas and 3 Tunes families.

3.7.25 AutoLisp Script Viruses

HyperTalk script viruses are very readable and easy to understand; AutoLisp threats are a little more difficult to read. A few script viruses, such as Pobresito[42] and ALS/Burstead[43], use the AutoLisp scripting feature of AutoCAD environments.

Note

Newer versions of AutoCAD also support VBA.

Pobresito was written during the summer of 2001. Burstead appeared much later, during December 2003 in Finland, and managed to infect a few major corporations that run modern versions of AutoCAD. AutoCAD is rather expensive software, and it is not used as widely as other script language environments.

AutoLisp scripts are stored in text files with the LSP extension. Burstead.A looks for the location of the base.dcl file in the AutoCAD search path, using the findfile function:

```
 (setq
path
(findfile
"base.dcl"))
```

This is done to locate the directory where the other LISP files can be found. Such viruses attempt to modify files with a load command to load their own LSP file. Thus, whenever the modified LSP file is executed, the virus can get control via the load command:

```
(load
 "foobar")
```

Here, foobar is the name of a file that has an LSP extension in the default folder.

Obviously, AutoLisp allows write-line functions, which could be used by attackers for different kinds of infection methods.

3.7.26 Registry Dependency

Some viruses are implemented to infect from Windows Registry files. The Registry is a central storage database on Windows systems. Previous versions of Windows mostly used INI files to store application settings. On modern Windows systems, the Registry database, called a *hive,* is used to store such information in trees.

An interesting capability of the Registry is that it stores file paths for system startup time execution under several different subentries of the hive, such as HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RUN.

Keys like this are commonly attacked by all kinds of malicious code, and other locations of the Registry provide similar attack points for virus writers. For instance, the W32/PrettyPark worm family modifies the Registry key located at HKEY_CLASSES_ROOT\exefile\shell\open\command to get executed whenever an EXE file is run by the user. The worm executes the program that the user wanted to run—but only after itself.

Registry-dependent viruses use such keys to insert a reference to system commands for later execution. Registry entry installation files are stored in textual format, and they contain information about keys and values to install via Regedit. Such viruses are implemented as a single command entry in the REG files. Regedit will interpret the commands in the REG file; as a result, a new entry will be stored in the Registry for later execution.

The malicious entry uses standard system commands with the passed parameters to look for other REG files on local and network sites and modifies them to include the command string to REG files. This technique is based on the fact that DOS batch commands can be executed from the Registry.

3.7.27 PIF and LNK Dependency

Viruses also attack PIFs (program information files) and LNKs (link files) on Windows systems. PIFs are created when you create a shortcut to or modify the properties of an MS-DOS program and allow you to set default properties, such as font size, screen colors, and memory allocation. PIFs also store the path of the executable to run.

Some viruses attack PIFs by modifying their internal links that point to an executable. The approach of typical PIF creations is to run commands via command.com execution, using this link path. They use the copy command to copy the PIF to other locations on the local disk, such as Windows, mIRC, or P2P folders, or to attack network resources.

The LNK (link shortcut files) on Windows 95 and above can be attacked in a manner similar to PIFs.

3.7.28 Lotus Word Pro Macro Viruses

Another class of macro viruses attacks Lotus Word Pro documents of Lotus SmartSuite. For example, the LWP/Spenty virus only replicates in the Chinese version of Word Pro. The virus infects files as they are opened by hooking the DocumentOpened() and DocumentedCreated() macros. The security settings of the document are changed in such a way that a password is set to 720401. In this way, the virus attempts to prevent any modifications of infected objects.

The Spenty virus became widespread in China in 2002. Spenty introduced the problem of Word Pro file parsing for antivirus producers. Word Pro uses a script-like macro language.

3.7.29 AmiPro Document Viruses

Viruses do not frequently attack AmiPro documents, and there is a good reason for this. Unlike most text editors, AmiPro saves documents and macros into two separate files. The documents are stored in files with SAM extensions, and the macro files are kept in files with SMM extensions. AmiPro viruses must connect the two files in such a way that when the SAM file is opened, it invokes execution of the SMM.

The APM/Greenstripe virus consists of four functions: Green_Stripe_Virus(), Infect_File(), SaveFile(), and SaveAsFile(). The SaveFile() and SaveAsFile() functions are hooks installed with the ChangeMenuAction() function, and they correspond to the Save and Save As menus. The virus uses the AssignMacroToFile() function to establish the connection between SAM and SMM files. The virus uses the FindFirst() and FindNext() functions to search for new SAM files to attack.

AmiPro viruses are much less likely to spread via e-mail than Microsoft Office macro viruses because of AmiPro's use of separate document and macro files, as opposed to a single container.

3.7.30 Corel Script Viruses

Corel Draw products also support a script language that is saved in files with CSC extensions. (In addition, contemporary versions of Corel Draw also support VBA.) Corel Script viruses typically look for victim files with the FindFirstFolder() function. The CSC/CSV virus identifies infected victim files by checking for the "REM ViRUS" marker in the CSC files.

If CSV does not find the marker in the file, it will attempt to infect it by prepending its script with print # commands. It then looks for the next file with the FindNextFolder() function. In practice, the virus creates a new host script with the same name, copies itself into it, and then appends the original host script to itself.

```
REM ViRUS GaLaDRieL FOR COREL SCRIPT bY zAxOn/DDT
```

The CSC/PVT virus follows a similar strategy. It uses the same functions to look for new files to infect. It even checks potential victims for REM PVT anywhere in the script before attempting to infect them.

```
REM PVT by Duke/SMF
```

Unlike CSV, the PVT virus appends itself to the end of the script. As a result, the original script runs first, and upon the exit of the original script, the appended script code is executed.

3.7.31 Lotus 1-2-3 Macro Dependency

Although there are widespread rumors about a Lotus 1-2-3 macro virus with the name Ramble, the actual threat is not viral. Rather, the known threat is a dropper of a BATCH virus. (This is not to say, however, that Lotus 1-2-3 macros would not be able to infect another set of Lotus 1-2-3 worksheets.)

The BAT/Ramble virus dropper, written by "Q The Misanthrope," works the following way: First, the user opens a Trojanized Lotus 1-2-3 document. The malicious Lotus macro activates upon when the document is opened. The malicious macro is then inserted in the A8167 ... A8191 range of the sheet. In this way, it is not visible to the user. After the macro runs, it creates a BATCH virus in the C:\WINSTART.BAT file.

After the BATCH virus is created by the dropper, the macro dropper code removes itself from the sheet, using the /RE command (Range Erase). It also removes the \0 macro name that automatically runs whenever a worksheet is opened.

It must be noted that newer versions of Lotus 1-2-3 have a different worksheet format, which has allowed a macro up-conversion problem to be introduced on this platform.

3.7.32 Windows Installation Script Dependency

The 32-bit Windows versions introduced a new installation script language in INF files. These scripts are invoked via the Windows Setup API. The install scripts have various sections for installation and uninstallation. The script can be generated manually or by using tools such as Microsoft's BATCH.EXE or INF generators.

One of the many features of installation scripts is the use of the autoexec.bat file. Commands can be directly installed into and removed from the automatically executed batch file on system startup. This is done via the UpdateAutoBat command in the Install section associated with a named section of the script. That section contains commands to delete lines—as well as to add new malicious commands—with CmdDelete and CmdAdd, respectively. (CmdDelete is used to delete the malicious code in case it was inserted into the file in a previous attack.)

The virus writer, 1nternal, introduced a couple of viruses, such as the INF/Vxer family, that take advantage of INF file infection via batch execution. The CmdAdd entries are used to deliver the source of the viral batch lines to AUTOEXEC.BAT. As a result, on each system startup the virus will look into the Windows\INF folder to infect other INF files.

3.7.33 AUTORUN.INF and Windows INI File Dependency

AUTORUN.INF files and Windows INI files are very similar in structure to Windows installation scripts. Some viruses modify the AUTORUN.INF file to get auto-launched whenever a removable disk is loaded.

AUTORUN.INF was a new feature in Microsoft Windows 95 systems. It was primarily designed to run an application automatically whenever a user inserted a CD into the CD-ROM drive. Whenever an AUTORUN.INF file exists in the root directory of a removable disk type, it is executed by most 32-bit Windows systems, although some of the newer editions of Windows primarily support the CD-ROMs only.

There are a couple of Registry entries associated with Autorun functionality. Whenever such options are enabled, the AUTORUN.INF is interpreted, and its Autorun section is invoked. The Autorun section supports an Open command that can be used to run an executable via the feature. This is the command that malicious code sets alter to be invoked automatically.

The HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer Registry entry must be modified with a NoDriveAutoRun or NoDriveTypeAutoRun entry set to customized values, such as 0xFF, to turn off the feature for each drive.

Windows INI files are attacked for a similar reason. For instance, the WIN.INI supports a Windows section. In that section, a run= entry can be used to RUN an application during the startup of Windows. Malicious Trojans often modify this entry to load themselves via system startup.

3.7.34 HTML (Hypertext Markup Language) Dependency

HTML does not support functionality to malicious attacks in its strict form, but it supports embedded scripting, such as VBScript or JScript. Several viruses attack HTML files. One of the most successful such attacks was implemented in the W32/Nimda worm in September 2001.

Nimda attacks HTML files by inserting a little JScript section into them. This section opens an EML file that contains a malformed MIME exploit. The JScript code uses the window.open function to launch the EML file. The result is an automatically executed worm executable upon accessing a compromised HTML page using a vulnerable Internet Explorer.

Some HTML threats get invoked from HTML files via HREF entries. They trick the user into clicking something that will, in turn, execute the referenced malicious code.

The first viruses that attacked HTML files were created by the virus writer, 1nternal. Although some vendors initially classified these threats as HTML viruses, the proper classification is based on the actual script language used, such as VBS.

3.8 Vulnerability Dependency

Fast spreading worms, such as W32/CodeRed, Linux/Slapper, W32/Blaster, or Solaris/Sadmind, can only infect a new host if the system can be exploited via a known vulnerability. If the system is not vulnerable or is already patched, such worms cannot infect them. However, several worms, such as W32/Welchia, exploits multiple vulnerabilities to invade new systems. Therefore, the system might remain exploitable by at least one of the nonpatched vulnerabilities.

Chapter 10 , "Exploits, Vulnerabilities, and Buffer Overflow Attacks," is dedicated to computer virus attacks that utilize exploits to spread themselves.

3.9 Date and Time Dependency

Tyrell: *What seems to be the problem?*

Roy: *Death!*

Tyrell: *Death. Well, I am afraid that's a little out of my jurisdiction.*

Roy: *I want more life...*
—Blade Runner, 1982

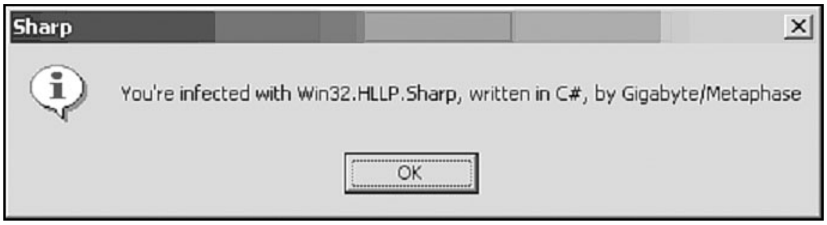
Several viruses replicate only within a certain time frame of the day. Others refuse to replicate before or after a certain date. For instance, the W32/Welchia worm only attempted to invade systems until January 2004.

Another example is the original W32/CodeRed worm, which was set to kill itself in 2001. However, other variants of the worm were modified to introduce an "endless life" version without this limitation. The life cycle manager of worms is discussed in more detail in Chapter 10.

3.10 JIT Dependency: Microsoft .NET Viruses

A natural evolution of Microsoft's ambitious computer language and execution environment developments is .NET Framework's Just-in-Time compilation. .NET uses executables that are somewhat special portable executable (PE) files. Currently, such executables contain a minimal architecture-dependent code (a single API call to an init function)[44]. Elsewhere, the compiled PE file contains MSIL (Microsoft Intermediate Language) and metadata information. The first viruses that targeted .NET executables were not JIT-dependent. For example, Donut[45] was created by Benny in February of 2002. This virus attacked .NET executables at their native entry point, replacing _CorExeMain() import (which currently runs the JIT initialization) with its own code and appending itself to the end of the file. A few months later, JIT-dependent viruses appeared that could infect other MSIL executables. The first such virus was written by Gigabyte.

W32/HLLP.Sharpei[40] implements a simple prepender infection technique. The MSIL code of the virus is JIT compiled by the CLR (common language runtime) of .NET Framework. JIT does not compile the module when it is loaded, but only when a particular method is first used. Only then is the MSIL code translated to the local architecture, and native code execution begins. Figure 3.11 shows the payload message of the W32/HLLP.Sharpei virus.



You're infected with Win32.HLLP.Sharp, written in C#, by Gigabyte/Metaphase

Figure 3.11 The payload message of Sharpei.

In 2004, new infection techniques appeared that targeted .NET executables. These new viruses parasitically infect MSIL programs. It is not surprising that such viruses did not show up any earlier because it is much more difficult to implement them. In fact, some researchers argued that such complex MSIL viruses will never appear. For example, the metamorphic virus, MSIL/Gastropod, uses the System.Reflection.Emit namespace to rebuild its code and the host program to alter the appearance of the virus body. Gastropod is a creation of the virus writer, Whale, who also authored the W95/Perenast viruses. (Whale was captured by the Russian police in November 2004. He was required to pay $50.)

On the other hand, the MSIL/Impanate virus is aware of both 32-bit and 64-bit MSIL files and infects them using EPO (Entry Point Obscuring) techniques without using any library code to do so. MSIL/Impanate was authored by the virus writer, roy g biv.

Note

More information on infection techniques is available in Chapter 4, "Classification of Infection Strategies." Metamorphic viruses are discussed in Chapter 7, "Advanced Code Evolution Techniques and Computer Virus Generator Kits."

3.11 Archive Format Dependency

Some viruses might not be able to spread without packed files. A few viruses only infect archive file formats. The majority of such viruses infect binary files, as well as ZIP, ARJ, RAR, and CAB files (to name the most common archive formats).

Spreading viruses in archive files gained popularity when Microsoft implemented a virus-protection feature for Outlook. Outlook no longer runs regular executable extensions, and recent versions simply do not provide such attachments to end users. However, virus writers quickly figured out that they could send packed files, such as zipped files, which Outlook does not remove from e-mail messages.

Some tricky mailer or mass-mailer worms, such as W32/Beagle@mm[46], even use password-protected attachments. Because the password and instructions on how to use it are available to the user, the malicious code can trick the user into running an application, such as Winzip, and typing the given password to unpack and then execute its content. Such viruses often carry their own packer engines, such as InfoZIP libraries, to create new packer containers.

File infector viruses typically insert a new file into an archive file. For example, ZIP infection is simple because ZIP stores a directory for each file in the container's archive. By locating such headers, viruses can insert new files into the project and trick the user into running the files. For example, viruses might insert a file with a name such as "readme.com" and simply hope that the user will execute it to read "the documentation" of the package.

Some very complex viruses, such as the Russian virus, Zhengxi[47], infect self-extracting EXE files with multiple archive infection capability, including the packed file format, HA, inside such binary files.

3.12 File Format Dependency Based on Extension

Some viruses have extension dependency. Depending on the extension, a file might be placed in a different execution environment. A simple example of this is COM and BAT (ASCII) extension replacement. As a COM file, the file can function as binary. With a BAT extension, it looks like an ASCII BATCH file.

Other common examples of this kind of dependency are as follows:

- COM/VBS
- COM/OLE2 (a trivial variant has the header of an OLE2 file)
- HTA/SCRIPT
- MHTML (Binary+Script)
- INF/COM
- PIF/mIRC/BATCH

This method is often used as an attempt to confuse scanners about the type of object they are scanning. Because scanners often use header and extension information to determine the environment of the file, their scanning capabilities (such as heuristics analysis) might be affected if they do not identify the type of object properly.

For example, PIF worms typically use mIRC, BAT, or even VBS combinations, based on extension dependency. A file with a PIF extension will function as a PIF. However, with a BAT extension, it will run as a BATCH instead, and the PIF section in the front of the file is simply ignored. Other examples include an mIRC and BATCH combination based on extension dependency tricks.

Figure 3.12 demonstrates how the PIF is organized for extension dependency. The Phager virus uses the previously discussed technique.

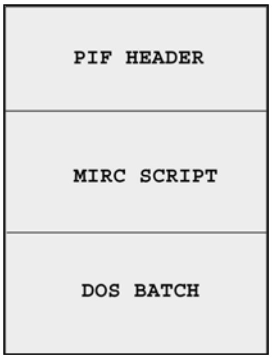


Figure 3.12 A high-level structure of a PIF with extension dependency.

Another example that involves extension dependency tricks is INF/Zox, which infects Windows INF files. The main virus body is stored in INF/Zox in an INF file called ULTRAS.INF. However, this INF file can run as a DOS COM executable when renamed.

In the INF form, the virus uses CmdAdd (add command) entries to attack AUTOEXEC.BAT. It also uses the CopyFile entry of the DefaultInstall section to copy the ULTRAS.INF file as Z0X.SYS. The trick is that the new AUTOEXEC.BAT section will rename the Z0X.SYS file to Z0X.COM and run it. The virus starts with a comment entry in the INF form using a semicolon (;) (0x3b).

When the file is loaded as a DOS COM file, the marker is ignored as a compare (CMP) instruction. After the comment, binary code is inserted that "translates" to a jump (JMP) instruction to the binary portion of the virus code at the end of the file:

```
13BE:0100 3B00         CMP     AX,[BX+SI]  ; Compare instruction ignored
13BE:0102 E9F001       JMP     02F5         ; Jump to binary virus start
```

Zox is a direct-action overwriter virus. It overwrites INF files with itself.

3.13 Network Protocol Dependency

Nowadays, the Internet is the largest target of virus attacks. TCP and UDP protocols are used by malicious mobile code[48] to attack new targets. There are some old worms, however, such as the Father Christmas worm, that could not spread on the Internet because they relied on DECNET protocols—thus, computer worms are typically network protocol–dependent.

3.14 Source Code Dependency

Some tricky computer viruses, such as those of the W32/Subit family, infect source files such as Visual Basic or Visual Basic .NET source files. Other viruses spread in C or Pascal sources. These threats have a very long history.

Consider the C source file shown in Listing 3.2, in clean and infected form.

Listing 3.2

A Source Infector Virus

```
#include <stdio.h>
void main(void)
{
    printf("Hello World!");
}
The infected copy would look similar to the following:
#include <stdio.h>
void infect(void)
{
  /* virus code to search for *.c files to infect */
}
void main(void)
{
   infect(); /* Do not remove this function!! */
   printf("Hello World!");
}
```

After the infected copy is compiled and executed, the virus will search for other C sources and infect them.

Source code viruses typically use a large string to carry their own source code, defined as a string. The W32/Subit family uses a concatenated string to define its source code, starting with the following lines:

```
J = "44696D20532041732053797374656D2E494F2E53747265616D5772697465720D"
J = J & "0A44696D204F2C205020417320446174650D0A44696D2052204173204D696372"
J = J & "6F736F66742E57696E33322E52656769737472794B65790D0A52203D204D6963"
```

This will be converted to Visual Basic .NET source code:

```
Dim S As System.IO.StreamWriter
Dim O, P As Date
Dim R As Microsoft.Win32.RegistryKey
:
:
```

The source code infectors replicate in two stages. The first stage is the running of an already infected application with the embedded virus code. After the New() function is called in the infected program, the virus code will search for other Visual Basic .NET project source files on the system and copy its own source code into those files. In the second stage, Subit inserts a function call to run the virus body itself. As a result, the virus can multiply again after the compromised source is compiled and executed on a system.

The major problem with such viruses is that they can appear virtually anywhere in the application, inserted somewhere in the code flow. The code of the virus will be translated differently, depending on the language and the compiler version and options, making the virus look different in binary form on various systems.

3.14.1 Source Code Trojans

The idea of source-only viruses originates in the famous "self-reproducing program" ideas of Ken Thomson (co-author of the UNIX operating system). In his article, "Reflections on Trusting Trust,"[49] Thomson introduced the idea of C programs, so-called "guines," that print an exact copy of their source as an output. The idea is nice and simple. The program source's code is defined as a string that is printed to the output with the printf() function.

Thomson also demonstrated a CC (C compiler) hack. The idea was to modify the source code of CC in such a way that whenever the modified compiler binary is used, it will do the following two things:

- Recognize when the source code of login was compiled and insert a Trojan function into the original source. The Trojanized version of login would let anybody log in to the system with his or her own password. Furthermore, it would let an attacker connect with a specific password for any user account.
- Introduce source modifications to the CC sources on the fly. Thus, the modification in the source code was available only during the compilation, and it was quickly removed after the compiler's source was compiled.

Source code infectors use the Thomson principle to inject themselves into application source files. Such viruses will be more relevant in the future as open source systems gain popularity.

3.15 Resource Dependency on Mac and Palm Platforms

Some computer viruses are dependent on system resources. For example, the Macintosh environment is a very rich platform of resources. Various functions are implemented in the form of resources that can be edited easily via Resource Editors. For instance, there is a menu definition resource on the Mac. Such definitions get invoked according to the applications' menu items. Macs store information in two forks for each file on the disk: the data fork and the resource fork. Resources, stored in the resource fork, contain code. Because even data files can contain resources on the Mac, the distinction between data and code files is not as clear-cut as it is for the PC, for example.

The MDEF (menu definition) viruses on the Apple Macintosh use the technique of replacing menu definitions with themselves. Thus, the virus code gets invoked whenever a particular menu is activated.

Table 3.3 contains common resource types on the Mac. It is an incomplete list of the most commonly attacked resources by malicious code on the Mac platform[36].

Table 3.3

Common Resource Types on the Mac

Resource Type	Description
ADBS	Apple Desktop Service
CDEF	Control Definition Function
DRVR	Device Driver
FMTR	Disk Format Code
CODE	Code Segment
INIT	Initialization Code Resource
WDEF	Windows Definition Function
FKEY	Command-Shift-Number Function
PTCH	ROM Patch Routine
MMAP	Mouse Function

Similar dependencies exists in the Palm viruses. The Palm stores executable applications in PRC files with special application resources. When the application is executed, the resources are accessed from it. In particular, the DATA and CODE resources are important for program execution. The virus Palm/Phage, discovered in September 2000, reads its own DATA and CODE resources and overwrites other applications resources with these. This resource dependency is very similar to the one on the Macintosh platforms.

3.16 Host Size Dependency

To infect applications accurately, many computer viruses have limits on how small or how large the applications they infect can be. For instance, COM files on DOS cannot load if they are larger than a code segment. Consequently, most DOS viruses introduce limits to avoid infecting files that would grow past acceptable limits if the virus code were included in them.

In other cases, viruses such as W95/Zmist use an upper size limitation, such as 400KB, for a file. This enhances the virus infection's reliability by reducing the risks involved in infecting files that are too large. Furthermore, host size dependency also can be used as an antigoat technique (see more details in Chapter 6, "Basic Self-Protection Strategies") to avoid test files that computer virus researchers use.

3.17 Debugger Dependency

Some viruses use an installed debugger, usually DEBUG.EXE of DOS, to convert themselves from textual to binary forms or simply to create binary files. Such threats typically use a piped debug script input to DEBUG, such as

```
DEBUG <debugs.txt
```

The input file contains DEBUG commands such as the following:

```
N example.com
E 100 c3
RCX
1
W
Q
```

This script would create a 1-byte long COM file containing a single RET instruction. A single RET instruction in a COM file is the shortest possible COM program. COM files are loaded to offset 0x100 of the program segment. Before the program segment, the PSP (program segment prefix) is located at offset 0; thus, a single RET instruction will give control to the top of the PSP, assuming that the stack is clear and a zero is popped. The trick is that the top of the PSP contains a 0xCD, 0x20 (INT 20 – Return to DOS interrupt) pattern:

```
13BA:0000 CD20          INT     20
```

So whenever the execution of a program lands at offset 0, the program will simply terminate.

Note

The N command is used to name an output file. The E command is used to enter data to a memory offset. The CX register holds the lower 16-bit word of the file size, and BX holds the upper 16-bit word. The W command is used to write the content to a file. Finally, the Q command quits the debugger. Viruses typically use several lines of data that use the Enter command to create the malicious code in memory.

The virus writer, Vecna, used this approach in the W95/Fabi family to create EXE files using Microsoft Word macros and debug scripts in combination. From the infected MS Office documents, Fabi creates a new file in the root directory as FABI.DRV and uses the PRINT commands to print the debug script into it:

```
OPEN "C:\FABI.DRV" FOR OUTPUT AS 1
PRINT #1, "N C:\FABI.EX"
PRINT #1, "E 0100 4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00"
PRINT #1, "E 0110 B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00"
```

The content of the FABI.DRV will look like the following:

```
N C:\FABI.EX
E 0100 4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00 ; DOS EXE header
E 0110 B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00

[Virus body is cut from here]

E 4D20 10 0F 10 0F 10 0F 10 0F 10 0F 10 0F 10 0F 10 0F
E 4D30 10 0F 10 0F 10 0F 10 FF FF FF
RCX
4C3A
W
Q
```

Another BATCH file is also created by the macro in a manner similar to the debug script. This contains the command to drive DEBUG with the debug script:

```
DEBUG <C:\FABI.DRV >NUL
```

Note that DEBUG cannot create EXE files. At least, it cannot save them from memory with an EXE suffix. It can, however, save the content of memory easily without an EXE extension, which works when the file is loaded without an extension in the first place. This is the approach that W95/Fabi uses. It first saves the file with DEBUG as FABI.EX and uses yet another BATCH file to copy FABI.EX as FABI.EXE to run it.

Evidently, if DEBUG.EXE is not installed on the system or is renamed, some of these viruses cannot function completely or at all.

3.17.1 Intended Threats that Rely on a Debugger

Some malicious code might require the user to trace code in a debugger to replicate the virus. In some circumstances, this might happen easily in the case of macro threats. For instance, an error occurs during the execution of the malicious macro. Microsoft Word might then offer the user an option to run the macro debugger to resolve the cause of the problem. When the user selects the macro debugger command and traces the problem, the error might be bypassed. As a result, the virus code can replicate itself in this limited, special environment. There is an agreement between computer virus researchers, however, that such threats should be classified as intended.

3.18 Compiler and Linker Dependency

Several binary viruses spread their own source code during replication. This technique can be found in worms that target systems where binary compatibility is not necessarily provided. To enhance the replication of such worms on more than one flavor of Linux, the Linux/Slapper worm replicates its own source code to new systems. First, it breaks into the system via an exploit code, and then it uses gcc to compile and link itself to a binary. The worm encodes its source on the attacker's system and copies that over to the target system's temporary folder as a hidden file. Then it uses the uudecode command to decode the file:

```
/usr/bin/uudecode -o /tmp/.bugtraq.c /tmp/.uubugtraq;
```

The source code is compiled on the target with the following command:

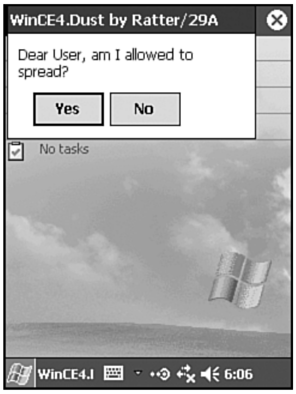
```
gcc -o /tmp/.bugtraq /tmp/.bugtraq.c –lcrypto;
```

The virus needs the crypto library to link its code perfectly, so not only must gcc be installed with standard source and header files on the target system, but the appropriate crypto libraries must also be available. Otherwise, the worm will not be able to infect the target system properly, although it might successfully penetrate the target by exploiting an Open SSL vulnerability.

The advantage of the source code-based infection method is the enhanced compatibility with the target operating system version. Fortunately, these techniques also have disadvantages. For example, it is a good practice to avoid installing sources and compilers on the path (unless it is absolutely necessary), greatly reducing the impact of such threats. Many system administrators tend to overlook this problem because it looks like a good idea to keep compilers at hand.

3.19 Device Translator Layer Dependency

Many articles circulated that concluded that no Windows CE viruses would ever be implemented, and for many years we did not know of any such creations. However, in July 2004, the virus writer, Ratter, released the first proof of concept virus, WinCE/Duts.1520, to target this platform, as shown in Figure 3.13.


WinCE4.Dust by Ratter/29A

Dear User, am I allowed to spread?

Figure 3.13 The message of the WinCE/Duts virus on an HP iPAQ H2200 Pocket PC.

Many recent devices run WinCE/Duts successfully because the ARM processor is available on a variety of devices, such as HP iPAQ H2200 (as well as many other iPAQ devices), the Sprint PCS Toshiba 2032SP, T-Mobile Pocket PC 2003, Toshiba e405, and Viewsonic V36, among others. Several additional GSM devices are built on the top of Pocket PC.

Interestingly, WinCE/Duts.1520 is able to infect Portable Executable files on several systems, despite the fact that the virus code looks “hard-coded” to a particular Windows CE release. For instance, the virus uses an ordinal-based function importing mechanism that would appear to be a serious limitation in attacking more than one flavor of Windows CE. In fact, it appears that the author of the virus believed that WinCE/Duts was only compatible with Windows CE 4. In our tests, however, we have seen the virus run correctly on Windows CE 3 as well.

It was not surprising that Windows CE was not attacked by viruses for so long. Windows CE was released on a variety of processors that create incompatibility issues (an inhomogeneous environment) and appear somewhat to limit the success of such viruses.

In addition, Windows CE does not support macros in Microsoft products such as Pocket Word or Pocket Excel, but there might be some troubling threats to come.

Prior to Windows CE 3.0, it was painful to create and distribute Windows CE programs because of binary compatibility issues. The compiled executables were developed in binary format as portable executable (PE) files, but the executable could only run on the processor on which it was compiled. So for each different device, the developer must compile a compatible binary. This can be a time-consuming process for both the developer and the user (who is impatient to install new executables).

The CPU dependency is hard-coded in the header of PE files. For instance, on the SH3 processor, the PE file header will contain the machine type 0x01A2, and its code section will contain compatible code only for that architecture.

Someone can easily create an application that is compiled to run on an SH3 platform; however, Windows CE was ported to support several processors, such as the SH3, SH4, MIPS, ARM, and so on. Consequently, a native Windows CE virus would be unable to spread easily among devices that use different processors. For example, WinCE/Duts.1520 will not infect SH3 processor-based systems.

Virus writers might be able to create a Win32 virus that drops a Windows CE virus via the Microsoft Active Sync. Such a virus could easily send mail and propagate its Intel version (with an embedded Pocket version), but it would only be able to infect a certain set of handheld devices that use a particular processor. In the future, this problem is going to be less of an issue for developers as more compatible processors are released. For example, the new XScale processors are compatible with the ARM series. XScale appears not only in Pocket PC systems, but in Palm devices as well. Obviously, this opens up possibilities for the attackers to create “cross-platform” viruses to target Palm as well as Pocket PC systems with the same virus.

Microsoft developed a new feature on the Pocket PC that made the Windows CE developers’ jobs easier. In the Pocket PC, Microsoft started to support a new executable file format: the common executable file (CEF) format.

CEF executables can be compiled with Windows CE development tools, such as eMbedded Visual C++ 3.0. A CEF executable is basically a special kind of PE file. CEF is a processor-neutral code format that enables the creation of portable applications across CPUs supported by Windows CE. In fact, CEF contains MSIL code.

In eMbedded Visual C++, CEF tools (compilers, linkers, and SDK) are made available to the developer the same way that a specific CPU target (such as MIPS or ARM) is selected. When a developer compiles a CEF application, the compiler and linker do everything but generate machine-specific code. You still get a DLL or EXE, but the file contains intermediate language instructions instead of native machine code instructions.

CEF enables WindowsCE application developers to deliver products that support all the CPU architectures that run the WindowsCE 3.0 and above operating systems. Because CEF is an intermediate language, processor vendors can easily add a new CPU family that runs CEF applications. For instance, HP Jornada 540 comes with such a built-in device translator layer. The CEF file might have an EXE extension when distributed, so nothing really changes from the user's perspective.

The device translator is specific to a particular processor and WindowsCE device. The device version normally translates a CEF executable to the native code of that processor when the user installs the CEF executable on the device. This occurs seamlessly, without any indication to the user, other than a brief pause for translation after the executable is clicked on. An operating system hook catches any attempt to load and execute a CEF EXE, DLL, or OCX file automatically and invokes the translator before running the file.

For example, if the Pocket PC is built on an SH3 processor, the translator layer will attempt to compile the CEF file to an SH3 format. The actual CEF executable will be replaced by its compiled SH3 native version, changing the content of the file completely to a native executable. Indeed, the first reincarnation of MSIL, JIT (Just-in-Time) compiling on Pocket PC rewrites the executables themselves on the file system.

Obviously, virus writers might take advantage of the CEF format in the near future. A 32-bit Windows virus could easily install a CEF version of itself to the Pocket device, allowing it to run on all Pocket PC devices because the OS would translate the CEF executable to native format. We can only hope that CEF will not be supported on systems other than Windows CE. A desktop implementation, for example, would be very painful to see in case the operating system would rewrite CEF objects to native executables.

Because executables are converted to new formats on the fly, the content of the file changes. This is an even bigger problem than the up-conversion of Macro viruses in Office products[50]. Obviously, this is going to be a challenge for antivirus software, integrity checkers, and behavior blocker systems.

First of all, it is clearly a major problem for antivirus software given that the virus code needs to be detected and identified in all possible native translations as well as the original MSIL form. If the MSIL virus is executed on a device, before a signature of the virus is known to the antivirus program, the virus will run and its code will be converted to any of a number of native formats according to the actual type of the system. As a result, the MSIL signature of the virus will not be useful to find the virus afterwards. The virus needs to be detected in all possible native translations as well, but this task is not trivial.

It is a problem for the integrity checkers because the content of the program changes on the disk, not only in memory. As a result, integrity checkers cannot be sure if the change was the result of a virus infection or a simple native code translation. Finally, it is a problem for behavior blocker systems because the content of an executable is changed on the disk, which easily can be confused with virus activity.

3.20 Embedded Object Insertion Dependency

The first known binary virus that could infect Word 6 documents, called Anarchy.6093[51], appeared in 1997. Not surprisingly, we have not seen many other viruses like this because attacking the document formats to add macros to them is no trivial task. Anarchy was a DOS-based COM, EXE, and DOC file infector.

The first virus to infect VBA documents from binary code was released from Russia. The virus is called {Win32,W97M}/Beast.41472.A, and it appeared in the wild in April 1999. The virus is written in Borland Delphi and compiled to 32-bit PE format.

Beast uses a different means of infection than other binary viruses that infect documents. Instead of having it attack the VBA format on a bit-by-bit level, the Beast author used OLE (Object Linking and Embedding) APIs, such as AddOLEObject(), to inject macro code and embedded executable code into documents by using the internal OLE support of Microsoft Word. Via OLE support, the virus injects an embedded object (executable) into VBA documents. However, this embedded object will not be visible to the user, as it normally would. This is because the virus uses a trick to hide the icon of the embedded object.

The virus looks for actively opened documents in Word. When a handle to an active document is available, the virus calls its infection module. First it tries to check whether there are no embedded objects in the document, but in some cases this routine fails because the virus might have added multiple embedded executables into the documents.

Next, Beast tries to add itself as C:\I.EXE shape into the document, named 3BEPb (Russian for *beast*). If this procedure goes as planned, then a new macro called AutoOpen()also will be injected into the document.

The execution of the embedded object is facilitated by using the Activate method for the 3BEPb shape in the active document:

```
ActiveDocument.Shapes("3BEPb").Activate
```

Beast introduced the need for detection and removal of malicious embedded objects in documents—not the simplest problem to solve.

3.21 Self-Contained Environment Dependency

One interesting dependency appears when malicious code carries its own environment to the platform. The W32/Franvir virus family offers a good example.

Franvir is clearly a Win32 application. It is compiled with Borland Delphi to a 32-bit PE program. However, the actual Win32 binary part is known as the Game Maker, written by Mark Overmars of the Netherlands (`http://www.cs.uu.nl/people/markov/gmaker/doc.html`).

The Franvir virus was written by a French virus writer using the script language of Game Maker, called GML (Game Maker Language). This is only available in the registered version of Game Maker, which provides developers with security options for using these functions (turning them on and off). It is up to the developer to set the security settings; therefore, a malicious author can easily use GML of Game Maker for virus writing.

Game Maker is a professional game developer environment. Hundreds of brilliant games have been created in it by professionals. It can be used to develop all kinds of games, including scrolling shooters, puzzle games, and even isometric games. For instance, the shooter game called *Doomed* was created using Game Maker.

GML provides functions for Registry, File, and program execution. The File operation functions are extremely rich and provide high flexibility for game developers to install and execute programs—but they also can be used by malicious attackers. Some of the functions of GML include the following:

```
file_exists(fname)
file_delete(fname)
file_copy(fname,newname)
file_open_write(fname)
directory_create(dname)
file_find_first(mask,attr)
file_find_next()
file_attributes(fname,attr)
registry_write_string_ext()
```

GML scripts are stored in the resources of Game Maker, but they are accessed and executed by the environment, the interpreter in Game Maker itself. Franvir is an encrypted GML script. It copies itself all over the hard disk under various existing program names. It also installs itself to local P2P (peer-to-peer) folders or even creates the shared folder for KaZaA if the directory is not installed ("kazaa\my shared folder\") and changes the KaZaA settings to share the folder. Furthermore, it does damage by deleting the win.com file of Windows. Thus, ultimately Franvir must be classified as a Win32 P2P worm. In reality, however, it is a GML script that is carried by its own environment to new platforms. When the virus successfully executes, it eventually uses the show_message() function to display the false error message shown in Figure 3.14.

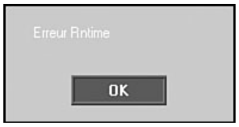

Erreur Rntime

Figure 3.14 The false error message of Franvir.

The virus could ultimately offer to play a game such as the DOS virus, Playgame, instead of executing the malicious file delete action as an activation routine, but well...what can we expect from a typical virus writer?

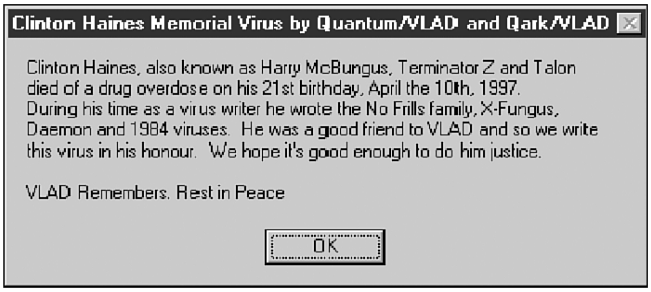
3.22 Multipartite Viruses

The first virus that infected COM files and boot sectors, Ghostball, was discovered by Fridrik Skulason in October 1989. Another early example of a multipartite virus was Tequila. Tequila could infect DOS EXE files as well as the MBR (master boot sector) of hard disks.

Multipartite viruses are often tricky and hard to remove. For instance, the Junkie virus infects COM files and is also a boot virus. Junkie can infect COM files on the hidden partitions[52] that some computer manufacturers use to hide data and extra code by marking the partition entries specifically. Because Junkie loads to memory before these hidden files are accessed, these files can get infected easily. Scanners typically scan the content of the visible partitions only, so such infections often lead to mysterious reinfections of the system. This is because the virus has been cleaned from everywhere but from the hidden partition, so the virus can infect the system again as soon as the hidden partition is used to run one of the infected COM files.

In the past, boot and multipartite viruses were especially successful at infecting machines that used the DOS operating system. On modern Windows systems, such viruses are less of a threat, but they do exist.

The Memorial virus[53] introduced DOS COM, EXE, and PE infection techniques in the same virus. The payload of the Memorial virus is show in Figure 3.15.


Clinton Haines Memorial Virus by Quantum/VLAD and Qark/VLAD

Clinton Haines, also known as Harry McBungus, Terminator Z and Talon died of a drug overdose on his 21st birthday, April the 10th, 1997.
During his time as a virus writer he wrote the No Frills family, X-Fungus, Daemon and 1984 viruses. He was a good friend to VLAD and so we write this virus in his honour. We hope it's good enough to do him justice.

VLAD Remembers. Rest in Peace

Figure 3.15 The message of the W95/Memorial virus.

W95/Memorial also used the VxD (Virtual Device Driver) format of Windows 9x systems to load itself into kernel mode and hook the file system to infect files on the fly whenever they were accessed. As a result, Memorial also infects 16-bit and 32-bit files.

Another interesting example of a multipartite infection is the Russian virus, 3APA3A, which was found in the wild in Moscow in October 1994[54]. 3APA3A is a normal boot virus on a diskette, occupying two sectors for itself, but it uses a special infection method on the hard disk. It infects the DOS core file IO.SYS. First it makes a copy of IO.SYS, and then it overwrites the original. After the infection, the root directory contains two IO.SYS files, but the first is set as a volume label of the disk; thus, the DIR command does not display two files, but a volume label "IO SYS" and a single IO.SYS file. The point is to trick DOS into loading the infected copy of IO.SYS. Then the virus starts the original one after itself. This happens because DOS will load the first IO.SYS file regardless of its attributes. This method represents a special subclass of companion infection techniques.

3.23 Conclusion

New viral environments are discovered each year. Over the last 20 years of PC viruses, there has been tremendous dark energy in place to develop computer viruses for almost every platform imaginable. All over the world, thousands of people created computer viruses. Because of this we are experiencing an ever-growing security problem with malicious code and, consequently, seeing the development of computer virus research as a new scientific field. There is absolutely no question whether computer viruses will stay with us and evolve to future platforms in the upcoming decades.

Fred Cohen's initial research with computer viruses in 1984 concluded that the computer virus problem is ultimately an integrity problem. Over the last 20 years, the scope of integrity expanded dramatically from file integrity to the integrity of applications and operating system software. Modern computer viruses, such as W32/CodeRed and W32/Slammer, clearly indicate this new era: Computer viruses cannot be controlled by file-based integrity checking alone because they jump from system to system over the network, injecting themselves into new process address spaces in such a way that they are never stored on the disk.

Computer viruses changing their environments to suit their needs is a problem that will likely begin to emerge. For example, the W32/Perrun virus appends itself to JPEG picture files. Normally, pictures files are not infectious unless some serious vulnerability condition exists in a picture file viewer (such as the one described in *Microsoft Security Bulletin* MS04-028[55]). However, Perrun modifies the environment of the infected host to include an extractor component, resulting in Perrun-compromised JPEG files not being infectious on a clean system but on infected computers only. Such computer viruses can modify the host's environment in such a way that previous assumptions about the environments no longer hold.

References

1. Dr. Vesselin Bontchev, "Methodology of Computer Anti-Virus Research," *University of Hamburg, Dissertation*, 1998.
2. Dr. Harold Highland, "A Macro Virus," *Computers & Security*, August 1989, pp. 178-188.
3. Joe Wells, "Brief History of Computer Viruses," 1996, `http://www.research.ibm.com/antivirus/timeline.htm`.
4. Dr. Peter Lammer, "Jonah's Journey," *Virus Bulletin*, November 1990, p. 20.
5. Peter Ferrie, personal communication, 2004.
6. Jim Bates, "WHALE...A Dinosaur Heading For Extinction," *Virus Bulletin*, November 1990, pp. 17-19.
7. Eric Chien, "Malicious Threats to Personal Digital Assistants," *Symantec*, 2000.
8. Dr. Alan Solomon, "A Brief History of Viruses," *EICAR*, 1994, pp. 117-129.
9. Intel Pentium Processor III Specification Update, `http://www.intel.com/design/PentiumIII/specupdt/24445349.pdf`.
10. Mikko Hypponen, Private Communication, 1996.
11. Thomas Lipp, "Computerviren," *64'er, Markt&Technik*, March 1989.
12. Peter Szor, "Stream of Consciousness," *Virus Bulletin*, October 2000, p. 6.
13. Peter Szor and Peter Ferrie, "64-bit Rugrats," *Virus Bulletin*, July 2004, pp. 4-6.
14. Marious Van Oers, "Linux Viruses—ELF File Format," *Virus Bulletin Conference*, 2000, pp. 381-400.
15. Jakub Kaminski, "Not So Quiet on the Linux Front: Linux Malware II," *Virus Bulletin Conference*, 2001, pp. 147-172.
16. Eugene Kaspersky, "Shifter.983," `http://www.viruslist.com, 1993`.
17. Sarah Gordon, "What a (Winword.) Concept," *Virus Bulletin*, September 1995, pp. 8-9.

18. Sarah Gordon, "Excel Yourself!" *Virus Bulletin*, August 1996, pp. 9-10.

19. Yoshihiro Yasuda, personal communication, 2004.

20. Dr. Igor Muttik, "Macro Viruses—Part 1," *Virus Bulletin*, September 1999, pp. 13-14.

21. Dr. Vesselin Bontchev, "The Pros and Cons of WordBasic Virus Up-conversion," *Virus Bulletin Conference*, 1998, pp. 153-172.

22. Dr. Vesselin Bontchev, "Possible Macro Virus Attacks and How to Prevent Them," *Virus Bulletin Conference*, 1996, pp. 97-127.

23. Dr. Vesselin Bontchev, "Solving the VBA Up-conversion Problem," *Virus Bulletin Conference*, 2001, pp. 273-300.

24. Nick FitzGerald, "If the CAP Fits," *Virus Bulletin*, September 1999, pp. 6-7.

25. Jimmy Kuo, "Free Anti-Virus Tips and Techniques: Common Sense to Protect Yourself from Macro Viruses," *NAI White Paper*, 2000.

26. Dr. Vesselin Bontchev, personal communication, 2004.

27. Jakub Kaminski, "Disappearing Macros—Natural Devolution of Up-converted Macro Viruses," *Virus Bulletin Conference*, 1998, pp. 139-151.

28. Katrin Tocheva, "Multiple Infections," *Virus Bulletin*, 1999, pp. 301-314.

29. Dr. Richard Ford, "Richard's Problem," private communication on *VMACRO* mailing list, 1997.

30. Dr. Vesselin Bontchev, "Macro Virus Identification Problems," *Virus Bulletin Conference*, 1997, pp. 157-196.

31. Dr. Vesselin Bontchev, private communication, 1998.

32. Vesselin Bontchev, "No Peace on the Excel Front," *Virus Bulletin*, April 1998, pp. 16-17.

33. Gabor Szappanos, "XML Heaven," *Virus Bulletin*, February 2003, pp. 8-9.

34. Peter G. Capek, David M. Chess, Alan Fedeli, and Dr. Steve R. White, "Merry Christmas: An Early Network Worm," *IEEE Security & Privacy*, `http://www.computer.org/security/v1n5/j5cap.htm`.

35. Dr. Klaus Brunnstein, "Computer 'Beastware': Trojan Horses, Viruses, Worms—A Survey," *HISEC'93*, 1993.

36. David Ferbrache, "A Pathology of Computer Viruses," Springer-Verlag, 1992, ISBN: 3-540-19610-2.

37. Peter Szor, "Warped Logic?" *Virus Bulletin*, June 2001, pp. 5-6.

38. Mikhail Pavlyushchik, "Virus Mapping," *Virus Bulletin*, November 2003, pp. 4-5.

39. Eugene Kaspersky, "Don't Press F1," *Virus Bulletin*, January 2000, pp. 7-8.

40. Peter Szor, "Sharpei Behaviour," *Virus Bulletin*, April 2002, pp. 4-5.

41. Gabor Kiss, "SWF/LFM-926—Flash in the Pan?" *Virus Bulletin*, February 2002, p. 6.

42. Dmitry Gryaznov, private communication, 2004.

43. Sami Rautiainen, private communication, 2004.

44. Philip Hannay and Richard Wang, "MSIL for the .NET Framework: The Next Battleground?," *Virus Bulletin Conference*, 2001, pp. 173-196.

45. Peter Szor, "Tasting Donut," *Virus Bulletin*, March 2002, pp. 6-8.

46. Peter Ferrie, "The Beagle Has Landed," `http://www.virusbtn.com/resources/viruses/indepth/beagle.xml`.

47. Eugene Kaspersky, "Zhengxi: Saucerful of Secrets," *Virus Bulletin*, April 1996, pp. 8-10.

48. Roger A. Grimes, *Malicious Mobile Code*, O'Reilly, 2001, ISBN: 1-56592-682-X (Paperback).

49. Ken Thomson, "Reflections on Trusting Trust," *Communication of the ACM*, Vol. 27, No. 8, August 1984, pp. 761-763, `http://cm.bell-labs.com/who/ken/trust.html`.

50. Peter Szor, "Pocket Monsters," *Virus Bulletin*, August 2001, pp. 8-9.

51. Igor Daniloff, "Anarchy in the USSR," *Virus Bulletin*, October 1997, pp. 6-8.

52. Lakub Kaminski, "Hidden Partitions vs. Multipartite Viruses—I'll be back!," *Virus Bulletin Conference*, 1996.

53. Peter Szor, "Junkie Memorial," *Virus Bulletin*, September 1997, pp. 6-8.

54. Dr. Igor Muttik, "3apa3a," `http://www.f-secure.com/v-descs/3apa3a.shtml`, 1994.

55. "Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution," MS04-028, `http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx`.

Index