111-61-CR
217911
15P.

Computer Viruses

Peter J. Denning

21 Mar 1988

RIACS Technical Report TR-88.10

NASA Cooperative Agreement Number NCC 2-387

(NASA-CR-184680) COMPUTER VIRUSES (Research Inst. for Advanced Computer Science) 15 p CSCL 09B

N89-26421

Unclas

G3/61 0217911

Research Institute for Advanced Computer Science

Computer Viruses

Peter J. Denning

21 Mar 1988

RIACS Technical Report TR-88.10

NASA Cooperative Agreement Number NCC 2-387

Computer Viruses

Peter J. Denning

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report TR-88.10
21 Mar 1988

The worm, Trojan horse, bacterium, and virus are destructive programs that attack information stored in a computer's memory. Virus programs, which propagate by incorporating copies of themselves into other programs, are a growing menace in the late-1980s world of unprotected, networked workstations and personal computers. Limited immunity is offered by memory protection hardware, digitally authenticated object programs, and antibody programs that kill specific viruses. Additional immunity can be gained from the practice of digital hygiene, primarily the refusal to use software from untrusted sources. Full immunity requires attention in a social dimension, the accountability of programmers.

This is a preprint of the column *The Science of Computing* for *American Scientist 76*, No 3 (May-June 1988).

Work reported herein was supported in part by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

Computer Viruses

Peter J. Denning

Research Institute for Advanced Computer Science

21 Mar 1988

Sometime in the middle 1970s, the network of computers at a Silicon Valley research center was taken over by a program that loaded itself into an idle workstation, disabled the keyboard, drew random pictures on the screen, and monitored the network for other idle workstations to invade. The entire network and all the workstations had to be shut down to restore normal operation.

In early September 1986, a talented intruder broke into a large number of computer systems in the San Francisco area, including 9 universities, 15 Silicon Valley companies, 9 ARPANET sites, and 3 government laboratories. The intruder left behind recompiled login programs to simplify his return. His goal was apparently to achieve a high score on the number of computers cracked; no damage was done (*1*).

In December 1987, a Christmas message that originated in West Germany propagated into the Bitnet network of IBM machines in the United States. The message contained a program that displayed an image of a Christmas tree and sent copies of itself to everyone in the mail distribution list of the user for whom it was running. This prolific program rapidly clogged the network with a geometrically growing number of copies of itself. Finally the network had to be shut down until all copies could be located and expurgated.

For two months in the fall of 1987, a program quietly incorporated copies of itself into programs on personal computers at the Hebrew University. It was discovered and dismantled by a student, Yuval Rakavy, who noticed that certain library programs were growing longer for no apparent reason. He isolated the errant code and discovered that if executed on certain Fridays the thirteenth the computer running it would slow down by 80%, and on Friday 13 May 1988, it would erase all files. That date was the fortieth anniversary of the last day Palestine was recognized as a separate political entity. Rakavy designed another program that detected and erased all copies of the errant program it could find. Even so, he could not be completely sure he had eradicated it.

These four incidents illustrate the major types of programs that attack other programs in a computer's memory. The first type is a *worm*, a program that invades a workstation and disables it. The second is a *Trojan horse*, a program that performs some apparently useful function, such as login, while containing hidden code that performs an unwanted, usually malicious function.

This name is inspired by the legendary wooden horse built by the Greek army, ostensibly as an offering to Athena, which in the dark of night disgorged its bellyful of murderous soldiers into the sleeping streets of Troy. The third type is a *bacterium*, a program that replicates itself and feeds off the host system by preempting processor and memory capacity. The fourth is a *virus*, a program that incorporates copies of itself into the machine codes of other programs and, when those programs are invoked, wreaks havoc in the manner of a Trojan horse.

I can cite numerous other incidents in which information stored in computers has been attacked by hostile programs. An eastern medical center lost nearly 40% of its records to a malicious program in its system. Students at Lehigh University lost homework and other data when a virus erased diskettes inserted into campus personal computers. Some programs available publicly from electronic bulletin boards have destroyed information on the disks of computers into which they were read. A recent *New York Times* article (*2*) describes many examples and documents the rising concern among computer network managers, software dealers, and personal computer users about these forms of electronic vandalism. In an effort to alert concerned computer scientists to the onslaught, the Association for Computing Machinery sponsors the Computer Risks Forum, an electronic newsletter moderated by Peter G. Neumann of SRI International, which regularly posts notices and analyses of such dangers.

The recent rash of viral attacks has drawn everyone's attention to the more general problem of computer security, a subject of great complexity which has fascinated researchers since the early 1960s (*3*). The possibility of pernicious programs propagating through a file system has been known for at least twenty-five years. In his May 1985 Computer Recreations column in *Scientific American*, Kee Dewdney documented a whole menagerie of beastly threats to information stored in computer memories, especially those of personal computers (*4*), where an infected diskette can transmit a virus to the main memory of the computer, and thence to any other diskette (or to hard disk). Ken Thompson, a principal designer of UNIX^{TM}, and Ian Witten have documented some of the more subtle threats to computers that have come to light in the 1980s (*5,6*).

It is important to keep in mind that worms, Trojan horses, bacteria, and viruses are all programs designed by human beings. Although a discussion of these menaces brings up many intriguing technical issues, we should not forget that at the root of the problem are programmers performing disruptive acts under the cloak of anonymity conveniently provided by many computer systems.

I will focus on viruses, the most pernicious of the attacks against information in computers. A virus is a code segment that has been incorporated into the body of another program, "infecting" it. When the virus code is executed, it locates a few other uninfected programs and infects them; in due course, the number of infected programs can grow quite large. Viruses can spread with remarkable speed: in experimental work performed in 1983 and 1984, Fred

Cohen of the University of Cincinnati demonstrated that a simple virus program can propagate to nearly every part of a normally operating computer system within a matter of hours. Most viruses contain a marker that allows them to recognize copies of themselves; this avoids discovery, because otherwise some programs would get progressively longer under multiple infections. The destructive acts themselves come later: any copy of the virus that runs after some appointed date will perform such an unwanted function.

A Trojan horse program is the most common means of introducing a virus into a system. It is possible to rig a compiler with an invisible Trojan horse that implants another Trojan horse into any selected program during compilation.

A virus that takes the form of statements inserted into the high-level language version of a program -- that is, into the source file -- can possibly be detected by an expert who reads the program, but finding such a program in a large system can be extremely difficult. Many viruses are designed to evade detection completely by attaching themselves to object files, the machine coded images of high-level program sources that are produced by compilation. These viruses cannot be detected from a reading source programs.

The first serious discussions of Trojan horses took place in the 1960s. Various hardware features were developed to reduce the chances of attack (*3*), including virtual memory, which restricts a program's to a limited region of memory, its "address space" (*7*). All these features are based on the principle of least privilege, which reduces the set of accessible objects to the minimum a program needs in order to perform its function. Because a suspect program can be run in a strictly confined mode, any Trojan horse it contains will be unable to perform much damage.

How effective is virtual memory against viruses? Memory protection hardware can significantly reduce the risk, but a virus can still propagate to legitimately accessible programs, including portions of the operating system. The rate of propagation may be slowed by virtual memory, but propagation is not stopped. Most PCs are especially vulnerable because they have no memory protection hardware at all; an executing program has free access to anything in memory or on disk. A network of PCs is even more vulnerable, because any PC can propagate an infected copy of a program to any other PC, no questions asked.

What can be done to protect against viruses in a computer or workstation without memory protection hardware or controls on access to files? One common proposal is to retrofit the operating system with a write query check that asks the user for permission to allow the running program to modify a file. This gives the user an opportunity to determine that the program is attempting to gain acces to unauthorized files. It is, unfortunately, hardly workable even for experienced programmers because of the difficulty of discovering which files a running program must legitimately modify. A design that suppresses write queries for files named in an authorization list associated with a program can be subverted by a virus that adds the name of the unauthorized file to the list before attacking it.

A more powerful immunization scheme is based on digital signatures of object files. When a program is installed in a system, an authenticator is created by producing a checksum that depends on all the bits of a file, which is then signed with the secret key of the person who stored the file (*8*). The authenticator can be unlocked by applying the public key of that person. A user can confirm that a file is an exact copy of what was stored by computing its checksum and comparing that with the unlocked authenticator. A program infected by a virus would fail this test. Without access to the secret key, the designer of the virus could not produce a valid authenticator for the infected program. This scheme also works for programs obtained from trusted sources over a network: each program comes with an authenticator sealed by the trusted producer.

One way to implement this scheme is to equip the operating system with a background process that randomly checks files against their authenticators. If a virus has entered the system, this process will eventually discover an infected file and raise the alarm. Another way to implement this scheme is to "innoculate" an object program by placing an authentication subroutine at its entry point. This implementation is slow, however, and can be defeated by a virus that invades entry points: by the time the authenticator gets control, the virus will already have acted.

The authenticator scheme relies on the protection of the secret key, which cannot be complete unless the key is kept outside the system. It also rests on the integrity of the system itself: for example, a sophisticated attack against the program that reports whether a file has been infected could disable the scheme.

A program called an antibody can offer limited remedies should a virus penetrate a system. Such a program examines an object file to determine whether a known virus has been incorporated. It may also remove the virus from the infected program. This limited form of protection can be very effective against known viruses, but it cannot identify new ones.

As we have seen, each of the major technical mechanisms -- memory protection hardware, digital-signature authenticators, and antibodies -- offers limited protection against viruses (and Trojan horses). Can the operating procedures followed by those who use a computer system lower the risk further?

Yes! An additional measure of protection can be obtained by care in the way one uses a computer. Analogies with food and drug safety are helpful. Just as one would not consider purchasing food or capsules in unsealed containers or from untrusted sources, one can refuse to use any unsealed software or software from untrusted sources. Never insert a diskette that has no manufacturer's seal into your PC. Never use a program borrowed from someone who does not practice digital hygiene to your own standards. Beware of software obtained from public bulletin boards. Purchase programs that check other programs for known viruses. Be wary of public domain software (including virus eradicators!). Monitor the last-modified dates of programs and files. Don't execute programs sent in electronic mail -- even your friends may have inadvertently forwarded a virus.

Don't let employees bring software from home.

The problem of viruses is difficult, both technically and operationally, and no solution oriented entirely along technical or operational lines can be complete. There is a third, social dimension to the problem: we don't know how to hold people fully accountable for the actions of their programs in a networked system of computers. A complete solution must involve all three dimensions.

Computer scientists are divided over whether it serves the field to publish accounts of viral attacks in full technical detail. (This article, being superficial, does not count.) Some hold that revelations of technical detail -- as in Dewdney (*4*) or Witten (*6*) -- are reprehensible because they give the few would-be perpetrators a blueprint for actions that can make life exceedingly difficult for the many innocent users, and because there are few successful defenses against the attacks. Others hold that the main hope for a long term solution is to mobilize the "good guys" by setting forth the problems in detail; the short term risk, according to this view, is offset by the long-term gain. Most computer scientists favor this way of mobilizing forces to oppose computer sabotage.

References

1. B. Reid. 1987. Reflections on some recent widespread computer breakins. *ACM Communications 30*, 2. February. 103-105.

2. Vin McLellan. 1988. Computer systems under siege. *NY Times* Sunday Business Section. January 31.

3. D. E. Denning. 1982. *Cryptography and Data Security.* Addison-Wesley.

4. A. K. Dewdney. 1985. Computer Recreations (A Core War Bestiary of Viruses, Worms, and other Threats to Computer Memories). *Scientific American 252*, 3. March. 14-23.

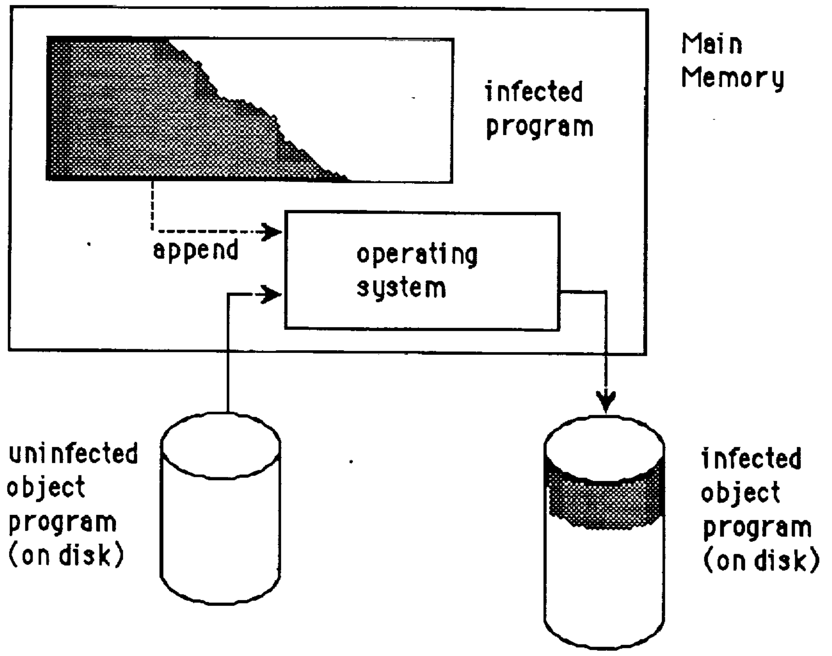
5. K. Thompson. 1984. Reflections on trusting trust. *ACM Communications 27*, 8. August. 172-80.

6. Ian H. Witten. 1987. Computer (In)security: Infiltrating Open Systems. *Abacus 4*, 4. Summer. 7-25.

7. P. J. Denning. 1986. Virtual memory. *American Scientist 74*, 3 (May-June). 227-229.

8. P. J. Denning. 1987. Security of data in networks. *American Scientist 75*, 1 (January-February). 12-14.

Box 1: How a Virus Works

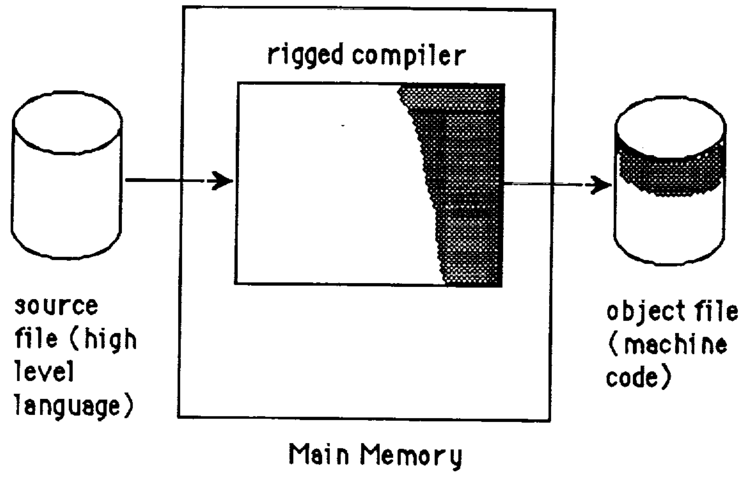

Main Memory
infected program
append
operating system
uninfected object program (on disk)
infected object program (on disk)

A program infected with a virus (shaded area) and loaded and executing in the main memory of a computer can infect another executable (object) program in the computer's disk storage system by secretly requesting the computer's operating system to append a copy of the virus code to the object program usually at the start. The infection makes the object program slightly longer.

When the newly infected program is itself loaded into memory and invoked, the virus in it takes control and performs hidden functions, such as infecting yet other object programs. The virus may also perform destructive functions before transferring control to the original entry point. The virus code contains a marker so that a virus won't attempt to infect a program already infected by its own kind: multiple infections will cause an object file to grow ever larger, leading to easy detection.

The same principle works in personal computers (PCs) , where floppy disks play the role of object files in the description above. In this case, the virus usually attacks the copy of the operating system contained on the floppy disk so that the virus is automatically invoked whenever the disk's operating system is started. Since the operating system then resides in the PC's main memory, it can infect any diskettes inserted into the PC.

Box 2: A Trojan Horse in a Compiler



A Trojan Horse is a useful program containing hidden code that performs an unwanted, mischevious function. It might copy the invoker's private files into an area of memory belonging to the Trojan Horse's designer, thereby circumventing the owner's file protection. It might obtain access to a subsystem normally inaccessible to the designer. A Trojan Horse that destroys or erases files is also called a logic bomb.

It is sometimes suggested that Trojan Horses can be detected by scanning the borrowed program's source file for statements that perform operations outside the program's specifications. Ken Thompson, one of the principal designers of UNIX™, has pointed out that this approach is fundamentally incomplete. He demonstrated how to rig a compiler to introduce a Trojan Horse into the object file of any other selected program, for example the login program (5). Whenever the login program is recompiled, the rigged compiler always inserts a segment of code that allows login whenever a special password (known only to the Trojan Horse's designer) is given. The login program's Trojan Horse cannot be detected by reading the login program's source file.

Now, it might seem that a careful reading of the rigged compiler's source file would reveal the Trojan Horse that inserts the login Trojan Horse. But this is not so. The rigged compiler is itself an object code, and can thereby contain its own Trojan Horse without a record in the compiler's source file. Thompson demonstrated a scheme to rig a compiler in this way (5,6).