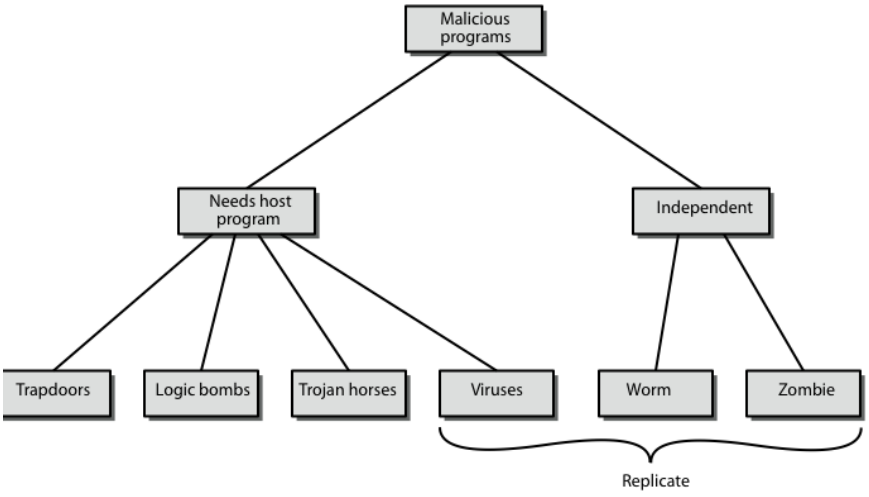
MALICIOUS SOFTWARE

This chapter examines malicious software (malware), especially viruses and worms, which exploit vulnerabilities in computing systems. These have been given a lot of (often uninformed) comment in the general media. They are however, of serious concern, and are perhaps the most sophisticated types of threats to computer systems. We begin with a survey of various types of malware, with a more detailed look at the nature of viruses and worms. We then turn to distributed denial-of-service attacks.

Viruses and Other Malicious Content

- computer viruses have got a lot of publicity
- one of a family of **malicious software**
- effects usually obvious
- have figured in news reports, fiction, movies (often exaggerated)
- getting more attention than deserve
- are a concern though

Malicious Software



Reference :William Stallings, Cryptography and Network Security: Principles and Practice, PHI 3rd Edition, 2006

The terminology used for malicious software presents problems because of a lack of universal agreement on all terms and because of overlap. Stallings Table 21.1, and this diagram from 3/e, provide a useful taxonomy. It can be divided into two categories: those that need a host program

(being a program fragment eg virus), and those that are independent programs (eg worm); alternatively you can also differentiate between those software threats that do not replicate (are activated by a trigger) and those that do (producing copies of themselves). Will now survey this range of malware.

Backdoor or Trapdoor

A backdoor, or trapdoor, is a secret entry point into a program that allows someone that is aware of it to gain access without going through the usual security access procedures. Have been used legitimately for many years to debug and test programs, but become a threat when left in production programs, allowing intruders to gain unauthorized access. It is difficult to implement operating system controls for backdoors. Security measures must focus on the program development and software update activities.

Logic Bomb

One of the oldest types of program threat, predating viruses and worms, is the logic bomb. The logic bomb is code embedded in some legitimate program that is set to "explode" when certain conditions are met. Examples of conditions that can be used as triggers for a logic bomb are the presence or absence of certain files, a particular day of the week or date, or a particular user running the application. Once triggered, a bomb may alter or delete data or entire files, cause a machine halt, or do some other damage.

Trojan Horse

A Trojan horse is a useful, or apparently useful, program or command procedure (eg game, utility, s/w upgrade etc) containing hidden code that performs some unwanted or harmful function that an unauthorized user could not accomplish directly. Commonly used to make files readable, propagate a virus or worm or backdoor, or simply to destroy data.

Mobile Code

Mobile code refers to programs (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics. The term also applies to situations involving a large homogeneous collection of platforms (e.g., Microsoft Windows). Mobile code is transmitted from a remote system to a local system and then executed on the local system without the user's explicit instruction. Mobile code often acts as a mechanism for a virus, worm, or Trojan horse to be transmitted to the user's workstation. In other cases, mobile code takes advantage of vulnerabilities to perform its own exploits, such as unauthorized data access or root compromise. Popular vehicles for mobile code include Java applets, ActiveX, JavaScript, and VBScript. The most common ways of using mobile code for malicious operations on local system are cross-site scripting, interactive and dynamic Web sites, e-mail attachments, and downloads from untrusted sites or of untrusted software.

Multiple-Threat Malware

Viruses and other malware may operate in multiple ways.

A **multipartite** virus infects in multiple ways. Typically, the multipartite virus is capable of infecting multiple types of files, so that virus eradication must deal with all of the possible sites of infection.

A **blended** attack uses multiple methods of infection or transmission, to maximize the speed of contagion and the severity of the attack. Some writers characterize a blended attack as a package that includes multiple types of malware. An example of a blended attack is the Nimda attack, erroneously referred to as simply a worm. Nimda has worm, virus, and mobile code characteristics. Blended attacks may also spread through other services, such as instant messaging and peer-to-peer file sharing.

Viruses

A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs. A virus can do anything that other programs do. The difference is that a virus attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs. Most viruses carry out their work in a manner that is specific to a particular operating system and, in some cases, specific to a particular hardware platform. Thus, they are designed to take advantage of the details and weaknesses of particular systems. During its lifetime, a typical virus goes through the following four phases:

• **Dormant phase:** The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.

• **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.

• **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.

• **Execution phase:** The function is performed, which may be harmless, e.g. a message on the screen, or damaging, e.g. the destruction of programs and data files

Virus Structure

A computer virus has three parts [AYCO06]:

• Infection mechanism:The means by which a virus spreads, enabling it to replicate. The mechanism is also referred to as the infection vector.

• Trigger: event or condition determining when the payload is activated or delivered.

• Payload: What the virus does, besides spreading. The payload may involve damage or may involve benign but noticeable activity.

A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable. The lack of access controls on early PCs is a key reason why traditional machine code based viruses spread rapidly on these systems. In contrast, while it is easy enough to write a machine code virus for UNIX systems, they were almost never seen in practice due to the existence of access controls on these systems prevented effective propagation of the virus.

Virus Structure

```
    program V :=

{goto main;
    1234567;

    subroutine infect-executable :=
        {loop:
        file := get-random-executable-file;
        if (first-line-of-file = 1234567)
          then goto loop
          else prepend V to file; }

    subroutine do-damage :=
        {whatever damage is to be done}

    subroutine trigger-pulled :=
        {return true if some condition holds}

main:   main-program :=
        {infect-executable;
        if trigger-pulled then do-damage;
        goto next;}

next:

}
```

A very general depiction of virus structure is shown in Figure. In this case, the virus code, V, is prepended to infected programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program. An infected program begins with the virus code and works as follows. The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the

system. This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

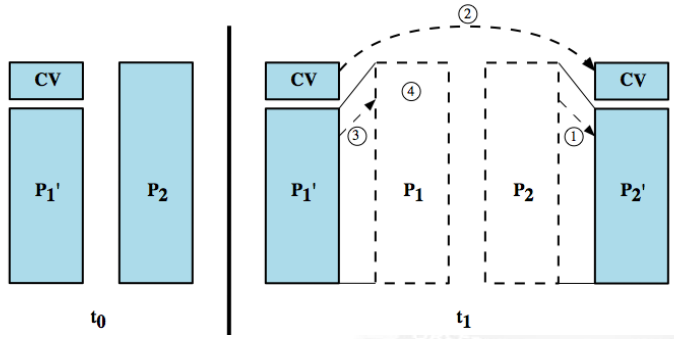
Compression Virus

```
    program CV :=

{goto main;
    01234567;

    subroutine infect-executable :=
          {loop:
               file := get-random-executable-file;
          if (first-line-of-file = 01234567) then goto loop;
       (1)     compress file;
       (2)     prepend CV to file;
       }

main:   main-program :=
          {if ask-permission then infect-executable;
       (3)     uncompress rest-of-file;
       (4)     run uncompressed file;}
       }
```



A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length. The code shown from Figure 21.2 shows in general terms the logic required. The key lines in this virus are numbered, and Figure 21.3 illustrates the operation. In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb. We assume that program P_1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

1. For each uninfected file P_2 that is found, the virus first compresses that file to produce , which is shorter than the original program by the size of the virus.

2. A copy of the virus is prepended to the compressed program.

3. The compressed version of the original infected program, , is uncompressed.

4. The uncompressed original program is executed

Virus Classification

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures have been developed for existing types of viruses, new types have been developed. A virus classification by target includes the following categories:

• Boot sector infector:Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.

• File infector: Infects files that operating system or shell consider to be executable.

• Macro virus: Infects files with macro code that is interpreted by an application.

A virus classification by concealment strategy includes the following categories:

• Encrypted virus: the virus creates a random encryption key, stored with the virus, and encrypts the remainder of the virus. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

• Stealth virus: A form of virus explicitly designed to hide itself from detection by antivirus software. Thus,the entire virus, not just a payload is hidden.

• Polymorphic virus: A virus that mutates with every infection, making detection by the "signature"of the virus impossible.

• Metamorphic virus: As with a polymorphic virus ,a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses may change their behavior as well as their appearance.

Macro Virus

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.

2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.

3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it

up so that the macro is invoked when a function key or special short combination of keys is input. Successive releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus (A/V) product vendors have also developed tools to detect and correct macro viruses. As in other types of viruses, the arms race continues in the field of macro viruses, but they no longer are the predominant virus threat.

E-Mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then the e-mail virus sends itself to everyone on the mailing list in the user's e-mail package, and also does local damage.

At the end of 1999, a more powerful version of the e-mail virus appeared. This newer version can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The virus uses the Visual Basic scripting language supported by the e-mail package.

Thus we see a new generation of malware that arrives via e-mail and uses e-mail software features to replicate itself across the Internet. The virus propagates itself as soon as activated (either by opening an e-mail attachment of by opening the e-mail) to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. This makes it very difficult for antivirus software to respond before much damage is done. Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

Virus Countermeasures

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place. This goal is, in general, impossible to achieve, although prevention can reduce the number of successful viral attacks. The next best approach is to be able to do the following:

• **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.

• **Identification:** Once detection has been achieved, identify the specific virus that has infected a program.

• **Removal:** Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

Anti-Virus Evolution

Advances in virus and antivirus technology go hand in hand. Early viruses were relatively simple code fragments and could be identified and purged with relatively simple antivirus software packages. As the virus arms race has evolved, both viruses and, necessarily, antivirus software have grown more complex and sophisticated. [STEP93] identifies four generations of antivirus software:

A **first-generation** scanner requires a virus signature to identify a virus. The virus may contain "wildcards" but has essentially the same structure and bit pattern in all copies. Such signature-specific scanners are limited to the detection of known viruses.

A **second-generation** scanner uses heuristic rules to search for probable virus infection, e.g to look for fragments of code that are often associated with viruses.. Another second-generation approach is integrity checking, using a hash function rather than a simpler checksum.

Third-generation programs are memory-resident programs that identify a virus by its actions rather than structure in an infected program. These have the advantage that it is not necessary to develop signatures / heuristics, but only to identify the small set of actions indicating an infection is attempted and then intervene.

Fourth-generation products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

Generic Decryption

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight some of the most important.

Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses, while maintaining fast scanning speeds. In order to detect encrypted viruses, executable files are run through a GD scanner:

• **CPU emulator:** A software-based virtual computer that interprets instructions in an executable file rather than executing them on the underlying processor.

• **Virus signature scanner:** scans target code looking for known virus signatures.

• **Emulation control module:** Controls the execution of the target code.

At the start of each simulation, the emulator begins interpreting instructions in the target code, one at a time. Thus, if the code includes a decryption routine that decrypts and hence exposes the virus, that code is interpreted. In effect, the virus does the work for the antivirus program by exposing the virus. Periodically, the control module interrupts interpretation to scan the target code for virus signatures. During interpretation, the target code can cause no damage to the actual personal computer environment, because it is being interpreted in a completely controlled environment. The most difficult design issue with a GD scanner is to determine how

long to run each interpretation. Typically, virus elements are activated soon after a program begins executing, but this need not be the case. The longer the scanner emulates a particular program, the more likely it is to catch any hidden viruses. However, the antivirus program can take up only a limited amount of time and resources before users complain.