

universal agreement on all of the terms and because some of the categories overlap. Table 10.1 is a useful guide to some of the terms in use. 1 I am indebted to Lawrie Brown of the Australian Defence Force Academy, who contributed substantially to this chapter.

Learning Objectives After studying this chapter, you should be able to: ■■ Describe three broad mechanisms malware uses to propagate. ■■ Understand the basic operation of viruses, worms, and trojans. ■■ Describe four broad categories of malware payloads. ■■ Understand the different threats posed by bots, spyware, and rootkits. ■■ Describe some malware countermeasure elements. ■■ Describe three locations for malware detection mechanisms.

10.1 / Types of Malicious Software (Malware) 339

A Broad Classification of Malware Although a range of schemes can be used, one useful approach classifies malware into two broad categories, based first on how it spreads or propagates to reach the desired targets and then on the actions or payloads it performs once a target is reached. Propagation mechanisms include infection of existing executable or interpreted content by viruses that is subsequently spread to other systems; exploit of software vulnerabilities either locally or over a network by worms or drive-by-downloads to allow the malware to replicate; and social engineering attacks that convince users to bypass security mechanisms to install trojans or to respond to phishing attacks.

Name Description

Virus	Malware that, when executed, tries to replicate itself into other executable code; when it succeeds the code is said to be infected. When the infected code is executed, the virus also executes.
Worm	A computer program that can run independently and can propagate a complete working version of itself onto other hosts on a network.
Logic bomb	A program inserted into software by an intruder. A logic bomb lies dormant until a predefined condition is met; the program then triggers an unauthorized act.
Trojan horse	A computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the Trojan horse program.
Backdoor (trapdoor)	Any mechanism that bypasses a normal security check; it may allow unauthorized access to functionality.
Mobile code	Software (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics.
Exploits	Code specific to a single vulnerability or set of vulnerabilities.
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely.
Kit (virus generator)	Set of tools for generating new viruses automatically.
Spammer	programs Used to send large volumes of unwanted e-mail.
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial-of-service (DoS) attack.
Keyloggers	Captures keystrokes on a compromised system.
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access.
Zombie, bot	Program activated on an infected machine that is activated to launch attacks on other machines.
Spyware	Software that collects information from a computer and transmits it to another system.
Adware	Advertising that is integrated into software. It can result in pop-up ads or redirection of a browser to a commercial site.

Table 10.1 Terminology for Malicious Software 340

chapter 10 / Malicious Software Earlier approaches to malware classification distinguished between those that need a host program, being parasitic code such as viruses, and those that are independent, self-contained programs run on the system such as worms, trojans, and bots. Another distinction used was between malware that does not replicate, such as trojans and spam e-mail, and malware that does, including viruses and worms. Payload actions performed by malware once it reaches a target system can

include corruption of system or data files; theft of service in order to make the system a zombie agent of attack as part of a botnet; theft of information from the system, especially of logins, passwords, or other personal details by keylogging or spyware programs; and stealthing where the malware hides its presence on the system from attempts to detect and block it. While early malware tended to use a single means of propagation to deliver a single payload, as it evolved we see a growth of blended malware that incorporates a range of both propagation mechanisms and payloads that increase its ability to spread, hide, and perform a range of actions on targets. A blended attack uses multiple methods of infection or propagation, to maximize the speed of contagion and the severity of the attack. Some malware even support an update mechanism that allows it to change the range of propagation and payload mechanisms utilized once it is deployed. In the following sections, we survey these various categories of malware, and then follow with a discussion of appropriate countermeasures.

Attack Kits Initially, the development and deployment of malware required considerable technical skill by software authors. This changed with the development of virus-creation toolkits in the early 1990s, and then later of more general attack kits in the 2000s, that greatly assisted in the development and deployment of malware [FOSS10]. These toolkits, often known as crimeware, now include a variety of propagation mechanisms and payload modules that even novices can combine, select, and deploy. They can also easily be customized with the latest discovered vulnerabilities in order to exploit the window of opportunity between the publication of a weakness and the widespread deployment of patches to close it. These kits greatly enlarged the population of attackers able to deploy malware. Although the malware created with such toolkits tends to be less sophisticated than that designed from scratch, the sheer number of new variants that can be generated by attackers using these toolkits creates a significant problem for those defending systems against them. The Zeus crimeware toolkit is a prominent, recent example of such an attack kit, which was used to generate a wide range of very effective, stealthed malware that facilitates a range of criminal activities, in particular capturing and exploiting banking credentials [BINS10]. Other widely used toolkits include Blackhole, Sakura, and Phoenix [SYMA13].

Attack Sources Another significant malware development over the last couple of decades is the change from attackers being individuals, often motivated to demonstrate their technical competence to their peers, to more organized and dangerous attack sources. These include politically motivated attackers, criminals, and organized

10.2 / Advanced Persistent Threat

341 crime; organizations that sell their services to companies and nations; and national government agencies. This has significantly changed the resources available and motivation behind the rise of malware, and indeed has led to development of a large underground economy involving the sale of attack kits, access to compromised hosts, and to stolen information.

10.2 Advanced Persistent Threat

Advanced Persistent Threats (APTs) have risen to prominence in recent years. These are not a new type of malware, but rather the well-resourced, persistent application of a wide variety of intrusion technologies and malware to selected targets, usually business or political. APTs are typically attributed to state-sponsored organizations, with some attacks likely from criminal enterprises as well. We discuss these categories of intruders further in Chapter 11. APTs differ from other types of attack by their careful target selection, and persistent, often stealthy, intrusion efforts over extended periods. A number of high profile attacks, including Aurora, RSA, APT1, and Stuxnet, are often cited as examples. They are named as a result of these characteristics:

- **Advanced:** Used by the attackers of a wide variety of intrusion technologies and malware, including the development of custom malware if

required. The individual components may not necessarily be technically advanced, but are carefully selected to suit the chosen target. ■■ Persistent: Determined application of the attacks over an extended period against the chosen target in order to maximize the chance of success. A variety of attacks may be progressively, and often stealthily, applied until the target is compromised. ■■ Threats: Threats to the selected targets as a result of the organized, capable, and well-funded attackers intent to compromise the specifically chosen targets. The active involvement of people in the process greatly raises the threat level from that due to automated attacks tools and the likelihood of successful attack. The aim of these attacks varies from theft of intellectual property or security and infrastructure related data, to the physical disruption of infrastructure. Techniques used include social engineering, spear-phishing e-mails, drive-bydownloads from selected compromised Web sites likely to be visited by personnel in the target organization, to infect the target with sophisticated malware with multiple propagation mechanisms and payloads. Once they have gained initial access to systems in the target organization, a further range of attack tools are used to maintain and extend their access. As a result, these attacks are much harder to defend against due to this specific targeting and persistence. It requires a combination of technical countermeasures, such as we discuss later in this chapter, as well as awareness training to assist personnel to resist such attacks. Even with current best-practice countermeasures, the use of zero-day exploits and new attack approaches means that some of these 342 chapter 10 / Malicious Software attacks are likely to succeed [SYMA13, MAND13]. Thus multiple layers of defense are needed, with mechanisms to detect, respond and mitigate such attacks. These may include monitoring for malware command and control traffic, and detection of exfiltration traffic.

10.3 Propagation—Infected Content—Viruses

The first category of malware propagation concerns parasitic software fragments that attach themselves to some existing executable content. The fragment may be machine code that infects some existing application, utility, or system program, or even the code used to boot a computer system. More recently, the fragment has been some form of scripting code, typically used to support active content within data files such as Microsoft Word documents, Excel spreadsheets, or Adobe PDF documents.

The Nature of Viruses

A computer virus is a piece of software that can “infect” other programs, or indeed any type of executable content, by modifying them. The modification includes injecting the original code with a routine to make copies of the virus code, which can then go on to infect other content. A computer virus carries in its instructional code the recipe for making perfect copies of itself. The typical virus becomes embedded in a program, or carrier of executable content, on a computer. Then, whenever the infected computer comes into contact with an uninfected piece of code, a fresh copy of the virus passes into the new location. Thus, the infection can spread from computer to computer, aided by unsuspecting users, who exchange these programs or carrier files on disk or USB stick, or who send them to one another over a network. In a network environment, the ability to access documents, applications, and system services on other computers provides a perfect culture for the spread of such viral code. A virus that attaches to an executable program can do anything that the program is permitted to do. It executes secretly when the host program is run. Once the virus code is executing, it can perform any function, such as erasing files and programs, that is allowed by the privileges of the current user. One reason viruses dominated the malware scene in earlier years was the lack of user authentication and access controls on personal computer systems at that time. This enabled a virus to infect any executable content on the system. The significant quantity of programs shared on floppy disk also enabled its

easy, if somewhat slow, spread. The inclusion of tighter access controls on modern operating systems significantly hinders the ease of infection of such traditional, machine-executable code, viruses. This resulted in the development of macro viruses that exploit the active content supported by some document types, such as Microsoft Word or Excel files, or Adobe PDF documents. Such documents are easily modified and shared by users as part of their normal system use and are not protected by the same access controls as programs. Currently, a viral mode of infection is typically one of several propagation mechanisms used by contemporary malware, which may also include worm and Trojan capabilities.

10.3 / Propagation—Infected Content—Viruses 343

A computer virus, and more generally many contemporary types of malware, includes one or more variants of each of these components:

- **Infection mechanism:** The means by which a virus spreads or propagates, enabling it to replicate. The mechanism is also referred to as the infection vector.
- **Trigger:** The event or condition that determines when the payload is activated or delivered, sometimes known as a logic bomb.
- **Payload:** What the virus does, besides spreading. The payload may involve damage or benign but noticeable activity. 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 a copy of itself into other programs or into certain system areas on the disk. The copy may not be identical to the propagating version; viruses often morph to evade detection. 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. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files. Most viruses that infect executable program files 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. Macro viruses, though, target specific document types, which are often supported on a variety of systems.

Executable Virus Structure Traditional machine-executable virus code can be prepended or postpended to some executable program, or it can be embedded into the program 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. A very general depiction of virus structure is shown in Figure 10.1a. 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. The infected program begins with the virus code and works as follows. The first line of code labels the program, which then begins execution with the main action block of the virus. 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 may first seek out uninfected executable files and infect them. Next, the virus may execute its payload if the required trigger 344 chapter 10 / Malicious Software conditions, if any, are met. 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 an uninfected program. 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. Figure 10.1b shows in general terms the logic required. The key lines in this virus are labeled with times, and Figure 10.2 illustrates the operation. We begin at time t_0 , with program $P_1 =$, which is program P_1 infected with virus CV, and a clean program P_2 , which is not infected with CV. When P_1 is invoked, control passes to its virus, which performs the following steps: t_1 : For each uninfected file P_2 that is found, the virus first compresses that file to produce $P_2 =$, which is shorter than the original program by the size of the virus CV. t_2 : A copy of CV is prepended to the compressed program. t_3 : The compressed version of the original infected program, $P_1 =$ is uncompressed. t_4 : The uncompressed original program P_1 is executed. In this example, the virus does nothing other than propagate. As previously mentioned, the virus may also include one or more payloads. Once a virus has gained entry to a system by infecting a single program, it is in a position to potentially infect some or all other executable files on that system

```

program V 1234567;
procedure attach-to-program;
begin
  repeat
    file := get-random-program;
  until first-program-line # 1234567;
  prepend V to file;
end;
procedure execute-payload;
begin (* perform payload actions *)
end;
procedure trigger-condition;
begin (* return true if trigger condition is true *)
end;
begin (* main action block *)
  attach-to-program;
  if trigger-condition then execute-payload;
  goto main;
end;
program CV 1234567;
procedure attach-to-program;
begin
  repeat
    file := get-random-program;
  until first-program-line # 1234567;
  compress file; (* t1 *)
  prepend CV to file; (* t2 *)
end;
procedure (* main action block *)
if ask-permission then attach-to-program;
uncompress rest of this file into tempfile; (* t3 *)
execute tempfile; (* t4 *)
end;

```

(a) A simple virus (b) A compression virus

Figure 10.1 Example Virus Logic

10.3 / Propagation—Infected Content—Viruses

345

when the infected program executes, depending on the access permissions the infected program has. Thus, viral infection can be completely prevented by blocking 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. Many forms of infection can also be blocked by denying normal users the right to modify programs on the system.

Viruses Classification

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures are developed for existing types of viruses, newer types are developed. There is no simple or universally agreed-upon classification scheme for viruses. In this section, we follow [AYCO06] and classify viruses along two orthogonal axes: the type of target the virus tries to infect and the method the virus uses to conceal itself from detection by users and antivirus software. Figure 10.2 A

Compression Virus

t_0 : P_1 is infected version of P_1 ; P_2 is clean CV
 t_1 : P_2 is compressed into P_2
 t_2 : CV attaches itself to P_2
 t_3 : P_1 is decompressed into the original program P_1
 CV $P_1 \oplus P_1 \oplus P_1 \oplus P_2 \oplus P_2$

346 chapter 10 / Malicious Software

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 the operating system or shell consider to be executable.
- Macro virus: Infects files with macro or scripting code that is interpreted by an application.
- Multipartite virus: Infects files 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 virus classification by

concealment strategy includes the following categories: ■■ Encrypted virus: A typical approach is as follows. A portion of the virus creates a random encryption key and encrypts the remainder of the virus. The key is stored with 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. Because the bulk of the virus is encrypted with a different key for each instance, there is no constant bit pattern to observe. ■■ 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. It may use both code mutation, for example, compression, and rootkit techniques to achieve this. ■■ Polymorphic virus: A form of virus that creates copies during replication that are functionally equivalent but have distinctly different bit patterns, in order to defeat programs that scan for viruses. In this case, the “signature” of the virus will vary with each copy. To achieve this variation, the virus may randomly insert superfluous instructions or interchange the order of independent instructions. A more effective approach is to use encryption. The strategy of the encryption virus is followed. The portion of the virus that is responsible for generating keys and performing encryption/decryption is referred to as the mutation engine. The mutation engine itself is altered with each use. ■■ 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 and Scripting Viruses Macro viruses infect scripting code used to support active content in a variety of user document types. Macro viruses are particularly threatening for a number of reasons: 1. A macro virus is platform independent. Many macro viruses infect active content in commonly used applications, such as macros in Microsoft Word documents or other Microsoft Office documents, or scripting code in Adobe PDF 10.4 / Propagation—Vulnerability Exploit—Worms 347 documents. Any hardware platform and operating system that supports these applications 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 documents rather than programs. 3. Macro viruses are easily spread, as the documents they exploit are shared in normal use. A very common method is by electronic mail. 4. Because macro viruses infect user documents rather than system programs, traditional file system access controls are of limited use in preventing their spread, since users are expected to modify them. Macro viruses take advantage of support for active content using a scripting or macro language, embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. They are also used to support dynamic content, form validation, and other useful tasks associated with these documents. Successive releases of MS Office products 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 product vendors have also developed tools to detect and remove 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. Another possible host for macro virus-style malware is in Adobe’s PDF documents. These can support a range of embedded components, including Javascript and other types of scripting code. Although recent PDF viewers include measures to warn users when such code is run, the message the user is shown can be manipulated to trick them into permitting its execution. If this occurs, the code could potentially act as a virus to infect

other PDF documents the user can access on his or her system. Alternatively, it can install a Trojan, or act as a worm, as we discuss later.

10.4 Propagation—Vulnerability Exploit—Worms

A worm is a program that actively seeks out more machines to infect, and then each infected machine serves as an automated launching pad for attacks on other machines. Worm programs exploit software vulnerabilities in client or server programs to gain access to each new system. They can use network connections to spread from system to system. They can also spread through shared media, such as USB drives or optical data disks. E-mail worms spread in macro or script code included in documents attached to e-mail or to instant messenger file transfers. Upon activation, the worm may replicate and propagate again. In addition to propagation, the worm usually carries some form of payload, such as those we discuss later.

348 chapter 10 / Malicious Software

To replicate itself, a worm uses some means to access remote systems. These include the following, most of which are still seen in active use [SYMA13]:

- Electronic mail or instant messenger facility: A worm e-mails a copy of itself to other systems or sends itself as an attachment via an instant message service, so that its code is run when the e-mail or attachment is received or viewed.
- File sharing: A worm either creates a copy of itself or infects other suitable files as a virus on removable media such as a USB drive; it then executes when the drive is connected to another system using the autorun mechanism by exploiting some software vulnerability or when a user opens the infected file on the target system.
- Remote execution capability: A worm executes a copy of itself on another system, either by using an explicit remote execution facility or by exploiting a program flaw in a network service to subvert its operations.
- Remote file access or transfer capability: A worm uses a remote file access or transfer service to another system to copy itself from one system to the other, where users on that system may then execute it.
- Remote login capability: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other, where it then executes. The new copy of the worm program is then run on the remote system where, in addition to any payload functions that it performs on that system, it continues to propagate. A worm typically uses the same phases as a computer virus: dormant, propagation, triggering, and execution. The propagation phase generally performs the following functions:

- Search for appropriate access mechanisms to other systems to infect by examining host tables, address books, buddy lists, trusted peers, and other similar repositories of remote system access details; by scanning possible target host addresses; or by searching for suitable removable media devices to use.
- Use the access mechanisms found to transfer a copy of itself to the remote system and cause the copy to be run. The worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it can also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator. More recent worms can even inject their code into existing processes on the system and run using additional threads in that process, to further disguise their presence.

Target Discovery

The first function in the propagation phase for a network worm is for it to search for other systems to infect, a process known as scanning or fingerprinting. For such worms, which exploit software vulnerabilities in remotely accessible network

10.4 / Propagation—Vulnerability Exploit—Worms

349 services, it must identify potential systems running the vulnerable service, and then infect them. Then, typically, the worm code now installed on the infected machines repeats the same scanning process, until a large distributed network of infected machines is created. [MIRK04] lists the following types of network address scanning strategies that such a worm can use:

- Random: Each

compromised host probes random addresses in the IP address space, using a different seed. This technique produces a high volume of Internet traffic, which may cause generalized disruption even before the actual attack is launched. ■■ Hit list: The attacker first compiles a long list of potential vulnerable machines. This can be a slow process done over a long period to avoid detection that an attack is underway. Once the list is compiled, the attacker begins infecting machines on the list. Each infected machine is provided with a portion of the list to scan. This strategy results in a very short scanning period, which may make it difficult to detect that infection is taking place. ■■ Topological: This method uses information contained on an infected victim machine to find more hosts to scan. ■■ Local subnet: If a host is infected behind a firewall, that host then looks for targets in its own local network. The host uses the subnet address structure to find other hosts that would otherwise be protected by the firewall.

Worm Propagation Model A well-designed worm can spread rapidly and infect massive numbers of hosts. It is useful to have a general model for the rate of worm propagation. Computer viruses and worms exhibit similar self-replication and propagation behavior to biological viruses. Thus we can look to classic epidemic models for understanding computer virus and worm propagation behavior. A simplified, classic epidemic model can be expressed as follows: $\frac{dI(t)}{dt} = bI(t)S(t)$ where $I(t)$ = number of individuals infected as of time t $S(t)$ = number of susceptible individuals (susceptible to infection but not yet infected) at time t b = infection rate N = size of the population, $N = I(t) + S(t)$ Figure 10.3 shows the dynamics of worm propagation using this model. Propagation proceeds through three phases. In the initial phase, the number of hosts increases exponentially. To see that this is so, consider a simplified case in which a worm is launched from a single host and infects two nearby hosts. Each of these 350 chapter 10 / Malicious Software hosts infects two more hosts, and so on. This results in exponential growth. After a time, infecting hosts waste some time attacking already-infected hosts, which reduces the rate of infection. During this middle phase, growth is approximately linear, but the rate of infection is rapid. When most vulnerable computers have been infected, the attack enters a slow finish phase as the worm seeks out those remaining hosts that are difficult to identify. Clearly, the objective in countering a worm is to catch the worm in its slow start phase, at a time when few hosts have been infected. Zou and others [ZOU05] describe a model for worm propagation based on an analysis of network worm attacks at that time. The speed of propagation and the total number of hosts infected depend on a number of factors, including the mode of propagation, the vulnerability or vulnerabilities exploited, and the degree of similarity to preceding attacks. For the latter factor, an attack that is a variation of a recent previous attack may be countered more effectively than a more novel attack. Zou's model agrees closely with Figure 10.3.

The Morris Worm The earliest significant worm infection was released onto the Internet by Robert Morris in 1988 [ORMA03]. The Morris worm was designed to spread on UNIX systems and used a number of different techniques for propagation. When a copy began execution, its first task was to discover other hosts known to this host that would allow entry from this host. The worm performed this task by examining a variety of lists and tables, including system tables that declared which other machines were trusted by this host, users' mail forwarding files, tables by which users gave themselves permission for access to remote accounts, and from a program that

Figure 10.3 Worm Propagation Model

Fraction of hosts infected	Fraction of hosts not infected	Time
0.2	0.8	0.4
0.6	0.4	0.6
0.8	0.2	1.0

Slow start phase Fast spread phase Slow finish phase

10.4 / Propagation—Vulnerability Exploit—Worms 351 reported the status of network connections. For each discovered host, the worm tried a number of methods for gaining access: 1. It attempted to log on to a remote

host as a legitimate user. In this method, the worm first attempted to crack the local password file and then used the discovered passwords and corresponding user IDs. The assumption was that many users would use the same password on different systems. To obtain the passwords, the worm ran a password-cracking program that tried a. Each user's account name and simple permutations of it b. A list of 432 built-in passwords that Morris thought to be likely candidates² c. All the words in the local system dictionary 4. It exploited a bug in the UNIX finger protocol, which reports the whereabouts of a remote user. 5. It exploited a trapdoor in the debug option of the remote process that receives and sends mail. If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter. It then sent this interpreter a short bootstrap program, issued a command to execute that program, and then logged off. The bootstrap program then called back the parent program and downloaded the remainder of the worm. The new worm was then executed.

State of Worm Technology The state of the art in worm technology includes the following:

- **Multiplatform:** Newer worms are not limited to Windows machines but can attack a variety of platforms, especially the popular varieties of UNIX, or exploit macro or scripting languages supported in popular document types.
- **Multiexploit:** New worms penetrate systems in a variety of ways, using exploits against Web servers, browsers, e-mail, file sharing, and other network-based applications, or via shared media.
- **Ultrafast spreading:** Exploit various techniques to optimize the rate of spread of a worm to maximize its likelihood of locating as many vulnerable machines as possible in a short time period.
- **Polymorphic:** To evade detection, skip past filters, and foil real-time analysis, worms adopt the virus polymorphic technique. Each copy of the worm has new code generated on the fly using functionally equivalent instructions and encryption techniques.
- **Metamorphic:** In addition to changing their appearance, metamorphic worms have a repertoire of behavior patterns that are unleashed at different stages of propagation.
- **Transport vehicles:** Because worms can rapidly compromise a large number of systems, they are ideal for spreading a wide variety of malicious payloads.

² The complete list is provided at this book's Premium Content Web site.

352 chapter 10 / Malicious Software such as distributed denial-of-service bots, rootkits, spam e-mail generators, and spyware.

- **Zero-day exploit:** To achieve maximum surprise and distribution, a worm should exploit an unknown vulnerability that is only discovered by the general network community when the worm is launched.

Mobile Code SP 800-28 (Guidelines on Active Content and Mobile Code, March 2008) defines mobile code as 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. 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.

Client-Side Vulnerabilities and Drive-by-Downloads Another approach to exploiting software vulnerabilities involves the exploit of bugs in user applications to install malware. One common approach to this exploits browser vulnerabilities so that when the user views a Web page controlled by the attacker, it contains code that exploits the browser bug to download and install malware on the

system without the user's knowledge or consent. This is known as a drive-by-download and is a common exploit in recent attack kits. In most cases this malware does not actively propagate as a worm does, but rather waits for unsuspecting users to visit the malicious Web page in order to spread to their systems. In general, drive-by-download attacks are aimed at anyone who visits a compromised site and is vulnerable to the exploits used. Watering-hole attacks are a variant of this used in highly targeted attacks. The attacker researches their intended victims to identify Web sites they are likely to visit and then scans these sites to identify those with vulnerabilities that allow their compromise with a drive-by-download attack. They then wait for one of their intended victims to visit one of the compromised sites. Their attack code may even be written so that it will only infect systems belonging to the target organization and take no action for other visitors to the site. This greatly increases the likelihood of the site compromise remaining undetected. Malvertising is another technique used to place malware on Web sites without actually compromising them. The attacker pays for advertisements that are highly likely to be placed on their intended target Web sites, and which incorporate malware in them. Using these malicious ads, attackers can infect visitors to sites displaying them. Again, the malware code may be dynamically generated to either reduce the chance of detection or only infect specific systems.

10.5 / Propagation—Social Engineering—Spam e-mail, Trojans

353 Related variants can exploit bugs in common e-mail clients, such as the Klez mass-mailing worm seen in October 2001, which targeted a bug in the HTML handling in Microsoft's Outlook and Outlook Express programs to automatically run itself. Or, such malware may target common PDF viewers to also download and install malware without the user's consent, when they view a malicious PDF document [STEV11]. Such documents may be spread by spam e-mail or be part of a targeted phishing attack, as we discuss next.

Clickjacking

Clickjacking, also known as a user-interface (UI) redress attack, is a vulnerability used by an attacker to collect an infected user's clicks. The attacker can force the user to do a variety of things from adjusting the user's computer settings to unwittingly sending the user to Web sites that might have malicious code. Also, by taking advantage of Adobe Flash or JavaScript, an attacker could even place a button under or over a legitimate button, making it difficult for users to detect. A typical attack uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is hijacking clicks meant for one page and routing them to another page, most likely owned by another application, domain, or both. Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their e-mail or bank account but are instead typing into an invisible frame controlled by the attacker. There is a wide variety of techniques for accomplishing a clickjacking attack, and new techniques are developed as defenses to older techniques are put in place. [NIEM11] and [STON10] are useful discussions.

10.5 Propagation—Social Engineering—Spam e-mail, Trojans

The final category of malware propagation we consider involves social engineering, "tricking" users to assist in the compromise of their own systems or personal information. This can occur when a user views and responds to some SPAM e-mail or permits the installation and execution of some Trojan horse program or scripting code.

Spam (Unsolicited Bulk) E-Mail

Unsolicited bulk e-mail, commonly known as spam, imposes significant costs on both the network infrastructure needed to relay this traffic and on users who need to filter their legitimate e-mails out of this flood. In response to the explosive growth in spam, there has been the equally rapid growth of the antispam

industry, which provides products to detect and filter spam e-mails. This has led to an arms race between the spammers devising techniques to sneak their content through and the defenders taking efforts to block them. In recent years, the volume of spam e-mail has started to decline. One reason is the rapid growth of attacks, including spam, 354 chapter 10 / Malicious Software spread via social media networks. This reflects the rapid growth in use of these networks, which form a new arena for attackers to exploit [SYMA13]. While some spam is sent from legitimate mail servers, most recent spam is sent by botnets using compromised user systems, as we discuss in Section 10.6. A significant portion of spam e-mail content is just advertising, trying to convince the recipient to purchase some product online, or used in scams, such as stock scams or money mule job ads. But spam is also a significant carrier of malware. The e-mail may have an attached document, which, if opened, may exploit a software vulnerability to install malware on the user's system, as we discussed in the previous section. Or, it may have an attached Trojan horse program or scripting code that, if run, also installs malware on the user's system. Some trojans avoid the need for user agreement by exploiting a software vulnerability in order to install themselves, as we discuss next. Finally the spam may be used in a phishing attack, typically directing the user either to a fake Web site that mirrors some legitimate service, such as an online banking site, where it attempts to capture the user's login and password details, or to complete some form with sufficient personal details to allow the attacker to impersonate the user in an identity theft. All of these uses make spam e-mails a significant security concern. However, in many cases it requires the user's active choice to view the e-mail and any attached document or to permit the installation of some program, in order for the compromise to occur.

Trojan Horses A Trojan horse is a useful, or apparently useful, program or utility containing hidden code that, when invoked, performs some unwanted or harmful function. Trojan horse programs can be used to accomplish functions indirectly that the attacker could not accomplish directly. For example, to gain access to sensitive, personal information stored in the files of a user, an attacker could create a Trojan horse program that, when executed, scans the user's files for the desired sensitive information and sends a copy of it to the attacker via a Web form or e-mail or text message. The author could then entice users to run the program by incorporating it into a game or useful utility program and making it available via a known software distribution site or app store. This approach has been used recently with utilities that "claim" to be the latest antivirus scanner, or security update, for systems, but which are actually malicious trojans, often carrying payloads such as spyware that searches for banking credentials. Hence, users need to take precautions to validate the source of any software they install. Trojan horses fit into one of three models: ■■ Continuing to perform the function of the original program and additionally performing a separate malicious activity ■■ Continuing to perform the function of the original program but modifying the function to perform malicious activity (e.g., a Trojan horse version of a login program that collects passwords) or to disguise other malicious activity (e.g., a Trojan horse version of a process-listing program that does not display certain processes that are malicious) ■■ Performing a malicious function that completely replaces the function of the original program

10.6 / Payload—System Corruption 355 Some trojans avoid the requirement for user assistance by exploiting some software vulnerability to enable their automatic installation and execution. In this they share some features of a worm, but unlike it, they do not replicate. A prominent example of such an attack was the Hydraq Trojan used in Operation Aurora in 2009 and early 2010. This exploited a vulnerability in Internet Explorer to install itself and targeted several high-profile companies [SYMA13]. It was typically distributed either by

spam e-mail or via a compromised Web site using a “drive-by-download.”

10.6 Payload—System Corruption

Once malware is active on the target system, the next concern is what actions it will take on this system, that is, what payload does it carry. Some malware has a nonexistent or nonfunctional payload. Its only purpose, either deliberate or due to accidental early release, is to spread. More commonly, it carries one or more payloads that perform covert actions for the attacker. An early payload seen in a number of viruses and worms resulted in data destruction on the infected system when certain trigger conditions were met [WEAV03]. A related payload is one that displays unwanted messages or content on the user’s system when triggered. More seriously, another variant attempts to inflict real-world damage on the system. All of these actions target the integrity of the computer system’s software or hardware, or of the user’s data. These changes may not occur immediately, but only when specific trigger conditions are met that satisfy their logic-bomb code. As an alternative to just destroying data, some malware encrypts the user’s data and demands payment in order to access the key needed to recover this information. This is sometimes known as ransomware. The PC Cyborg Trojan seen in 1989 was an early example of this. However, around mid-2006 a number of worms and trojans, such as the Gpcode Trojan, that used public-key cryptography with increasingly larger key sizes to encrypt data. The user needed to pay a ransom or to make a purchase from certain sites, in order to receive the key to decrypt this data. While earlier instances used weaker cryptography that could be cracked without paying the ransom, the later versions using public-key cryptography with large key sizes could not be broken this way.

Real-World Damage

A further variant of system corruption payloads aims to cause damage to physical equipment. The infected system is clearly the device most easily targeted. The Chernobyl virus not only corrupts data, it attempts to rewrite the BIOS code used to initially boot the computer. If it is successful, the boot process fails, and the system is unusable until the BIOS chip is either reprogrammed or replaced. The Stuxnet worm targets some specific industrial control system software as its key payload [CHEN11]. If control systems using certain Siemens industrial control software with a specific configuration of devices are infected, then the worm replaces the original control code with code that deliberately drives the controlled equipment outside its normal operating range, resulting in the failure of the attached equipment. The centrifuges used in the Iranian uranium enrichment program were strongly suspected as the target, with reports of much higher than normal failure rates observed in them over the period when this worm was active. As noted in our 356 chapter 10 / Malicious Software earlier discussion, this has raised concerns over the use of sophisticated targeted malware for industrial sabotage.

Logic Bomb

A key component of data-corrupting malware is the logic bomb. The logic bomb is code embedded in the malware 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 or devices on the system, a particular day of the week or date, a particular version or configuration of some software, 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. All of the examples we describe in this section include such code.

10.7 Payload—Attack Agent—Zombie, Bots

The next category of payload we discuss is where the malware subverts the computational and network resources of the infected system for use by the attacker. Such a system is known as a bot (robot), zombie, or drone, and secretly takes over another Internet-attached computer and then uses that computer to launch or manage attacks that are difficult to trace to the bot’s creator. The bot is typically planted on hundreds or thousands of computers belonging to

unsuspecting third parties. The collection of bots often is capable of acting in a coordinated manner; such a collection is referred to as a botnet. This type of payload attacks the integrity and availability of the infected system. Uses of Bots [HONE05] lists the following uses of bots: ■■ Distributed denial-of-service (DDoS) attacks: A DDoS attack is an attack on a computer system or network that causes a loss of service to users. We examine DDoS attacks in Section 10.10. ■■ Spamming: With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of bulk e-mail (spam). ■■ Sniffing traffic: Bots can also use a packet sniffer to watch for interesting cleartext data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords. ■■ Keylogging: If the compromised machine uses encrypted communication channels (e.g., HTTPS or POP3S), then just sniffing the network packets on the victim's computer is useless because the appropriate key to decrypt the packets is missing. But by using a keylogger, which captures keystrokes on the infected machine, an attacker can retrieve sensitive information. ■■ Spreading new malware: Botnets are used to spread new bots. This is very easy since all bots implement mechanisms to download and execute a file via HTTP or FTP. A botnet with 10,000 hosts that acts as the start base for a worm or mail virus allows very fast spreading and thus causes more harm. 10.8 / Payload—Information Theft—Keyloggers, Phishing, Spyware 357 ■■ Installing advertisement add-ons and browser helper objects (BHOs): Botnets can also be used to gain financial advantages. This works by setting up a fake Web site with some advertisements: The operator of this Web site negotiates a deal with some hosting companies that pay for clicks on ads. With the help of a botnet, these clicks can be “automated” so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start-page of a compromised machine so that the “clicks” are executed each time the victim uses the browser. ■■ Attacking IRC chat networks: Botnets are also used for attacks against Internet Relay Chat (IRC) networks. Popular among attackers is the so-called clone attack: In this kind of attack, the controller orders each bot to connect a large number of clones to the victim IRC network. The victim is flooded by service requests from thousands of bots or thousands of channel-joins by these cloned bots. In this way, the victim IRC network is brought down, similar to a DDoS attack. ■■ Manipulating online polls/games: Online polls/games are getting more and more attention and it is rather easy to manipulate them with botnets. Since every bot has a distinct IP address, every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way. Remote Control Facility The remote control facility is what distinguishes a bot from a worm. A worm propagates itself and activates itself, whereas a bot is controlled from some central facility, at least initially. A typical means of implementing the remote control facility is on an IRC server. All bots join a specific channel on this server and treat incoming messages as commands. More recent botnets tend to avoid IRC mechanisms and use covert communication channels via protocols such as HTTP. Distributed control mechanisms, using peer-to-peer protocols, are also used, to avoid a single point of failure. Once a communications path is established between a control module and the bots, the control module can activate the bots. In its simplest form, the control module simply issues command to the bot that causes the bot to execute routines that are already implemented in the bot. For greater flexibility, the control module can issue update commands that instruct the bots to download a file from some Internet location and execute it. The bot in this latter case becomes a more general-purpose tool that can be used for multiple attacks. 10.8 Payload—Information Theft—Keyloggers, Phishing, Spyware We now consider payloads

where the malware gathers data stored on the infected system for use by the attacker. A common target is the user's login and password credentials to banking, gaming, and related sites, which the attacker then uses to impersonate the user to access these sites for gain. Less commonly, the payload may target documents or system configuration details for the purpose of reconnaissance or espionage. These attacks target the confidentiality of this information. 358 chapter 10 / Malicious Software Credential Theft, Keyloggers, and Spyware Typically, users send their login and password credentials to banking, gaming, and related sites over encrypted communication channels (e.g., HTTPS or POP3S), which protect them from capture by monitoring network packets. To bypass this, an attacker can install a keylogger, which captures keystrokes on the infected machine to allow an attacker to monitor this sensitive information. Since this would result in the attacker receiving a copy of all text entered on the compromised machine, keyloggers typically implement some form of filtering mechanism that only returns information close to desired keywords (e.g., "login" or "password" or "paypal.com"). In response to the use of keyloggers, some banking and other sites switched to using a graphical applet to enter critical information, such as passwords. Since these do not use text entered via the keyboard, traditional keyloggers do not capture this information. In response, attackers developed more general spyware payloads, which subvert the compromised machine to allow monitoring of a wide range of activity on the system. This may include monitoring the history and content of browsing activity, redirecting certain Web page requests to fake sites controlled by the attacker, dynamically modifying data exchanged between the browser and certain Web sites of interest. All of which can result in significant compromise of the user's personal information. Phishing and Identity Theft Another approach used to capture a user's login and password credentials is to include a URL in a spam e-mail that links to a fake Web site controlled by the attacker, but which mimics the login page of some banking, gaming, or similar site. This is normally included in some message suggesting that urgent action is required by the user to authenticate his or her account, to prevent it being locked. If the user is careless, and doesn't realize that he or she is being conned, then following the link and supplying the requested details will certainly result in the attackers exploiting the user's account using the captured credentials. More generally, such a spam e-mail may direct a user to a fake Web site controlled by the attacker or to complete some enclosed form and return to an e-mail accessible to the attacker, which is used to gather a range of private, personal information on the user. Given sufficient details, the attacker can then "assume" the user's identity for the purpose of obtaining credit or sensitive access to other resources. This is known as a phishing attack, which exploits social engineering to leverage user's trust by masquerading as communications from a trusted source [GOLD10]. Such general spam e-mails are typically widely distributed to very large numbers of users, often via a botnet. While the content will not match appropriate trusted sources for a significant fraction of the recipients, the attackers rely on it reaching sufficient users of the named trusted source, a gullible portion of whom will respond, for it to be profitable. A more dangerous variant of this is the spear-phishing attack. This again is an e-mail claiming to be from a trusted source. However, the recipients are carefully researched by the attacker, and each e-mail is carefully crafted to suit its recipient specifically, often quoting a range of information to convince him or her of its 10.9 / Payload—Stealth—Backdoors, Rootkits 359 authenticity. This greatly increases the likelihood of the recipient responding as desired by the attacker. Reconnaissance and Espionage Credential theft and identity theft are special cases of a more general reconnaissance payload, which aims to obtain certain

types of desired information and return this to the attacker. These special cases are certainly the most common; however other targets are known. Operation Aurora in 2009 used a Trojan to gain access to and potentially modify source code repositories at a range of high-tech, security, and defense contractor companies [SYMA13]. The Stuxnet worm discovered in 2010 included capture of hardware and software configuration details in order to determine whether it had compromised the specific desired target systems. Early versions of this worm returned this same information, which was then used to develop the attacks deployed in later versions [CHEN11].

10.9 Payload—Stealth—Backdoors, Rootkits

The final category of payload we discuss concerns techniques used by malware to hide its presence on the infected system and to provide covert access to that system. This type of payload also attacks the integrity of the infected system.

Backdoor

A backdoor, also known as a trapdoor, is a secret entry point into a program that allows someone who is aware of the backdoor to gain access without going through the usual security access procedures. The backdoor is code that recognizes some special sequence of input or is triggered by being run from a certain user ID or by an unlikely sequence of events. A backdoor is usually implemented as a network service listening on some nonstandard port that the attacker can connect to and issue commands through to be run on the compromised system. It is difficult to implement operating system controls for backdoors in applications. Security measures must focus on the program development and software update activities, and on programs that wish to offer a network service.

Rootkit

A rootkit is a set of programs installed on a system to maintain covert access to that system with administrator (or root)³ privileges, while hiding evidence of its presence to the greatest extent possible. This provides access to all the functions and services of the operating system. The rootkit alters the host's standard functionality in a malicious and stealthy way. With root access, an attacker has complete control of the system and can add or change programs and files, monitor processes, send and receive network traffic, and get backdoor access on demand. ³ On UNIX systems, the administrator, or superuser, account is called root; hence the term root access.

360 chapter 10 / Malicious Software

A rootkit can make many changes to a system to hide its existence, making it difficult for the user to determine that the rootkit is present and to identify what changes have been made. In essence, a rootkit hides by subverting the mechanisms that monitor and report on the processes, files, and registries on a computer. A rootkit can be classified using the following characteristics:

- **Persistent:** Activates each time the system boots. The rootkit must store code in a persistent store, such as the registry or file system, and configure a method by which the code executes without user intervention. This means it is easier to detect, as the copy in persistent storage can potentially be scanned.
- **Memory based:** Has no persistent code and therefore cannot survive a reboot. However, because it is only in memory, it can be harder to detect.
- **User mode:** Intercepts calls to APIs (application program interfaces) and modifies returned results. For example, when an application performs a directory listing, the return results don't include entries identifying the files associated with the rootkit.
- **Kernel mode:** Can intercept calls to native APIs in kernel mode.⁴ The rootkit can also hide the presence of a malware process by removing it from the kernel's list of active processes.
- **Virtual machine based:** This type of rootkit installs a lightweight virtual machine monitor and then runs the operating system in a virtual machine above it. The rootkit can then transparently intercept and modify states and events occurring in the virtualized system.
- **External mode:** The malware is located outside the normal operation mode of the targeted system, in BIOS or system management mode, where it can directly access hardware. This classification shows a

continuing arms race between rootkit authors, who exploit ever more stealthy mechanisms to hide their code, and those who develop mechanisms to harden systems against such subversion or to detect when it has occurred.

10.10 Countermeasures

Malware Countermeasure Approaches SP 800-83 lists four main elements of prevention: policy, awareness, vulnerability mitigation, and threat mitigation. Having a suitable policy to address malware prevention provides a basis for implementing appropriate preventative countermeasures.

4 The kernel is the portion of the OS that includes the most heavily used and most critical portions of software. Kernel mode is a privileged mode of execution reserved for the kernel. Typically, kernel mode allows access to regions of main memory that are unavailable to processes executing in a less privileged mode and also enables execution of certain machine instructions that are restricted to the kernel mode.

10.10 / Countermeasures 361

One of the first countermeasures that should be employed is to ensure all systems are as current as possible, with all patches applied, in order to reduce the number of vulnerabilities that might be exploited on the system. The next is to set appropriate access controls on the applications and data stored on the system, to reduce the number of files that any user can access, and hence potentially infect or corrupt, as a result of them executing some malware code. These measures directly target the key propagation mechanisms used by worms, viruses, and some trojans. The third common propagation mechanism, which targets users in a social engineering attack, can be countered using appropriate user awareness and training. This aims to equip users to be more aware of these attacks, and less likely to take actions that result in their compromise. SP 800-83 provides examples of suitable awareness issues. If prevention fails, then technical mechanisms can be used to support the following threat mitigation options:

- Detection: Once the infection has occurred, determine that it has occurred and locate the malware.
- Identification: Once detection has been achieved, identify the specific malware that has infected the system.
- Removal: Once the specific malware has been identified, remove all traces of malware virus from all infected systems so that it cannot spread further. If detection succeeds but either identification or removal is not possible, then the alternative is to discard any infected or malicious files and reload a clean backup version. In the case of some particularly nasty infections, this may require a complete wipe of all storage, and rebuild of the infected system from known clean media.

To begin, let us consider some requirements for effective malware countermeasures:

- Generality: The approach taken should be able to handle a wide variety of attacks.
- Timeliness: The approach should respond quickly so as to limit the number of infected programs or systems and the consequent activity.
- Resiliency: The approach should be resistant to evasion techniques employed by attackers to hide the presence of their malware.
- Minimal denial-of-service costs: The approach should result in minimal reduction in capacity or service due to the actions of the countermeasure software, and should not significantly disrupt normal operation.
- Transparency: The countermeasure software and devices should not require modification to existing (legacy) OSs, application software, and hardware.
- Global and local coverage: The approach should be able to deal with attack sources both from outside and inside the enterprise network. Achieving all these requirements often requires the use of multiple approaches. Detection of the presence of malware can occur in a number of locations. It may occur on the infected system, where some host-based “antivirus” program is running, monitoring data imported into the system, and the execution and behavior

362 chapter 10 / Malicious Software of programs running on the system. Or, it may take place as part of the perimeter security mechanisms used in an organization’s firewall and intrusion detection systems (IDSs).

Lastly, detection may use distributed mechanisms that gather data from both host-based and perimeter sensors, potentially over a large number of networks and organizations, in order to obtain the largest scale view of the movement of malware.

Host-Based Scanners

The first location where antivirus software is used is on each end system. This gives the software the maximum access to information not only on the behavior of the malware as it interacts with the targeted system but also on the smallest overall view of malware activity. The use of antivirus software on personal computers is now widespread, in part caused by the explosive growth in malware volume and activity. Advances in virus and other malware technology, and in antivirus technology and other countermeasures, go hand in hand. Early malware used relatively simple and easily detected code, and hence could be identified and purged with relatively simple antivirus software packages. As the malware arms race has evolved, both the malware code and, necessarily, antivirus software have grown more complex and sophisticated. [STEP93] identifies four generations of antivirus software: ■■ First generation: Simple scanners ■■ Second generation: Heuristic scanners ■■ Third generation: Activity traps ■■ Fourth generation: Full-featured protection

A first-generation scanner requires a malware signature to identify the malware. The signature may contain “wildcards” but matches essentially the same structure and bit pattern in all copies of the malware. Such signature-specific scanners are limited to the detection of known malware. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length as a result of virus infection. A second-generation scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable malware instances. One class of such scanners looks for fragments of code that are often associated with malware. For example, a scanner may look for the beginning of an encryption loop used in a polymorphic virus and discover the encryption key. Once the key is discovered, the scanner can decrypt the malware to identify it, and then remove the infection and return the program to service. Another second-generation approach is integrity checking. A checksum can be appended to each program. If malware alters or replaces some program without changing the checksum, then an integrity check will catch this change. To counter malware that is sophisticated enough to change the checksum when it alters a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the malware cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the malware is prevented from adjusting the program to produce the same hash code as before. If a protected list of programs in trusted locations is kept, this

10.10 / Countermeasures 363

approach can also detect attempts to replace or install rogue code or programs in these locations. Third-generation programs are memory-resident programs that identify malware by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide array of malware. Rather, it is necessary only to identify the small set of actions that indicate that malicious activity is being attempted and then to 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 malware to penetrate a system and then limits the ability of a malware to update files in order to propagate. The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures. These include more sophisticated antivirus approaches. We now highlight two

of the most important. **Host-Based Behavior-Blocking Software** Unlike heuristics or fingerprint-based scanners, behavior-blocking software integrates with the operating system of a host computer and monitors program behavior in real time for malicious actions [CONR02, NACH02]. The behavior blocking software then blocks potentially malicious actions before they have a chance to affect the system. Monitored behaviors can include the following: ■■ Attempts to open, view, delete, and/or modify files ■■ Attempts to format disk drives and other unrecoverable disk operations ■■ Modifications to the logic of executable files or macros ■■ Modification of critical system settings, such as start-up settings ■■ Scripting of e-mail and instant messaging clients to send executable content ■■ Initiation of network communications

Because a behavior blocker can block suspicious software in real time, it has an advantage over such established antivirus detection techniques as fingerprinting or heuristics. There are literally trillions of different ways to obfuscate and rearrange the instructions of a virus or worm, many of which will evade detection by a fingerprint scanner or heuristic. But eventually, malicious code must make a welldefined request to the operating system. Given that the behavior blocker can intercept all such requests, it can identify and block malicious actions regardless of how obfuscated the program logic appears to be. Behavior blocking alone has limitations. Because the malicious code must run on the target machine before all its behaviors can be identified, it can cause harm before it has been detected and blocked. For example, a new item of malware might shuffle a number of seemingly unimportant files around the hard drive before modifying a single file and being blocked. Even though the actual modification was blocked, the user may be unable to locate his or her files, causing a loss to productivity or possibly having worse consequences.

364 chapter 10 / **Malicious Software**

Spyware Detection and Removal Although general antivirus products include signatures to detect spyware, the threat this type of malware poses, and its use of stealthing techniques, means that a range of spyware specific detection and removal utilities exist. These specialize in the detection and removal of spyware, and provide more robust capabilities. Thus they complement, and should be used along with, more general antivirus products.

Rootkit Countermeasures Rootkits can be extraordinarily difficult to detect and neutralize, particularly so for kernel-level rootkits. Many of the administrative tools that could be used to detect a rootkit or its traces can be compromised by the rootkit precisely so that it is undetectable. Countering rootkits requires a variety of network- and computer-level security tools. Both network- and host-based IDSs can look for the code signatures of known rootkit attacks in incoming traffic. Host-based antivirus software can also be used to recognize the known signatures. Of course, there are always new rootkits and modified versions of existing rootkits that display novel signatures. For these cases, a system needs to look for behaviors that could indicate the presence of a rootkit, such as the interception of system calls or a keylogger interacting with a keyboard driver. Such behavior detection is far from straightforward. For example, antivirus software typically intercepts system calls. Another approach is to do some sort of file integrity check. An example of this is RootkitRevealer, a freeware package from SysInternals. The package compares the results of a system scan using APIs with the actual view of storage using instructions that do not go through an API. Because a rootkit conceals itself by modifying the view of storage seen by administrator calls, RootkitRevealer catches the discrepancy. If a kernel-level rootkit is detected, the only secure and reliable way to recover is to do an entire new OS install on the infected machine.

Perimeter Scanning Approaches The next location where antivirus software is used is on an organization's firewall and IDS. It is typically included in e-mail and Web proxy services running on these systems. It may also

be included in the traffic analysis component of an IDS. This gives the antivirus software access to malware in transit over a network connection to any of the organization's systems, providing a larger-scale view of malware activity. This software may also include intrusion prevention measures, blocking the flow of any suspicious traffic, thus preventing it reaching and compromising some target system, either inside or outside the organization. However, this approach is limited to scanning the malware content, as it does not have access to any behavior observed when it runs on an infected system. Two types of monitoring software may be used: ■■ Ingress monitors: These are located at the border between the enterprise network and the Internet. They can be part of the ingress-filtering software of a border router or external firewall or a separate passive monitor. A honeypot 10.10 / Countermeasures 365 can also capture incoming malware traffic. An example of a detection technique for an ingress monitor is to look for incoming traffic to unused local IP addresses. ■■ Egress monitors: These can be located at the egress point of individual LANs on the enterprise network as well as at the border between the enterprise network and the Internet. In the former case, the egress monitor can be part of the egress-filtering software of a LAN router or switch. As with ingress monitors, the external firewall or a honeypot can house the monitoring software. Indeed, the two types of monitors can be collocated. The egress monitor is designed to catch the source of a malware attack by monitoring outgoing traffic for signs of scanning or other suspicious behavior. Perimeter monitoring can also assist in detecting and responding to botnet activity by detecting abnormal traffic patterns associated with this activity. Once bots are activated and an attack is underway, such monitoring can be used to detect the attack. However, the primary objective is to try to detect and disable the botnet during its construction phase, using the various scanning techniques we have just discussed, identifying and blocking the malware that is used to propagate this type of payload. Perimeter Worm Countermeasures There is considerable overlap in techniques for dealing with viruses and worms. Once a worm is resident on a machine, antivirus software can be used to detect it, and possibly remove it. In addition, because worm propagation generates considerable network activity, perimeter network activity and usage monitoring can form the basis of a worm defense. Following [JHI07], we list six classes of worm defense that address the network activity it may generate: A. Signature-based worm scan filtering: This type of approach generates a worm signature, which is then used to prevent worm scans from entering/leaving a network/host. Typically, this approach involves identifying suspicious flows and generating a worm signature. This approach is vulnerable to the use of polymorphic worms: Either the detection software misses the worm or, if it is sufficiently sophisticated to deal with polymorphic worms, the scheme may take a long time to react. [NEWS05] is an example of this approach. B. Filter-based worm containment: This approach is similar to class A but focuses on worm content rather than a scan signature. The filter checks a message to determine if it contains worm code. An example is Vigilante [COST05], which relies on collaborative worm detection at end hosts. This approach can be quite effective but requires efficient detection algorithms and rapid alert dissemination. C. Payload-classification-based worm containment: These network-based techniques examine packets to see if they contain a worm. Various anomaly detection techniques can be used, but care is needed to avoid high levels of false positives or negatives. An example of this approach, which looks for exploit code in network flows, is reported in [CHIN05]. This approach does not generate signatures based on byte patterns but rather looks for control and data flow structures that suggest an exploit. 366 chapter 10 / Malicious Software D. Threshold random walk (TRW) scan detection: TRW

exploits randomness in picking destinations to connect to as a way of detecting if a scanner is in operation [JUNG04]. TRW is suitable for deployment in high-speed, lowcost network devices. It is effective against the common behavior seen in worm scans.

E. Rate limiting: This class limits the rate of scanlike traffic from an infected host. Various strategies can be used, including limiting the number of new machines a host can connect to in a window of time, detecting a high connection failure rate, and limiting the number of unique IP addresses a host can scan in a window of time. [CHEN04] is an example. This class of countermeasures may introduce longer delays for normal traffic. This class is also not suited for slow, stealthy worms that spread slowly to avoid detection based on activity level.

F. Rate halting: This approach immediately blocks outgoing traffic when a threshold is exceeded either in outgoing connection rate or in diversity of connection attempts [JHI07]. The approach must include measures to quickly unblock mistakenly blocked hosts in a transparent way. Rate halting can integrate with a signature- or filter-based approach so that once a signature or filter is generated, every blocked host can be unblocked. Rate halting appears to offer a very effective countermeasure. As with rate limiting, rate-halting techniques are not suitable for slow, stealthy worms.

Distributed Intelligence Gathering Approaches

The final location where antivirus software is used is in a distributed configuration. It gathers data from a large number of both host-based and perimeter sensors, relays this intelligence to a central analysis system able to correlate and analyze the data, which can then return updated signatures and behavior patterns to enable all of the coordinated systems to respond and defend against malware attacks. A number of such systems have been proposed. We discuss one such approach in the remainder of this section. Figure 10.4 shows an example of a distributed worm countermeasure architecture (based on [SID105]). The system works as follows (numbers in figure refer to numbers in the following list):

1. Sensors deployed at various network locations detect a potential worm. The sensor logic can also be incorporated in IDS sensors.
2. The sensors send alerts to a central server, which correlates and analyzes the incoming alerts. The correlation server determines the likelihood that a worm attack is being observed and the key characteristics of the attack.
3. The server forwards its information to a protected environment, where the potential worm may be sandboxed for analysis and testing.
4. The protected system tests the suspicious software against an appropriately instrumented version of the targeted application to identify the vulnerability.
5. The protected system generates one or more software patches and tests these.
6. If the patch is not susceptible to the infection and does not compromise the application's functionality, the system sends the patch to the application host to update the targeted application.

10.11 Distributed Denial of Service Attacks

A denial-of-service (DoS) attack is an attempt to prevent legitimate users of a service from using that service. When this attack comes from a single host or network node, then it is simply referred to as a DoS attack. A more serious threat is posed by a DDoS attack. DDoS attacks make computer systems inaccessible by flooding servers, networks, or even end-user systems with useless traffic so that legitimate users can no longer gain access to those resources. In a typical DDoS attack, a large number of compromised hosts are amassed to send useless packets. This section is concerned with DDoS attacks. First, we look at the nature and types of attacks. Next, we examine methods by which an attacker is able to recruit a network of hosts for attack launch. Finally, this section looks at countermeasures.

Figure 10.4 Placement of Worm Monitors

The diagram illustrates a distributed worm countermeasure architecture. It shows a flow from the Internet through various sensors (Remote, Honeypot, Passive, Firewall) to a Correlation server. The Correlation server then interacts with a Sandboxed environment, which contains Instrumented applications. These applications are connected to an Enterprise network, which includes an Application server. A Hypothesis is also shown as part of the system.