6.1 TYPES OF MALICIOUS SOFTWARE (MALWARE)

The terminology in this area presents problems because of a lack of universal agreement on all of the terms and because some of the categories overlap. **Table 6.1** is a useful guide to some of the terms in use.

Table 6.1 Terminology for Malicious Software (Malware)

Name	Description
Advanced Persistent Threat (APT)	Cybercrime directed at business and political targets, using a wide variety of intrusion technologies and malware, applied persistently and effectively to specific targets over an extended period, often attributed to state-sponsored organizations.
Adware	Advertising that is integrated into software. It can result in pop-up ads or redirection of a browser to a commercial site.
Attack kit	Set of tools for generating new malware automatically using a variety of supplied propagation and payload mechanisms.
Auto-rooter	Malicious hacker tools used to break into new machines remotely.
Backdoor (trapdoor)	Any mechanism that bypasses a normal security check; it may allow unauthorized access to functionality in a program, or onto a compromised system.
Downloaders	Code that installs other items on a machine that is under attack. It is normally included in the malware code first inserted on to a compromised system to then import a larger malware package.
Drive-by-download	An attack using code on a compromised website that exploits a browser vulnerability to attack a client system when the site is viewed.

Exploits	Code specific to a single vulnerability or set of vulnerabilities.
Flooders (DoS client)	Used to generate a large volume of data to attack networked computer systems, by carrying out some form of denial-of-service (DoS) attack.
Keyloggers	Captures keystrokes on a compromised system.
Logic bomb	Code inserted into malware by an intruder. A logic bomb lies dormant until a predefined condition is met; the code then triggers some payload.
Macro virus	A type of virus that uses macro or scripting code, typically embedded in a document or document template, and triggered when the document is viewed or edited, to run and replicate itself into other such documents.
Mobile code	Software (e.g., script and macro) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics.
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access.
Spammer programs	Used to send large volumes of unwanted e-mail.
Spyware	Software that collects information from a computer and transmits it to another system by monitoring keystrokes, screen data, and/or network traffic; or by scanning files on the system for sensitive information.
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 it.
Virus	Malware that, when executed, tries to replicate itself into other executable machine or script 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, by exploiting software vulnerabilities in the target system, or using captured authorization credentials.
Zombie, bot	Program installed on an infected machine that is activated to launch attacks on other machines.

A Broad Classification of Malware

A number of authors attempt to classify malware, as shown in the survey and proposal of [HANS04]. Although a range of aspects 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, 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.

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, 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 later of more general attack kits in the 2000s. These 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 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]. The Angler exploit kit, first seen in 2013, was the most active kit seen in 2015, often distributed via malvertising that exploited Flash vulnerabilities. It is sophisticated and technically advanced, in both attacks executed and counter-measures deployed to resist detection. There are a number of other attack kits in active use, though the specific kits change from year to year as attackers continue to evolve and improve them [SYMA16].

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 crime; organizations that sell their services to companies and nations, and national government agencies, as we will discuss in **Section 8.1**. This has significantly changed the resources available and motivation behind the rise of malware, and indeed has led to the development of a large underground economy involving the sale of attack kits, access to compromised hosts, and to stolen information.

6.2 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 will discuss these categories of intruders further in **Section 8.1**.

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:** Use 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 also 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, and drive-by-downloads from selected compromised Web sites likely to be visited by personnel in the target organization. The intent is 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 will discuss later in this chapter, as well as awareness training to assist personnel to resist such attacks, as we will discuss in **Chapter 17**. Even with current best-practice countermeasures, the use of zero-day exploits and new attack approaches means that some of these attacks are likely to succeed

[SYMA16, 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.

6.3 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. Computer virus infections formed the majority of malware seen in the early personal computer era. The term “computer virus” is still often used to refer to malware in general, rather than just computer viruses specifically. More recently, the virus software 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. Computer viruses first appeared in the early 1980s, and the term itself is attributed to Fred Cohen. Cohen is the author of a groundbreaking book on the subject [COHE94]. The Brain virus, first seen in 1986, was one of the first to target MSDOS systems, and resulted in a significant number of infections for this time.

Biological viruses are tiny scraps of genetic code—DNA or RNA—that can take over the machinery of a living cell and trick it into making thousands of flawless replicas of the original virus. Like its biological counterpart, 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 documents 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.

[AYCO06] states that a computer virus has three parts. More generally, many contemporary types of malware also include 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 may involve 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 however target specific document types, which are often supported on a variety of systems.

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 of the other files on that system with executable content 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.

Macro and Scripting Viruses

In the mid-1990s, macro or scripting code viruses became by far the most prevalent type of virus. NISTIR 7298 (*Glossary of Key Information Security Terms,* May 2013) defines a **macro virus** as a virus that attaches itself to documents and uses the macro programming capabilities of the document's application to execute and propagate. 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 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, particularly since these documents can sometimes be opened automatically without prompting the user.
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.
5. Macro viruses are much easier to write or to modify than traditional executable viruses.

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.

Microsoft Word and Excel documents are common targets due to their widespread use. 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. Office 2000 improved macro security by allowing macros to be digitally signed by their author, and for authors to be listed as trusted. Users were then warned if a document being opened contained unsigned, or signed but untrusted, macros, and were advised to disable macros in this case. Various anti-virus product vendors have also developed tools to detect and remove macro viruses. As in other types of malware, the arms race continues in the field of macro viruses, but they no longer are the predominant malware 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 their system. Alternatively, it can install a Trojan, or act as a worm, as we will discuss later [STEV11].

Macro Virus Structure

Although macro languages may have a similar syntax, the details depend on the application interpreting the macro, and so will always target documents for a specific application. For example, a Microsoft Word macro, including a macro virus, will be different to an Excel macro. Macros can either be saved with a document, or be saved in a global template or worksheet. Some macros are run automatically when certain actions occur. In Microsoft Word, for example, macros can run when Word starts, a document is opened, a new document is created, or when a document is closed. Macros can perform a wide range of operations, not just only on the document content, but can read and write files, and call other applications.

As an example of the operation of a macro virus, pseudo-code for the Melissa macro virus is shown in **Figure 6.1**. This was a component of the Melissa e-mail worm that we will describe further in the next section. This code would be introduced onto a system by opening an infected Word document, most likely sent by e-mail. This macro code is contained in the Document_Open macro, which is automatically run when the document is opened. It first disables the Macro menu and some related security features, making it harder for the user stop or remove its operation. Next it checks to see if it is being run from an infected document, and if so copies itself into the global template file. This file is opened with every subsequent document, and the macro virus run, infecting that document. It then checks to see if it has been run on this system before, by looking to see if a specific key “Melissa” has been added to the registry. If that key is absent, and Outlook is the e-mail client, the macro virus then sends a copy of the current, infected document to each of the first 50 addresses in the current user’s Address Book. It then creates the “Melissa” registry entry, so this is only done once on any system. Finally it checks the current time and date for a specific trigger condition, which if met results in a Simpson quote being inserted into the current document. Once the macro virus code has finished, the document continues opening and the user can then edit as normal. This code illustrates how a macro virus can manipulate both the document contents, and access other applications on the system. It also shows two infection mechanisms, the first infecting every subsequent document opened on the system, the second sending infected documents to other users via e-mail.

```
macro Document_Open
   disable Macro menu and some macro security features
   if called from a user document
      copy macro code into Normal template file
   else
      copy macro code into user document being opened
   end if
   if registry key "Melissa" not present
      if Outlook is email client
         for first 50 addresses in address book
            send email to that address
            with currently infected document attached
         end for
      end if
      create registry key "Melissa"
   end if
   if minute in hour equals day of month
      insert text into document being opened
   end if
end macro
```

Figure 6.1 Melissa Macro Virus Pseudo-code

More sophisticated macro virus code can use stealth techniques such as encryption or polymorphism, changing its appearance each time, to avoid scanning detection.

Viruses Classification

There has been a continuous arms race between virus writers and writers of anti-virus 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 anti-virus 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 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 form of virus that uses encryption to obscure it’s content. 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 anti-virus software. Thus, the entire virus, not just a payload, is hidden. It may use code mutation, compression, or 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, using multiple transformation techniques, increasing the difficulty of detection. Metamorphic viruses may change their behavior as well as their appearance.

6.4 PROPAGATION—VULNERABILITY EXPLOIT—WORMS

The next category of malware propagation concerns the exploit of software vulnerabilities, such as those we will discuss in **Chapters 10** and **11**, which are commonly exploited by computer worms, and in hacking attacks on systems. 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 CD and DVD data disks. E-mail worms can 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.

The concept of a computer worm was introduced in John Brunner's 1975 SF novel *The Shockwave Rider*. The first known worm implementation was done in Xerox Palo Alto Labs in the early 1980s. It was nonmalicious, searching for idle systems to use to run a computationally intensive task.

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:

- **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 (as we will discuss in **Chapters 10** and **11**).
- **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 on 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 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 can be 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:

$$\frac{dI(t)}{dt} = \beta I(t)\, S(t)$$

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

β = infection rate

N = size of the population, $N = I(t) + S(t)$

Figure 6.2 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 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.

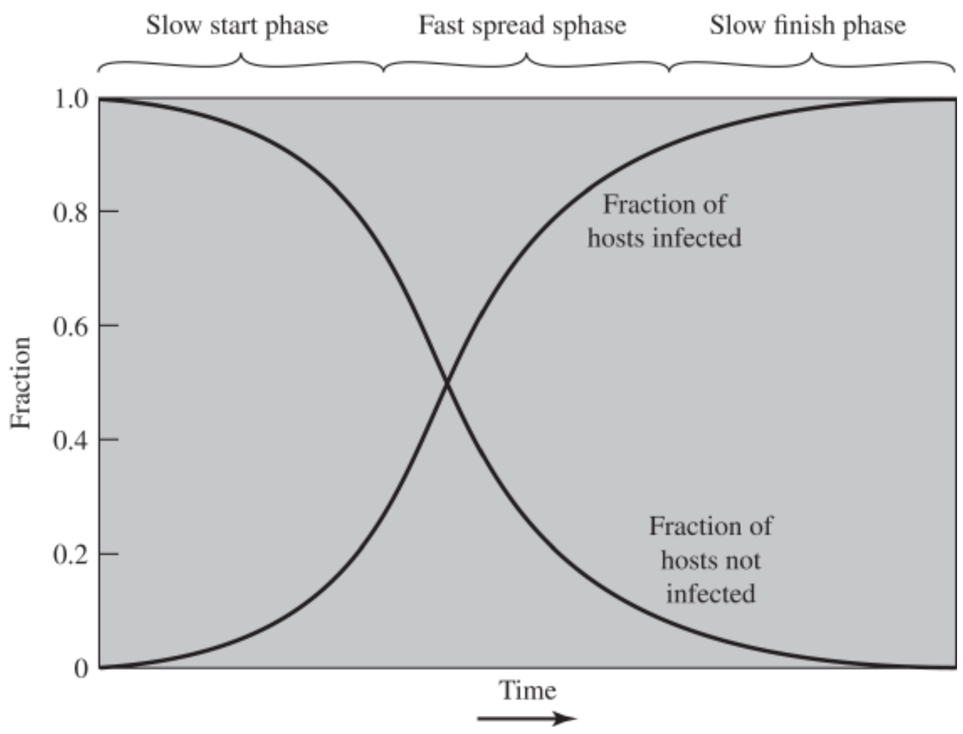

Figure 6.2 Worm Propagation Model

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 et al. [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 6.2.

The Morris Worm

Arguably, the earliest significant, and hence well-known, 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 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 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 candidates[1].

      [1] The complete list is provided at this book's website.

   c. All the words in the local system dictionary.

2. It exploited a bug in the UNIX finger protocol, which reports the whereabouts of a remote user.
3. 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.

A Brief History of Worm Attacks

The Melissa e-mail worm that appeared in 1998 was the first of a new generation of malware that included aspects of virus, worm, and Trojan in one package [CASS01]. Melissa made use of a Microsoft Word macro embedded in an attachment, as we described in the previous section. If the recipient opens the e-mail attachment, the Word macro is activated. Then it:

1. Sends itself to everyone on the mailing list in the user's e-mail package, propagating as a worm; and
2. Does local damage on the user's system, including disabling some security tools, and also copying itself into other documents, propagating as a virus; and
3. If a trigger time was seen, it displayed a Simpson quote as its payload.

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

Melissa propagates itself as soon as it is activated (either by opening an e-mail attachment or 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, this next generation of malware could do so in hours. [CASS01] notes that it took only three days for Melissa to infect over 100,000 computers, compared to the months it took the Brain virus to infect a few thousand computers a decade before. This makes it very difficult for anti-virus software to respond to new attacks before much damage is done.

The Code Red worm first appeared in July 2001. Code Red exploits a security hole in the Microsoft Internet Information Server (IIS) to penetrate and spread. It also disables the system file checker in Windows. The worm probes random IP addresses to spread to other hosts. During a certain period of time, it only spreads. It then initiates a denial-of-service attack against a government website by flooding the site with packets from numerous hosts. The worm then suspends activities and reactivates periodically. In the second wave of attack, Code Red infected nearly 360,000 servers in 14 hours. In addition to the havoc it caused at the targeted server, Code Red consumed enormous amounts of Internet capacity, disrupting service [MOOR02].

Code Red II is another distinct variant that first appeared in August 2001, and also targeted Microsoft IIS. It tried to infect systems on the same subnet as the infected system. Also, this newer worm installs a backdoor, allowing a hacker to remotely execute commands on victim computers.

The Nimda worm that appeared in September 2001 also has worm, virus, and mobile code characteristics. It spread using a variety of distribution methods:

- **E-mail:** A user on a vulnerable host opens an infected e-mail attachment; Nimda looks for e-mail addresses on the host then sends copies of itself to those addresses.
- **Windows shares:** Nimda scans hosts for unsecured Windows file shares; it can then use NetBIOS86 as a transport mechanism to infect files on that host in the hopes that a user will run an infected file, which will activate Nimda on that host.
- **Web servers:** Nimda scans Web servers, looking for known vulnerabilities in Microsoft IIS. If it finds a vulnerable server, it attempts to transfer a copy of itself to the server and infects it and its files.
- **Web clients:** If a vulnerable Web client visits a Web server that has been infected by Nimda, the client’s workstation will become infected.
- **Backdoors:** If a workstation was infected by earlier worms, such as “Code Red II,” then Nimda will use the backdoor access left by these earlier infections to access the system.

In early 2003, the SQL Slammer worm appeared. This worm exploited a buffer overflow vulnerability in Microsoft SQL server. The Slammer was extremely compact and spread rapidly, infecting 90% of vulnerable hosts within 10 minutes. This rapid spread caused significant congestion on the Internet.

Late 2003 saw the arrival of the Sobig.F worm, which exploited open proxy servers to turn infected machines into spam engines. At its peak, Sobig.F reportedly accounted for one in every 17 messages and produced more than one million copies of itself within the first 24 hours.

Mydoom is a mass-mailing e-mail worm that appeared in 2004. It followed the growing trend of installing a backdoor in infected computers, thereby enabling hackers to gain remote access to data such as passwords and credit card numbers. Mydoom replicated up to 1,000 times per minute and reportedly flooded the Internet with 100 million infected messages in 36 hours.

The Warezov family of worms appeared in 2006 [KIRK06]. When the worm is launched, it creates several executables in system directories and sets itself to run every time Windows starts by creating a registry entry. Warezov scans several types of files for e-mail addresses and sends itself as an e-mail attachment. Some variants are capable of downloading other malware, such as Trojan horses and adware. Many variants disable security-related products and/or disable their updating capability.

The Conficker (or Downadup) worm was first detected in November 2008 and spread quickly to become one of the most widespread infections since SQL Slammer in 2003 [LAWT09]. It spread initially by exploiting a Windows buffer overflow vulnerability, though later versions could also spread via USB drives and network file shares. Recently, it still comprised the second most common family of malware observed by Symantec [SYMA16], even though patches were available from Microsoft to close the main vulnerabilities it exploits.

In 2010, the Stuxnet worm was detected, though it had been spreading quietly for some time previously [CHEN11, KUSH13]. Unlike many previous worms, it deliberately restricted its rate of spread to reduce its chance of detection. It also targeted industrial control systems, most likely those associated with the Iranian nuclear program, with the likely aim of disrupting the operation of their equipment. It supported a range of propagation mechanisms, including via USB drives, network file shares, and using no less than four unknown, zero-day vulnerability exploits. Considerable debate resulted from the size and complexity of its code, the use of an unprecedented four zero-day exploits, and the cost and effort apparent in its development. There are claims that it appears to be the first serious use of a cyberwarfare weapon against a nation’s physical infrastructure. The researchers who analyzed Stuxnet noted that while they were expecting to find espionage, they never expected to see malware with targeted sabotage as its aim. As a result, greater attention is now being directed at the use of malware as a weapon by a number of nations.

In late 2011, the Duqu worm was discovered, which uses code related to that in Stuxnet. Its aim is different, being cyber-espionage, though it appears to also target the Iranian nuclear program. Another prominent, recent, cyber-espionage worm is the Flame family, which was discovered in 2012 and appears to target Middle-Eastern countries. Despite the specific target areas for these various worms, their infection strategies have been so successful that they have been identified on computer systems in a very large number of countries, including on systems kept physically isolated from the general Internet. This reinforces the need for significantly improved countermeasures to resist such infections.

In May 2017, the WannaCry ransomware attack spread extremely rapidly over a period of hours to days, infecting hundreds of thousands of systems belonging to both public and private organisations in more than 150 countries (US-CERT Alert TA17-132A) [GOOD17]. It spread as a worm by aggressively scanning both local and random remote networks, attempting to exploit a vulnerability in the SMB file sharing service on unpatched Windows systems. This rapid spread was only slowed by the accidental activation of a “kill-switch” domain by a UK security researcher, whose existence was checked for in the initial versions of this malware. Once installed on infected systems, it also encrypted files, demanding a ransom payment to recover them, as we will discuss later.

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.
- **Multi-exploit:** 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 virus polymorphic techniques. 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, 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. In 2015, 54 zero-day exploits were discovered and exploited, significantly more than in previous years [SYMA16]. Many of these were in common computer and mobile software. Some, though, were in common libraries and development packages, and some in industrial control systems. This indicates the range of systems being targeted.

Mobile Code

NIST 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 executed with identical semantics.

Mobile code is transmitted from a remote system to a local system 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 methods of using mobile code for malicious operations on local system are cross-site scripting, interactive and dynamic websites, e-mail attachments, and downloads from untrusted sites or of untrusted software.

Mobile Phone Worms

Worms first appeared on mobile phones with the discovery of the Cabir worm in 2004, then Lasco and CommWarrior in 2005. These worms communicate through Bluetooth wireless connections or via the multimedia messaging service (MMS). The target is the smartphone, which is a mobile phone that permits users to install software applications from sources other than the cellular network operator. All these early mobile worms targeted mobile phones using the Symbian operating system. More recent malware targets Android and iPhone systems. Mobile phone malware can completely disable the phone, delete data on the phone, or force the device to send costly messages to premium-priced numbers.

The CommWarrior worm replicates by means of Bluetooth to other phones in the receiving area. It also sends itself as an MMS file to numbers in the phone’s address book and in automatic replies to incoming text messages and MMS messages. In addition, it copies itself to the removable memory card and inserts itself into the program installation files on the phone.

Although these examples demonstrate that mobile phone worms are possible, the vast majority of mobile phone malware observed use trojan apps to install themselves [SYMA16].

Client-Side Vulnerabilities and Drive-by-Downloads

Another approach to exploiting software vulnerabilities involves the exploit of bugs in user applications to install malware. A common technique exploits browser and plugin vulnerabilities so when the user views a webpage controlled by the attacker, it contains code that exploits the 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. Multiple vulnerabilities in the Adobe Flash Player and Oracle Java plugins have been exploited by attackers over many years, to the point where many browsers are now removing support for them. In most cases, this malware does not actively propagate as a worm does, but rather waits for unsuspecting users to visit the malicious webpage in order to spread to their systems [SYMA16].

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 websites they are likely to visit, 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 websites without actually compromising them. The attacker pays for advertisements that are highly likely to be placed on their intended target websites, and which incorporate malware in them. Using these malicious adds, attackers can infect visitors to sites displaying them. Again, the malware code may be dynamically generated to either reduce the chance of detection, or to only infect specific systems. Malvertising has grown rapidly in recent years, as they are easy to place on desired websites with few questions asked, and are hard to track. Attackers have placed these ads for as little as a few hours, when they expect their intended victims could be browsing the targeted websites, greatly reducing their visibility [SYMA16].

Other 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 will discuss in the next section.

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 websites 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.

6.5 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

With the explosive growth of the Internet over the last few decades, the widespread use of e-mail, and the extremely low cost required to send large volumes of e-mail, has come the rise of unsolicited bulk e-mail, commonly known as spam. [SYMA16] notes that more than half of inbound business e-mail traffic is still spam, despite a gradual decline in recent years. This 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 this explosive growth, there has been the equally rapid growth of the anti-spam industry that 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 with the defenders, efforts to block them [KREI09].

However, the spam problem continues, as spammers exploit other means of reaching their victims. This includes the use of social media, reflecting the rapid growth in the use of these networks. For example, [SYMA16] described a successful weight-loss spam campaign that exploited hundreds of thousands of fake Twitter accounts, mutually supporting and reinforcing each other, to increase their credibility and likelihood of users following them, and then falling for the scam. Social network scams often rely on victims sharing the scam, or on fake offers with incentives, to assist their spread.

While some spam e-mail is sent from legitimate mail servers using stolen user credentials, most recent spam is sent by botnets using compromised user systems, as we will discuss in **Section 6.6**. A significant portion of spam e-mail content is just advertising, trying to convince the recipient to purchase some product online, such as pharmaceuticals, or used in scams, such as stock, romance or fake trader 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 will discuss next. Finally the spam may be used in a phishing attack, typically directing the user either to a fake website 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. In recent years, the evolving criminal marketplace makes phishing campaigns easier by selling packages to scammers that largely automate the process of running the scam [SYMA16]. 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. Hence the importance of providing appropriate security awareness training to users, so they are better able to recognize and respond appropriately to such e-mails, as we will discuss in **Chapter 17**.

Trojan Horses

A Trojan horse[2] is a useful, or apparently useful, program or utility containing hidden code that, when invoked, performs some unwanted or harmful function.

[2] In Greek mythology, the Trojan horse was used by the Greeks during their siege of Troy. Epeios constructed a giant hollow wooden horse in which 30 of the most valiant Greek heroes concealed themselves. The rest of the Greeks burned their encampment and pretended to sail away but actually hid nearby. The Trojans, convinced the horse was a gift and the siege over, dragged the horse into the city. That night, the Greeks emerged from the horse and opened the city gates to the Greek army. A bloodbath ensued, resulting in the destruction of Troy and the death or enslavement of all its citizens.

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 webform 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 anti-virus 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.

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. It was typically distributed using either spam e-mail or via a compromised website using a “watering-hole” attack. Tech Support Scams are a growing social engineering concern. These involve call centers calling users about non-existent problems on their computer systems. If the users respond, the attackers try to sell them bogus tech support or ask them to install Trojan malware or other unwanted applications on their systems, all while claiming this will fix their problem [SYMA16].

Mobile Phone Trojans

Mobile phone Trojans also first appeared in 2004 with the discovery of Skuller. As with mobile worms, the target is the smartphone, and the early mobile Trojans targeted Symbian phones. More recently, a significant number of Trojans have been detected that target Android phones and Apple iPhones. These Trojans are usually distributed via one or more of the app marketplaces for the target phone O/S.

The rapid growth in smartphone sales and use, which increasingly contain valuable personal information, make them an attractive target for criminals and other attackers. Given five in six new phones run Android, they are a key target [SYMA16]. The number of vulnerabilities discovered in, and malware families targeting these phones, have both increased steadily in recent years. Recent examples include a phishing Trojan that tricks the user into entering their banking details, and ransomware that mimics Google’s design style to appear more legitimate and intimidating.

The tighter controls that Apple impose on their app store, mean that many iPhone Trojans target “jail-broken” phones, and are distributed via unofficial sites. However a number of versions of the iPhone O/S contained some form of graphic or PDF vulnerability. Indeed these vulnerabilities were the main means used to “jail-break” the phones. But they also provided a path that malware could use to target the phones. While Apple has fixed a number of these vulnerabilities, new variants continued to be discovered. This is yet another illustration of just how difficult it is, for even well-resourced organizations, to write secure software within a complex system, such as an operating system. We will return to this topic in **Chapters 10** and **11**. More recently in 2015, XcodeGhost malware was discovered in a number of legitimate Apple Store apps. The apps were not intentionally designed to be malicious, but their developers used a compromised Xcode development system that covertly installed the malware as the apps were created [SYMA16]. This is one of several examples of attackers exploiting the development or enterprise provisioning infrastructure to assist malware distribution.

6.6 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.

Data Destruction and Ransomware

The Chernobyl virus is an early example of a destructive parasitic memory-resident Windows-95 and 98 virus, which was first seen in 1998. It infects executable files when they are opened. And when a trigger date is reached, the virus deletes data on the infected system by overwriting the first megabyte of the hard drive with zeroes, resulting in massive corruption of the entire file system. This first occurred on April 26, 1999, when estimates suggest more than one million computers were affected.

Similarly, the Klez mass-mailing worm is an early example of a destructive worm infecting Windows-95 to XP systems, and was first seen in October 2001. It spreads by e-mailing copies of itself to addresses found in the address book and in files on the system. It can stop and delete some anti-virus programs running on the system. On trigger dates, being the 13th of several months each year, it causes files on the local hard drive to become empty.

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 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 appeared, 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. [SYMA16, VERI16] note that ransomware is a growing challenge, comprising one of the most common types of malware installed on systems, and is often spread via “drive-by-downloads” or via SPAM e-mails.

The WannaCry ransomware, that we mentioned earlier in our discussion of Worms, infected a large number of systems in many countries in May 2017. When installed on infected systems, it encrypted a large number of files matching a list of particular file types, and then demanded a ransom payment in Bitcoins to recover them. Once this had occurred, recovery of this information was generally only possible if the organization had good backups, and an appropriate incident response and disaster recovery plan, as we will discuss in **Chapter 17**. The WannaCry ransomware attack generated a significant amount of media attention, in part due to the large number of affected organizations, and the significant costs they incurred in recovering from it. The targets for these attacks have widened beyond personal computer systems to include mobile devices and Linux servers. And tactics such as threatening to publish sensitive personal information, or to permanently destroy the encryption key after a short period of time, are sometimes used to increase the pressure on the victim to pay up.

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 mentioned above not only corrupts data, but 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 re-programmed or replaced.

More recently, the Stuxnet worm that we discussed previously targets some specific industrial control system software as its key payload [CHEN11, KUSH13]. 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 earlier discussion, this has raised concerns over the use of sophisticated targeted malware for industrial sabotage.

The British Government’s 2015 Security and Defense Review noted their growing concerns over the use of cyber attacks against critical infrastructure by both state-sponsored and non state actors. The December 2015 attack that disrupted Ukrainian power systems shows these concerns are well-founded, given that much critical infrastructure is not sufficiently hardened to resist such attacks [SYMA16].

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 to halt, or do some other damage.

A striking example of how logic bombs can be employed was the case of Tim Lloyd, who was convicted of setting a logic bomb that cost his employer, Omega Engineering, more than $10 million, derailed its corporate growth strategy, and eventually led to the layoff of 80 workers [GAUD00]. Ultimately, Lloyd was sentenced to 41 months in prison and ordered to pay $2 million in restitution.

6.7 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 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 compromised systems are not just personal computers, but include servers, and recently embedded devices such as routers or surveillance cameras. 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 will examine DDoS attacks in **Chapter 7**.
- **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 clear-text 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.
- **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 website with some advertisements: The operator of this website 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 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 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 especially 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 by some form of command-and-control (C&C) server network. This contact does not need to be continuous, but can be initiated periodically when the bot observes it has network access.

An early means of implementing the remote control facility used 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.

Originally these C&C servers used fixed addresses, which meant they could be located and potentially taken over or removed by law enforcement agencies. Some more recent malware families have used techniques such as the automatic generation of very large numbers of server domain names that the malware will try to contact. If one server name is compromised, the attackers can setup a new server at another name they know will be tried. To defeat this requires security analysts to reverse engineer the name generation algorithm, and to then attempt to gain control over all of the extremely large number of possible domains. Another technique used to hide the servers is fast-flux DNS, where the address associated with a given server name is frequently changed, often every few minutes, to rotate over a large number of server proxies, usually other members of the botnet. Such approaches hinder attempts by law enforcement agencies to respond to the botnet threat.

Once a communications path is established between a control module and the bots, the control module can manage 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. The control module can also collect information gathered by the bots that the attacker can then exploit. One effective counter measure against a botnet is to take-over or shutdown its C&C network. Increasing cooperation and coordination between law enforcement agencies in a number of countries resulted in a growing number of successful C&C seizures in recent years [SYMA16], and the consequent suppression of their associated botnets. These actions also resulted in criminal charges on a number of people associated with them.

6.8 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.

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 webpage requests to fake sites controlled by the attacker, and dynamically modifying data exchanged between the browser and certain websites of interest, all of which can result in significant compromise of the user's personal information.

The Zeus banking Trojan, created from its crimeware toolkit, is a prominent example of such spyware that has been widely deployed [BINS10]. It steals banking and financial credentials using both a keylogger and capturing and possibly altering form data for certain websites. It is typically deployed using either spam e-mails or via a compromised website in a “drive-by-download.”

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 website 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 their account, to prevent it being locked. If the user is careless, and does not realize that they are being conned, then following the link and supplying the requested details will certainly result in the attackers exploiting their account using the captured credentials.

More generally, such a spam e-mail may direct a user to a fake website 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 and 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, but containing malicious attachments disguised as fake invoices, office documents, or other expected content. 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 them of its authenticity. This greatly increases the likelihood of the recipient responding as desired by the attacker. This type of attack is particularly used in industrial and other forms of espionage, or in financial fraud such as bogus wire-transfer authorizations, by well-resourced organizations. Whether as a result of phishing, drive-by-download, or direct hacker attack, the number of incidents, and the quantity of personal records exposed, continues to grow. For example, the Anthem medical data breach in January 2015 exposed more than 78 million personal information records that could potentially be used for identity theft. The well-resourced Black Vine cyber-espionage group is thought responsible for this attack [SYMA16].

Reconnaissance, Espionage, and Data Exfiltration

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 [SYMA16]. 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, KUSH13]. There are a number of other high-profile examples of mass record exposure. These include the Wikileaks leak of sensitive military and diplomatic documents by Chelsea (born Bradley) Manning in 2010, and the release of information on NSA surveillance programs by Edward Snowden in 2013. Both of these are examples of insiders exploiting their legitimate access rights to release information for ideological reasons. And both resulted in significant global discussion and debate on the consequences of these actions. In contrast, the 2015 release of personal information on the users of the Ashley Madison adult website, and the 2016 Panama Papers leak of millions of documents relating to off-shore entities used as tax havens in at least some cases, are thought to have been carried out by outside hackers attacking poorly secured systems. Both have resulted in serious consequences for some of the people named in these leaks.

APT attacks may result in the loss of large volumes of sensitive information, which is sent, exfiltrated from the target organization, to the attackers. To detect and block such data exfiltration requires suitable “data-loss” technical counter measures that manage either access to such information, or its transmission across the organization’s network perimeter.

6.9 PAYLOAD—STEALTHING—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. Programmers have used backdoors legitimately for many years to debug and test programs; such a backdoor is called a maintenance hook. This usually is done when the programmer is developing an application that has an authentication procedure, or a long setup, requiring the user to enter many different values to run the application. To debug the program, the developer may wish to gain special privileges or to avoid all the necessary setup and authentication. The programmer may also want to ensure that there is a method of activating the program should something be wrong with the authentication procedure that is being built into the application. 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.

Backdoors become threats when unscrupulous programmers use them to gain unauthorized access. The backdoor was the basic idea for the vulnerability portrayed in the 1983 movie *War Games*. Another example is that during the development of Multics, penetration tests were conducted by an Air Force “tiger team” (simulating adversaries). One tactic employed was to send a bogus operating system update to a site running Multics. The update contained a Trojan horse that could be activated by a backdoor and that allowed the tiger team to gain access. The threat was so well-implemented that the Multics developers could not find it, even after they were informed of its presence [ENGE80].

In more recent times, a backdoor is usually implemented as a network service listening on some non-standard port that the attacker can connect to and issue commands through to be run on the compromised system. The WannaCry ransomware, that we described earlier in this chapter, included such a backdoor.

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.

3On UNIX systems, the administrator, or *superuser*, account is called root; hence the term *root access*.

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 do not 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.
  4The 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.

- **Virtual machine based:** This type of rootkit installs a lightweight virtual machine monitor, 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. Much of this advance is associated with finding “layer-below” forms of attack. The early rootkits worked in user mode, modifying utility programs and libraries in order to hide their presence. The changes they made could be detected by code in the kernel, as this operated in the layer below the user. Later-generation rootkits used more stealthy techniques, as we will discuss next.

Kernel Mode Rootkits

The next generation of rootkits moved down a layer, making changes inside the kernel and co-existing with the operating systems code, in order to make their detection much harder. Any “anti-virus” program would now be subject to the same “low-level” modifications that the rootkit uses to hide its presence. However, methods were developed to detect these changes.

Programs operating at the user level interact with the kernel through system calls. Thus, system calls are a primary target of kernel-level rootkits to achieve concealment. As an example of how rootkits operate, we look at the implementation of system calls in Linux. In Linux, each system call is assigned a unique *syscall number*. When a user-mode process executes a system call, the process refers to the system call by this number. The kernel maintains a system call table with one entry per system call routine; each entry contains a pointer to the corresponding routine. The syscall number serves as an index into the system call table.

[LEVI06] lists three techniques that can be used to change system calls:

- **Modify the system call table:** The attacker modifies selected syscall addresses stored in the system call table. This enables the rootkit to direct a system call away from the legitimate routine to the rootkit’s replacement. Figure 6.3 shows how the knark rootkit achieves this.

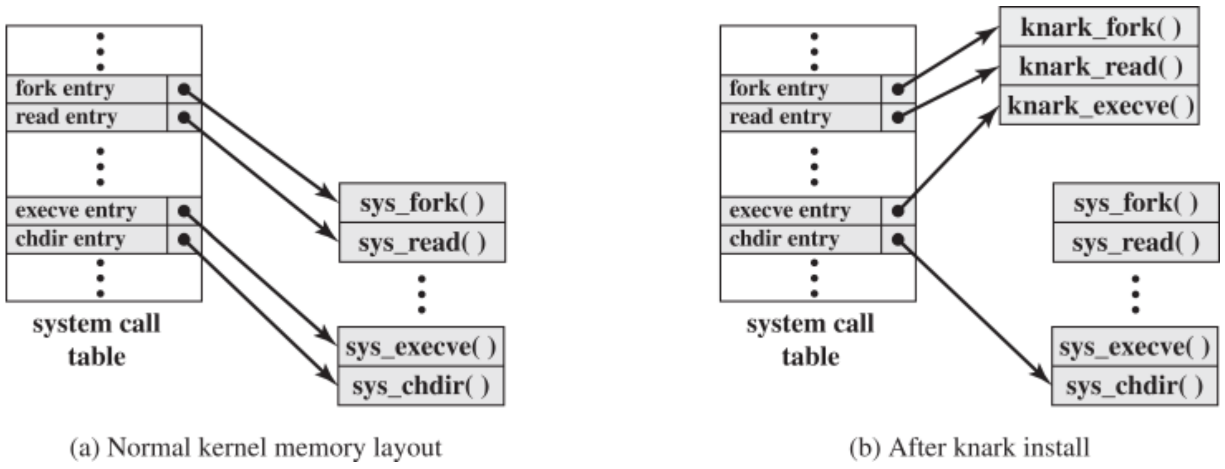

Figure 6.3 System Call Table Modification by Rootkit

- **Modify system call table targets:** The attacker overwrites selected legitimate system call routines with malicious code. The system call table is not changed.

- **Redirect the system call table:** The attacker redirects references to the entire system call table to a new table in a new kernel memory location.

Virtual Machine and Other External Rootkits

The latest generation of rootkits uses code that is entirely invisible to the targeted operating system. This can be done using a rogue or compromised virtual machine monitor or hypervisor, often aided by the hardware virtualization support provided in recent processors. The rootkit code then runs entirely below the visibility of even kernel code in the targeted operating system, which is now unknowingly running in a virtual machine, and capable of being silently monitored and attacked by the code below [SKAP07].

Several prototypes of virtualized rootkits were demonstrated in 2006. SubVirt attacked Windows systems running under either Microsoft’s Virtual PC or VMware Workstation hypervisors by modifying the boot process they used. These changes did make it possible to detect the presence of the rootkit.

However, the Blue Pill rootkit was able to subvert a native Windows Vista system by installing a thin hypervisor below it, then seamlessly continuing execution of the Vista system in a virtual machine. As it only required the execution of a rogue driver by the Vista kernel, this rootkit could install itself while the targeted system was running, and is much harder to detect. This type of rootkit is a particular threat to systems running on modern processors with hardware virtualization support, but where no hypervisor is in use.

Other variants exploit the System Management Mode (SMM)[5] in Intel processors that is used for low-level hardware control, or the BIOS code used when the processor first boots. Such code has direct access to attached hardware devices, and is generally invisible to code running outside these special modes [EMBL08].

[5]The System Management Mode (SMM) is a relatively obscure mode on Intel processors used for low-level hardware control, with its own private memory space and execution environment, that is generally invisible to code running outside (e.g., in the operating system).

To defend against these types of rootkits, the entire boot process must be secure, ensuring that the operating system is loaded and secured against the installation of these types of malicious code. This needs to include monitoring the loading of any hypervisor code to ensure it is legitimate. We will discuss this further in **Chapter 12**.

6.10 COUNTERMEASURES

We now consider possible countermeasures for malware. These are generally known as "anti-virus" mechanisms, as they were first developed to specifically target virus infections. However, they have evolved to address most of the types of malware we discuss in this chapter.

Malware Countermeasure Approaches

The ideal solution to the threat of malware is prevention: Do not allow malware to get into the system in the first place, or block the ability of it to modify the system. This goal is, in general, nearly impossible to achieve, although taking suitable countermeasures to harden systems and users in preventing infection can significantly reduce the number of successful malware attacks. NIST SP 800-83 suggests there are 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.

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. We will discuss them further in **Chapter 12** when we discuss hardening operating systems and applications.

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. NIST SP 800-83 provides examples of suitable awareness issues. We will return to this topic in **Chapter 17**.

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 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 counter- measures:

- **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, in a defense-in-depth strategy.

Detection of the presence of malware can occur in a number of locations. It may occur on the infected system, where some host-based "anti-virus" program is running, monitoring data imported into the system, and the execution and behavior 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 (IDS). 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. We now consider each of these approaches in more detail.

Host-Based Scanners and Signature-Based Anti-Virus

The first location where anti-virus software is used is on each end system. This gives the software the maximum access to information on not only the behavior of the malware as it interacts with the targeted system, but also the smallest overall view of malware activity. The use of anti-virus software on personal computers is now widespread, in part caused by the explosive growth in malware volume and activity. This software can be regarded as a form of host-based intrusion detection system, which we will discuss more generally in **Section 8.4**. Advances in virus and other malware technology, and in anti-virus 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 anti-virus software packages. As the malware arms race has evolved, both the malware code and, necessarily, anti-virus software have grown more complex and sophisticated.

[STEP93] identifies four generations of anti-virus 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, 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 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 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 malicious activity is being attempted and then to intervene. This approach uses dynamic analysis techniques, such as those we will discuss in the next sections.

Fourth-generation products are packages consisting of a variety of anti-virus 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 anti-virus approaches.

Sandbox Analysis

One method of detecting and analyzing malware involves running potentially malicious code in an emulated sandbox or on a virtual machine. These allow the code to execute in a controlled environment, where its behavior can be closely monitored without threatening the security of a real system. These environments range from sandbox emulators that simulate memory and CPU of a target system, up to full virtual machines, of the type we will discuss in **Section 12.8**, that replicate the full functionality of target systems, but which can easily be restored to a known state. Running potentially malicious software in such environments enables the detection of complex encrypted, polymorphic, or metamorphic malware. The code must transform itself into the required machine instructions, which it then executes to perform the intended malicious actions. The resulting unpacked, transformed, or decrypted code can then be scanned for known malware signatures, or its behavior monitored as execution continues for possibly malicious activity [EGEL12, KERA16]. This extended analysis can be used to develop anti-virus signatures for new, unknown malware.

The most difficult design issue with sandbox analysis is to determine how long to run each interpretation. Typically, malware elements are activated soon after a program begins executing, but recent malware increasingly uses evasion approaches such as extended sleep to evade detection in the analysis time used by sandbox systems [KERA16]. The longer the scanner emulates a particular program, the more likely it is to catch any hidden malware. However, the sandbox analysis has only a limited amount of time and resources available, given the need to analyze large amounts of potential malware.

As analysis techniques improve, an arms race has developed between malware authors and defenders. Some malware checks to see if it is running in a sandbox or virtualized environment, and suppresses malicious behavior if so. Other malware includes extended sleep periods before engaging in malicious activity, in an attempt to evade detection before the analysis terminates. Or the malware may include a logic bomb looking for a specific date, or specific system type or network location before engaging in malicious activity, which the sandbox environment does not match. In response, analysts adapt their sandbox environments to attempt to evade these tests. This race continues.

Host-Based Dynamic Malware Analysis

Unlike heuristics or fingerprint-based scanners, dynamic malware analysis or behavior-blocking software integrates with the operating system of a host computer and monitors program behavior in real time for malicious actions [CONR02, EGEL12]. It is a type of host-based intrusion prevention system, which we will discuss further in **Section 9.6**. This software monitors the behavior of possibly malicious code, looking for potentially malicious actions, similar to the sandbox systems we discussed in the previous section. However, it then has the capability to block malicious actions before they can affect the target 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 dynamic analysis software can block suspicious software in real time, it has an advantage over such established anti-virus 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 well-defined 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.

Dynamic analysis 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 worse.

Spyware Detection and Removal

Although general anti-virus 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 anti-virus 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 it is undetectable.

Countering rootkits requires a variety of network- and computer-level security tools. Both network-based and host-based IDSs can look for the code signatures of known rootkit attacks in incoming traffic. Host-based anti-virus 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, anti-virus 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 anti-virus 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 anti-virus 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. These monitors can use either anomaly or signature and heuristic approaches to detect malware traffic, as we will discuss further in **Chapter 8**. A honeypot 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 installed in one device. 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. This monitoring could look for the common sequential or random scanning behavior used by worms and rate limit or block it. It may also be able to detect and respond to abnormally high e-mail traffic such as that used by mass e-mail worms, or spam payloads. It may also implement data exfiltration “data-loss” technical counter measures, monitoring for unauthorized transmission of sensitive information out of the organization.

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.

Distributed Intelligence Gathering Approaches

The final location where anti-virus 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. This is a specific example of a distributed intrusion prevention system (IPS), targeting malware, which we will discuss further in **Section 9.6**.

6.11 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

advanced persistent threat
adware
attack kit
backdoor
blended attack
boot-sector infector
bot
botnet
crimeware
data exfiltration
downloader
drive-by-download
e-mail virus
infection vector
keyloggers
logic bomb
macro virus
malicious software
malware
metamorphic virus
mobile code
parasitic virus
payload
phishing
polymorphic virus
propagate
ransomware
rootkit
scanning
spear-phishing
spyware
stealth virus

trapdoor
Trojan horse
virus
watering-hole attack
worm
zombie
zero-day exploit

Review Questions

6.1 What are three broad mechanisms that malware can use to propagate?
6.2 What are four broad categories of payloads that malware may carry?
6.3 What characteristics of an advanced persistent threat give it that name?
6.4 What are typical phases of operation of a virus or worm?
6.5 What mechanisms can a virus use to conceal itself?
6.6 What is the difference between machine executable and macro viruses?
6.7 What means can a worm use to access remote systems to propagate?
6.8 What is a “drive-by-download” and how does it differ from a worm?
6.9 How does a Trojan enable malware to propagate? How common are Trojans on computer systems? Or on mobile platforms?
6.10 What is a “logic bomb”?
6.11 What is the difference between a backdoor, a bot, a keylogger, spyware, and a rootkit? Can they all be present in the same malware?
6.12 What is the difference between a “phishing” attack and a “spear-phishing” attack, particularly in terms of who the target may be?
6.13 List some the different levels in a system that a rootkit may use.
6.14 Describe some malware countermeasure elements.
6.15 List three places malware mitigation mechanisms may be located.
6.16 Briefly describe the four generations of anti-virus software.

Problems

6.1 A computer virus places a copy of itself into other programs, and arranges for that code to be run when the program executes. The “simple” approach just appends the code after the existing code, and changes the address where code execution starts. This will clearly increase the size of the program, which is easily observed. Investigate and briefly list some other approaches that do not change the size of the program.
6.2 The question arises as to whether it is possible to develop a program that can analyze a piece of software to determine if it is a virus. Consider that we have a program D that is supposed to be able to do that. That is, for any program P, if we run D(P), the result returned is TRUE (P is a virus) or FALSE (P is not a virus). Now consider the following

program:

```
Program CV :=
  {. . .
  main-program :=
       {if D(CV) then goto next:
          else infect-executable;
       }
next:
  }
```

In the preceding program, infect-executable is a module that scans memory for executable programs and replicates itself in those programs. Determine if D can correctly decide whether CV is a virus.

6.3 The following code fragments show a sequence of virus instructions and a metamorphic version of the virus. Describe the effect produced by the metamorphic code.

Original Code	Metamorphic Code
`mov eax, 5`	`mov eax, 5`
`add eax, ebx`	`push ecx`
`call [eax]`	`pop ecx`
	`add eax, ebx`
	`swap eax, ebx`
	`swap ebx, eax`
	`call [eax]`
	`nop`

6.4 The list of passwords used by the Morris worm is provided at this book's website.

a. The assumption has been expressed by many people that this list represents words commonly used as passwords. Does this seem likely? Justify your answer.
b. If the list does not reflect commonly used passwords, suggest some approaches that Morris may have used to construct the list.

6.5 Consider the following fragment:

```
legitimate code
```

```
    if data is Friday the 13th;
       crash_computer();
    legitimate code
```

What type of malware is this?
6.6 Consider the following fragment in an authentication program:

```
    username = read_username();
    password = read_password();
    if username is "133t h4ck0r"
          return ALLOW_LOGIN;
    if username and password are valid
          return ALLOW_LOGIN
    else return DENY_LOGIN
```

What type of malicious software is this?
6.7 Assume you have found a USB memory stick in your work parking area. What threats might this pose to your work computer should you just plug the memory stick in and examine its contents? In particular, consider whether each of the malware propagation mechanisms we discuss could use such a memory stick for transport. What steps could you take to mitigate these threats, and safely determine the contents of the memory stick?
6.8 Suppose you observe that your home PC is responding very slowly to information requests from the net. And then you further observe that your network gateway shows high levels of network activity, even though you have closed your e-mail client, Web browser, and other programs that access the net. What types of malware could cause these symptoms? Discuss how the malware might have gained access to your system. What steps can you take to check whether this has occurred? If you do identify malware on your PC, how can you restore it to safe operation?
6.9 Suppose while trying to access a collection of short videos on some website, you see a pop-up window stating that you need to install this custom codec in order to view the videos. What threat might this pose to your computer system if you approve this installation request?
6.10 Suppose you have a new smartphone and are excited about the range of apps available for it. You read about a really interesting new game that is available for your phone. You do a quick Web search for it and see that a version is available from one of the free marketplaces. When you download and start to install this app, you are asked to approve the access permissions granted to it. You see that it wants permission to "Send SMS messages" and to "Access your address-book". Should you be suspicious that a game wants these types of permissions? What threat might the app pose to your smartphone, should you grant these permissions and proceed to install it? What types of malware might it be?
6.11 Assume you receive an e-mail, which appears to come from a senior manager in your

company, with a subject indicating that it concerns a project that you are currently working on. When you view the e-mail, you see that it asks you to review the attached revised press release, supplied as a PDF document, to check that all details are correct before management releases it. When you attempt to open the PDF, the viewer pops up a dialog labeled “Launch File” indicating that “the file and its viewer application are set to be launched by this PDF file.” In the section of this dialog labeled “File,” there are a number of blank lines, and finally the text “Click the ‘Open’ button to view this document.” You also note that there is a vertical scroll-bar visible for this region. What type of threat might this pose to your computer system should you indeed select the “Open” button? How could you check your suspicions without threatening your system? What type of attack is this type of message associated with? How many people are likely to have received this particular e-mail?

6.12 Assume you receive an e-mail, which appears to come from your bank, includes your bank logo in it, and with the following contents:

“Dear Customer, Our records show that your Internet Banking access has been blocked due to too many login attempts with invalid information such as incorrect access number, password, or security number. We urge you to restore your account access immediately, and avoid permanent closure of your account, by clicking on this *link to restore your account*. Thank you from your customer service team.”

What form of attack is this e-mail attempting? What is the most likely mechanism used to distribute this e-mail? How should you respond to such e-mails?

6.13 Suppose you receive a letter from a finance company stating that your loan payments are in arrears, and that action is required to correct this. However, as far as you know, you have never applied for, or received, a loan from this company! What may have occurred that led to this loan being created? What type of malware, and on which computer systems, might have provided the necessary information to an attacker that enabled them to successfully obtain this loan?

6.14 List the types of attacks on a personal computer that each of a (host-based) personal firewall, and anti-virus software, can help you protect against. Which of these counter-measures would help block the spread of macro viruses spread using e-mail attachments? Which would block the use of backdoors on the system?