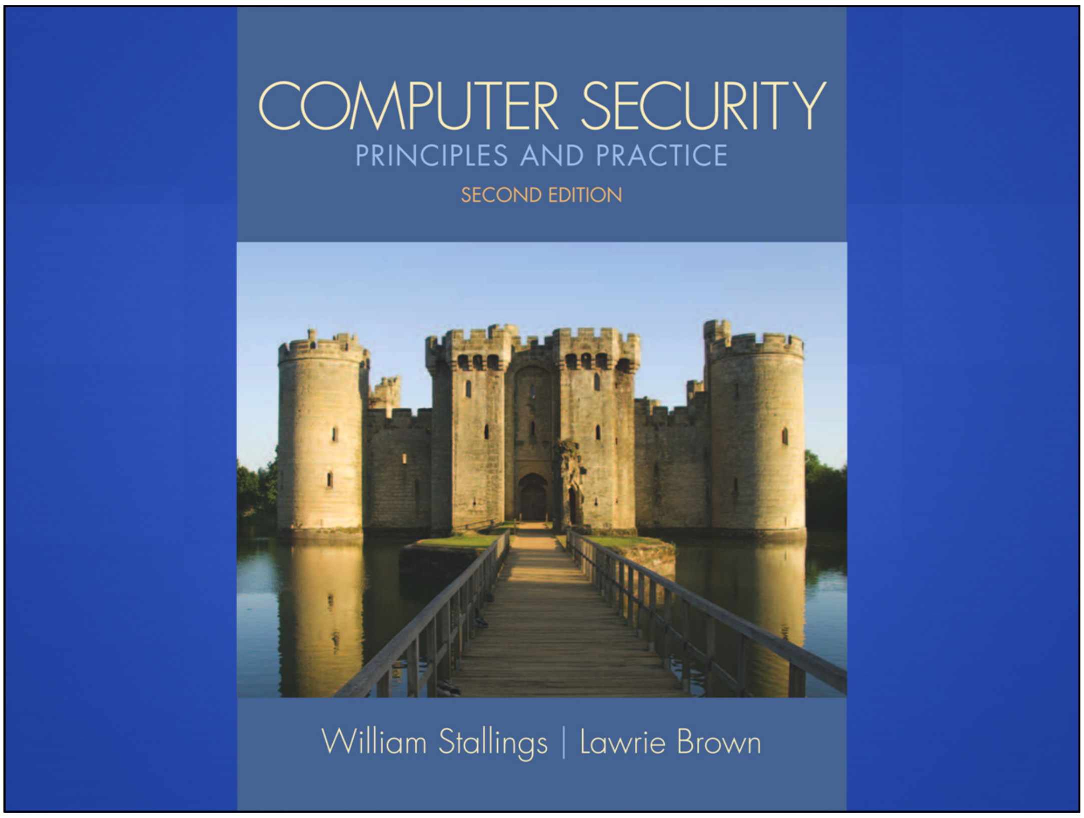
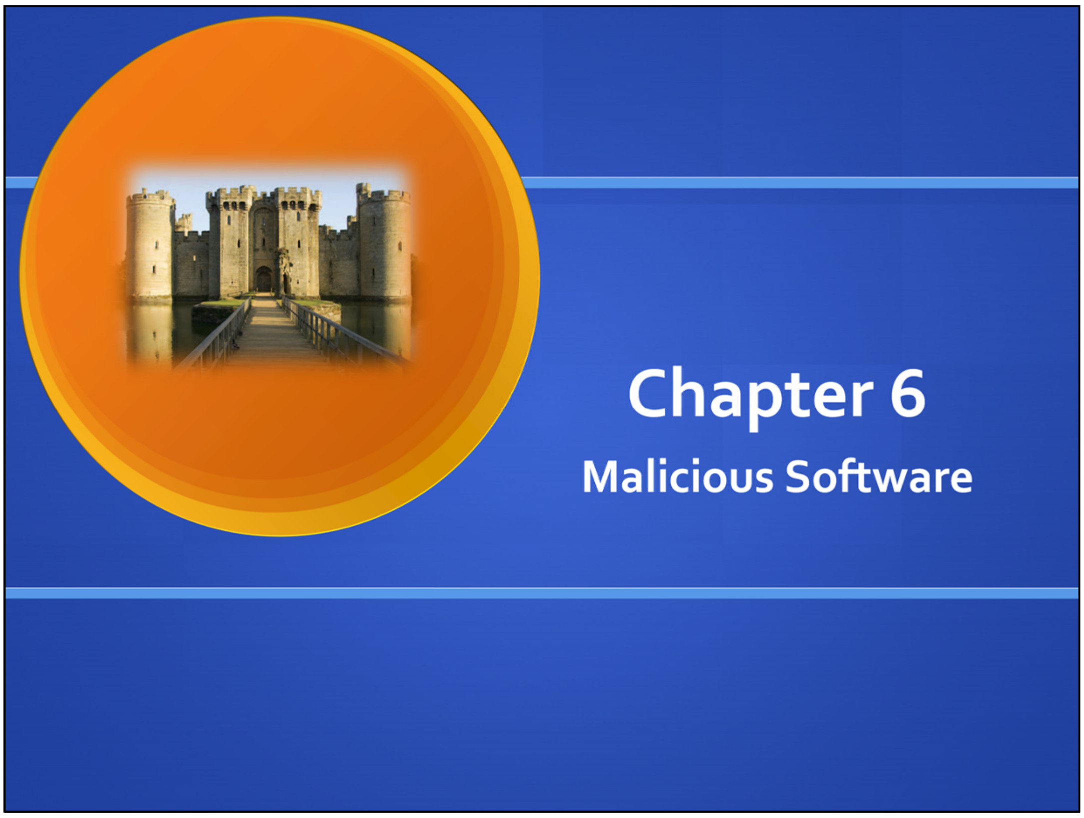


Lecture slides prepared for “Computer Security: Principles and Practice”, 2/e, by William Stallings and Lawrie Brown, Chapter 6 “Malicious Software”.



Chapter 6

Malicious Software

This chapter examines the wide spectrum of malware threats and countermeasures.

We begin with a survey of various types of malware, and offer a broad

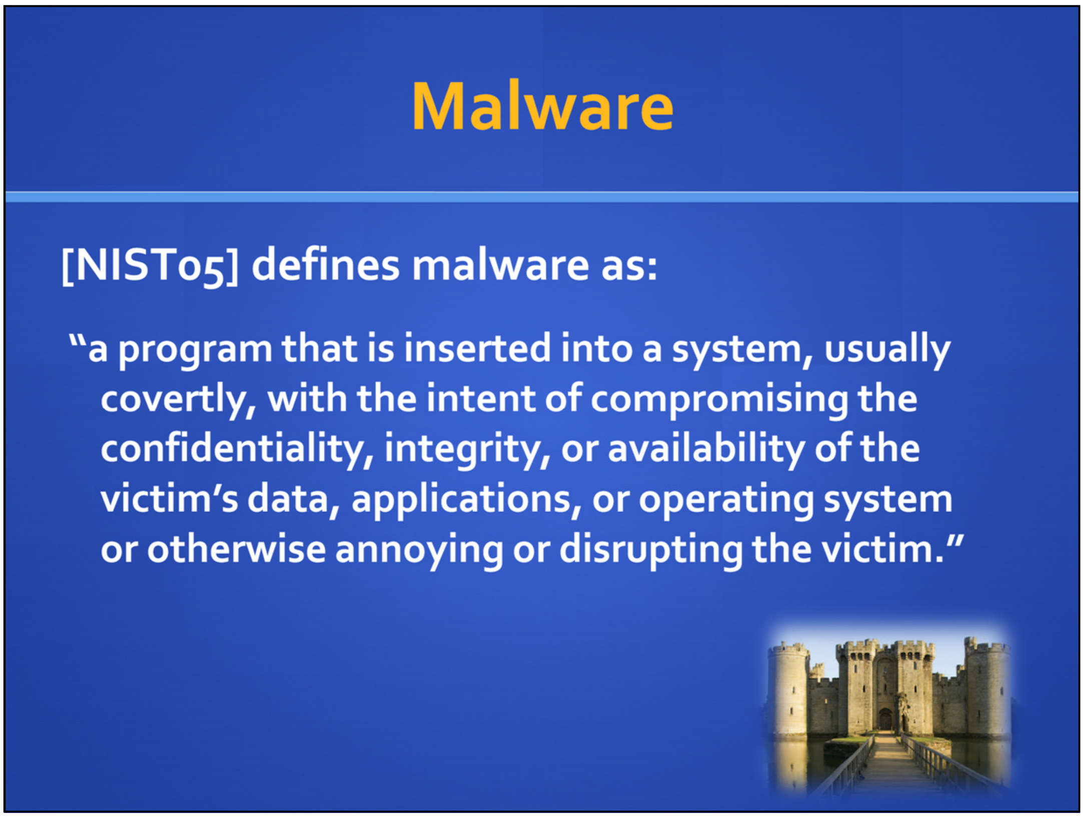
classification based first on the means malware uses to spread or **propagate , and**

then on the variety of actions or **payloads used once the malware has reached a**

target. Propagation mechanisms include those used by viruses, worms, and Trojans.

Payloads include system corruption, bots, phishing, spyware, and rootkits. The

discussion concludes with a review of countermeasure approaches.



Malware

[NIST05] defines malware as:

"a program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or otherwise annoying or disrupting the victim."

Malicious software , or malware , arguably constitutes one of the most significant categories

of threats to computer systems. [NIST05] defines malware as "a program that

is inserted into a system, usually covertly, with the intent of compromising the confidentiality,

integrity, or availability of the victim's data, applications, or operating

system or otherwise annoying or disrupting the victim." Hence, we are concerned

with the threat malware poses to application programs, to utility programs, such as

editors and compilers, and to kernel-level programs. We are also concerned with

its use on compromised or malicious Web sites and servers, or in especially crafted

spam e-mails or other messages, which aim to trick users into revealing sensitive

personal information.

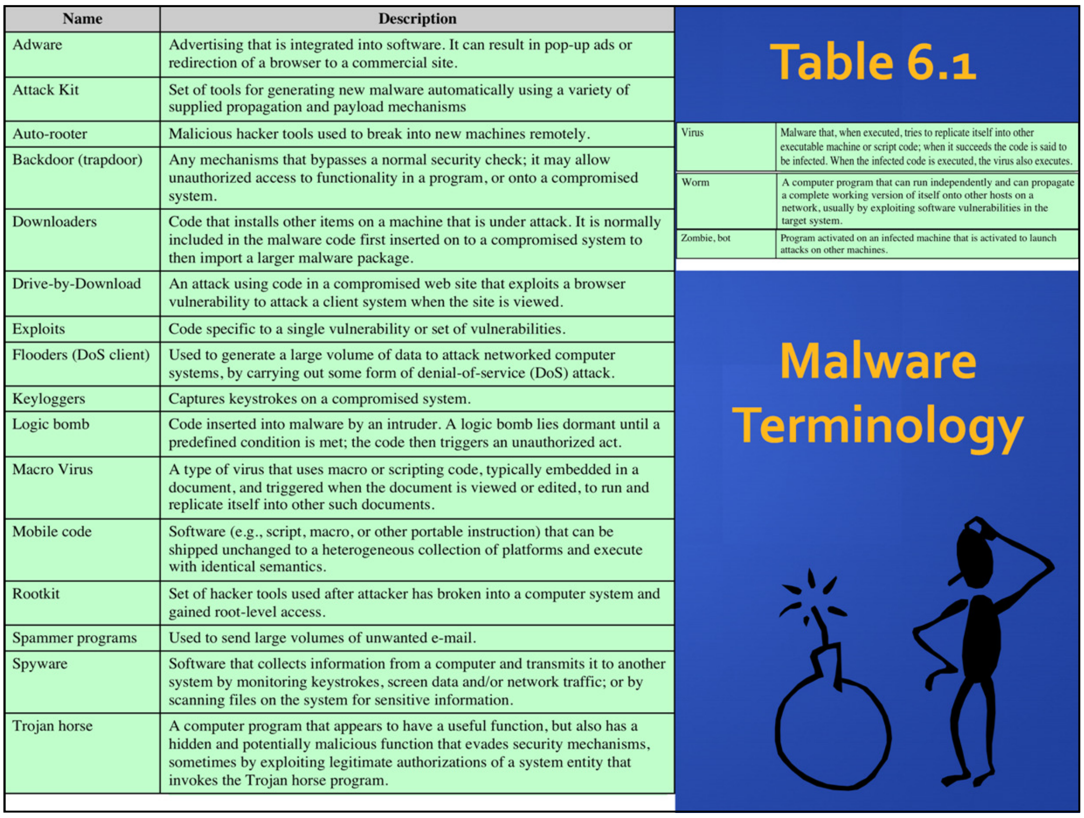


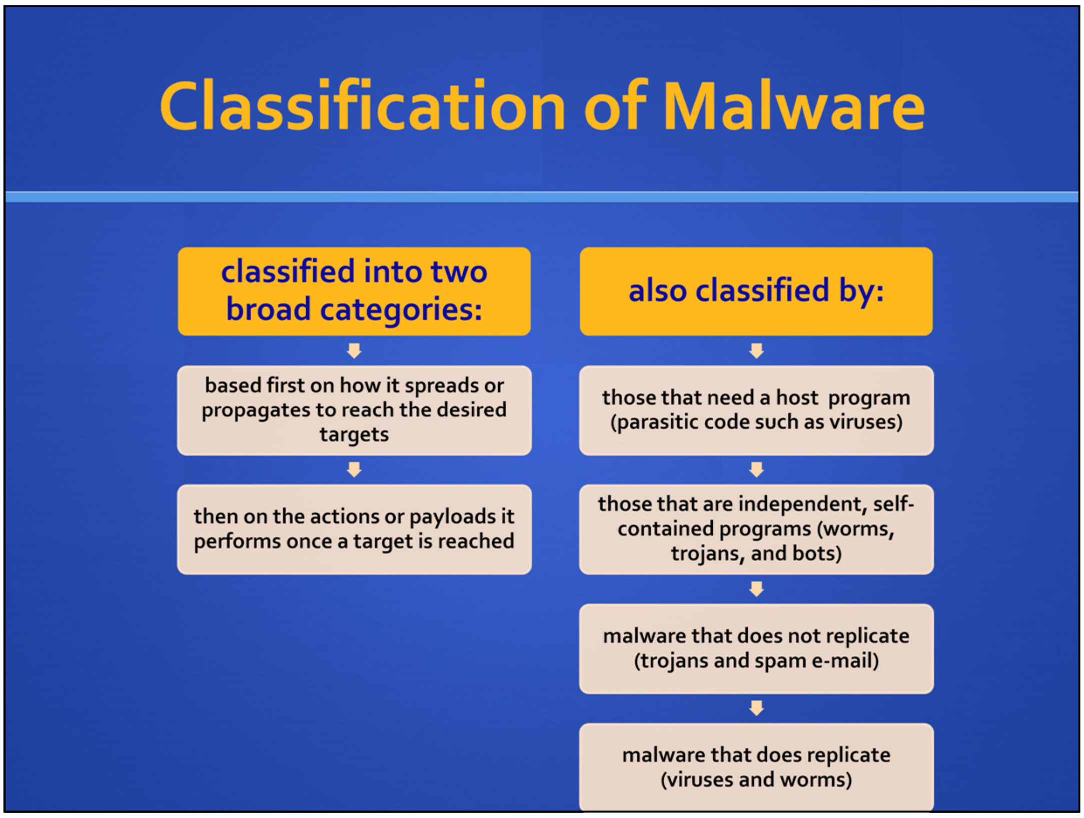
Table 6.1

Malware Terminology

Name	Description
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 mechanisms 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 in a compromised web site 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 an unauthorized act.
Macro Virus	A type of virus that uses macro or scripting code, typically embedded in a document, and triggered when the document is viewed or edited, to run and replicate itself into other such documents.
Mobile code	Software (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics.
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 the Trojan horse program.
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, usually by exploiting software vulnerabilities in the target system.
Zombie, bot	Program activated on an infected machine that is activated to launch attacks on other machines.

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.



Classification of Malware

classified 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

also classified by:

those that need a host program (parasitic code such as viruses)

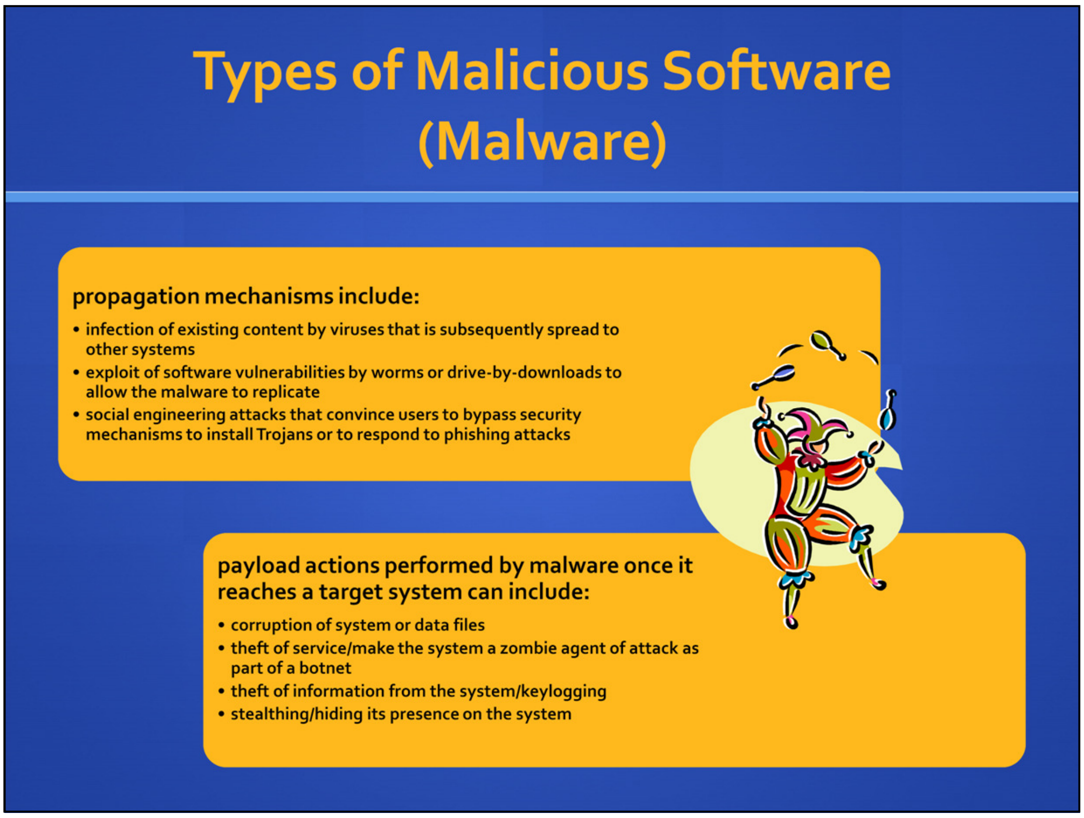
those that are independent, self-contained programs (worms, trojans, and bots)

malware that does not replicate (trojans and spam e-mail)

malware that does replicate (viruses and worms)

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; and then on the actions or payloads it performs
once a target is reached.

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.



Types of Malicious Software (Malware)

propagation mechanisms include:

- infection of existing content by viruses that is subsequently spread to other systems
- exploit of software vulnerabilities by worms or drive-by-downloads to allow the malware to replicate
- social engineering attacks that convince users to bypass security mechanisms to install Trojans or to respond to phishing attacks

payload actions performed by malware once it reaches a target system can include:

- corruption of system or data files
- theft of service/make the system a zombie agent of attack as part of a botnet
- theft of information from the system/keylogging
- stealthing/hiding its presence on the system

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.

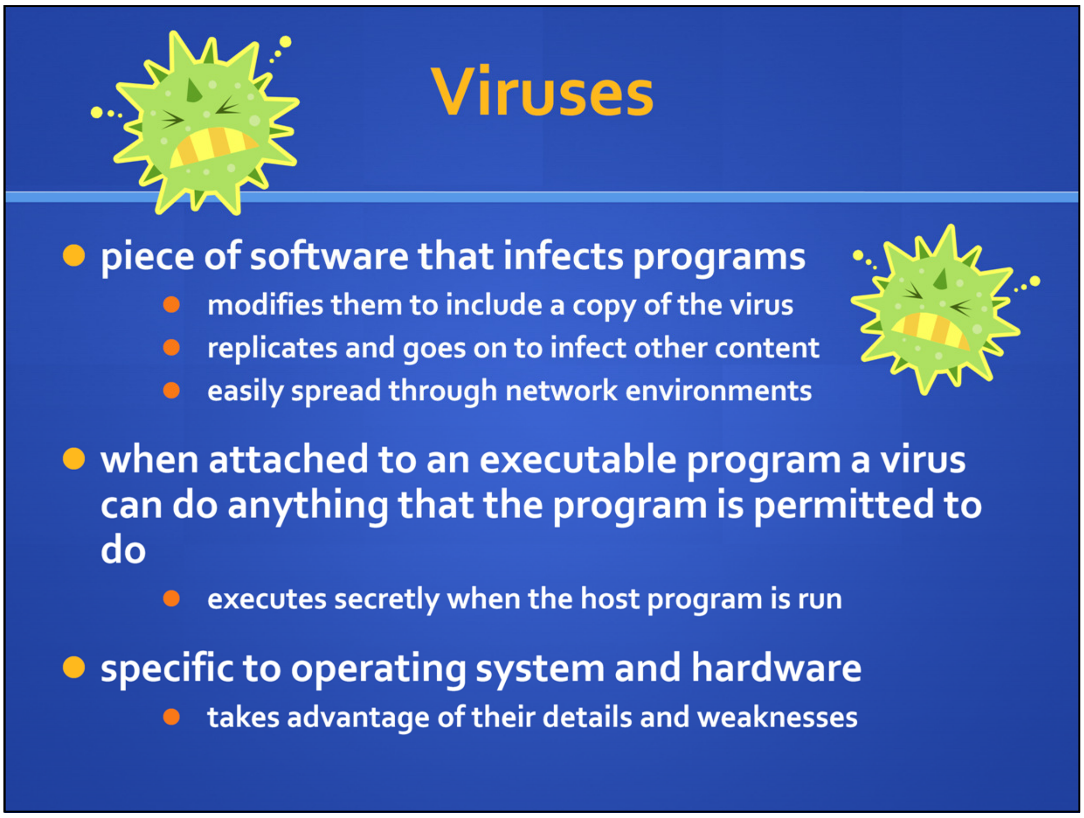
Earlier approaches to malware classification distinguished between those that need a host program, being parasitic code such as viruses, and those that are independent, self-contained programs run on the system such as worms, trojans, and bots. Another distinction used was between malware that does not 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, and
then follow with a discussion of appropriate countermeasures.



Viruses

- piece of software that infects programs
  - modifies them to include a copy of the virus
  - replicates and goes on to infect other content
  - easily spread through network environments
- when attached to an executable program a virus can do anything that the program is permitted to do
  - executes secretly when the host program is run
- specific to operating system and hardware
  - takes advantage of their details and weaknesses

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

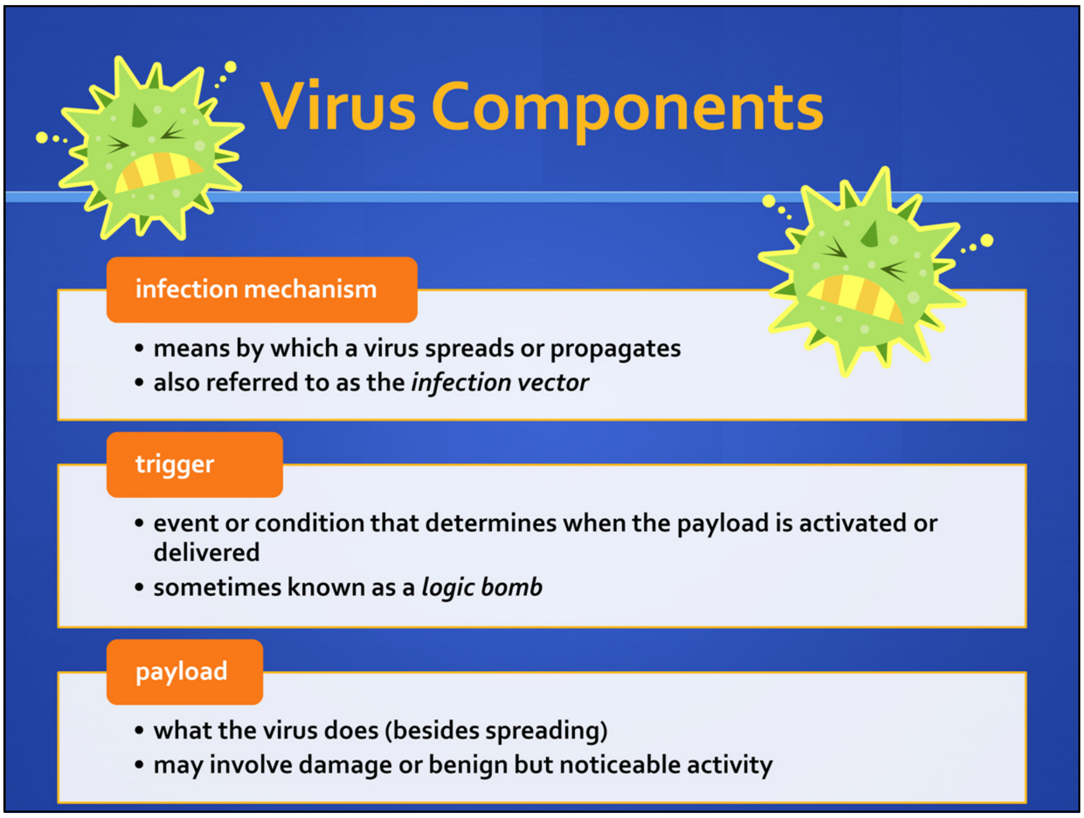
A computer virus is a piece of software that can “infect” other programs, or
indeed
any type of executable content, by modifying them. The modification includes
injecting the original code with a routine to make copies of the virus code, which
can then go on to infect other content. Computer viruses 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.



Virus Components

infection mechanism

- means by which a virus spreads or propagates
- also referred to as the *infection vector*

trigger

- event or condition that determines when the payload is activated or delivered
- sometimes known as a *logic bomb*

payload

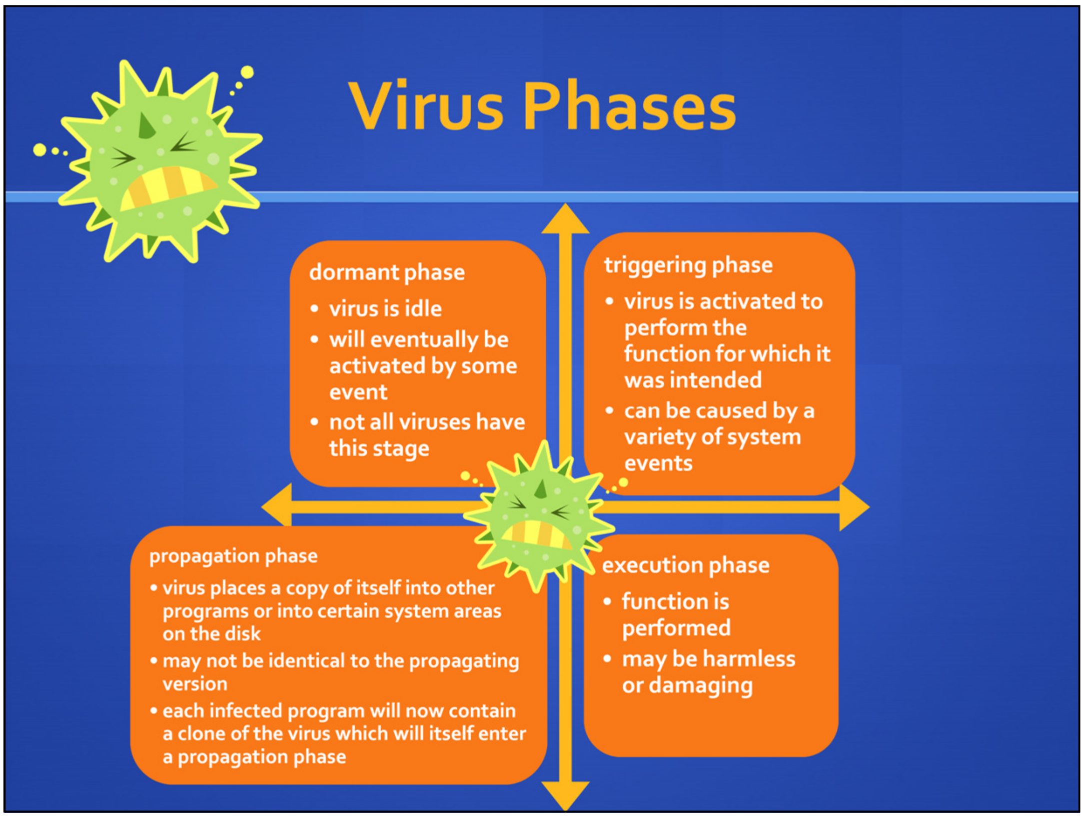
- what the virus does (besides spreading)
- may involve damage or benign but noticeable activity

[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.



Virus Phases

dormant phase
- virus is idle
- will eventually be activated by some event
- not all viruses have this stage

triggering phase
- virus is activated to perform the function for which it was intended
- can be caused by a variety of system events

propagation phase
- virus places a copy of itself into other programs or into certain system areas on the disk
- may not be identical to the propagating version
- each infected program will now contain a clone of the virus which will itself enter a propagation phase

execution phase
- function is performed
- may be harmless or damaging

During its lifetime, a typical virus goes through the following four phases:

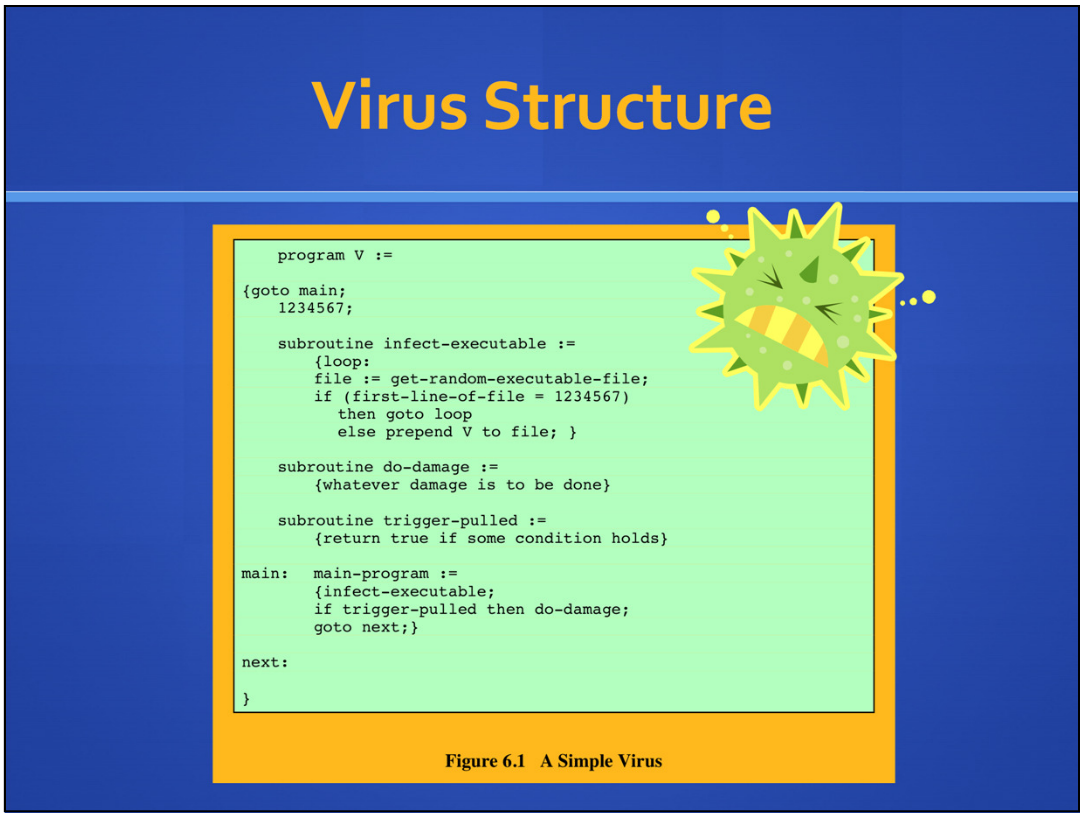
• **Dormant phase: The virus is idle. The virus will eventually be activated by** some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.

• **Propagation phase: The virus places a copy of itself into other programs or**
into certain system areas on the disk. The copy may not be identical to the propagating version; viruses often morph to evade detection. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.

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

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

Most viruses that infect executable program files carry out their work in a manner that is specific to a particular operating system and, in some cases, specific
to a particular hardware platform. Thus, they are designed to take advantage of the
details and weaknesses of particular systems. Macro viruses though, target specific
document types, which are often supported on a variety of systems.



Virus Structure

```
    program V :=

{goto main;
    1234567;

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

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

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

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

next:

}
```

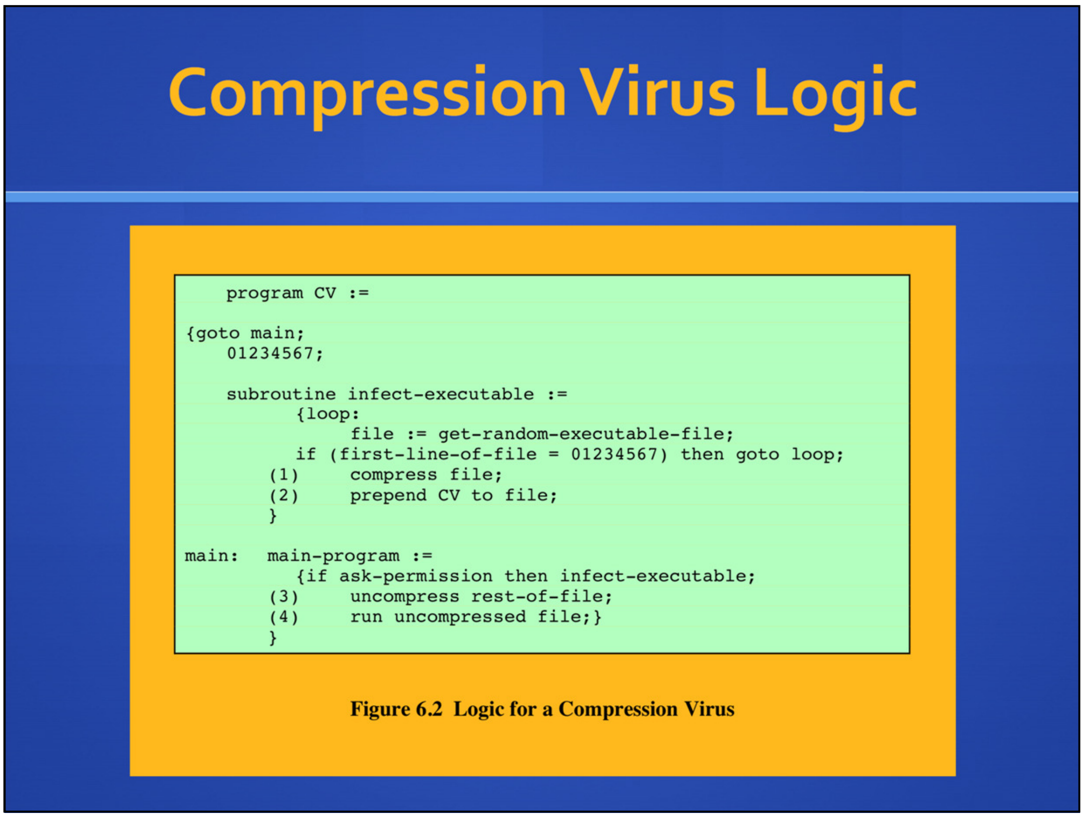
Figure 6.1 A Simple Virus

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

A very general depiction of virus structure is shown in Figure 6.1 (based on [COHE94]). In this case, the virus code, V, is prepended to infected programs, and
it is assumed that the entry point to the program, when invoked, is the first line of the program.

The infected program begins with the virus code and works as follows. The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program may first seek out uninfected executable files and infect them. Next, the virus

may
execute its payload if the required trigger conditions, if any, are met. Finally, the
virus transfers control to the original program. If the infection phase of the program
is reasonably rapid, a user is unlikely to notice any difference between the
execution
of an infected and an uninfected program.



Compression Virus Logic

```
    program CV :=

{goto main;
    01234567;

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

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

Figure 6.2 Logic for a Compression Virus

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file

so that both the infected and uninfected versions are of identical length. Figure 6.2

shows in general terms the logic required. The key lines in this virus are numbered,

and Figure 6.3 illustrates the operation.

Operation for Figure 6.2

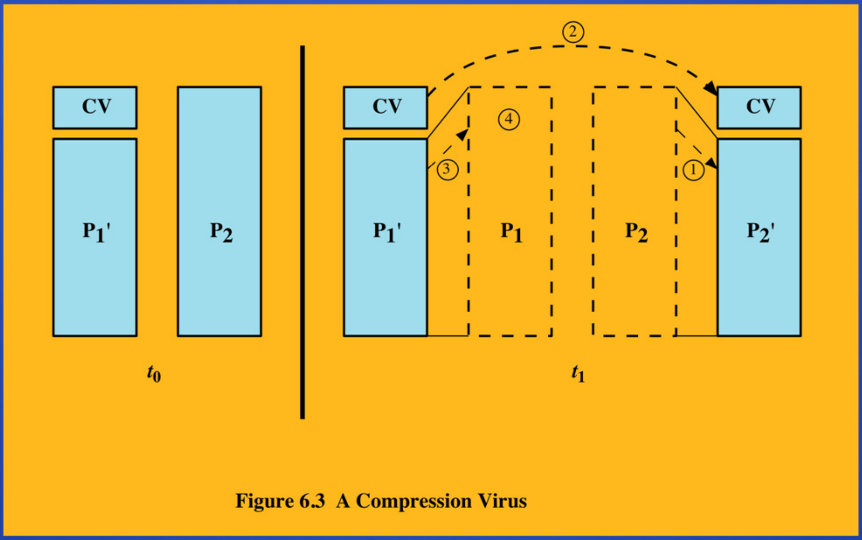


Figure 6.3 A Compression Virus

We assume that program P1 is infected with
the virus CV. When this program is invoked, control passes to its virus, which performs
the following steps:

1. For each uninfected file P_2 that is found, the virus first compresses that file to

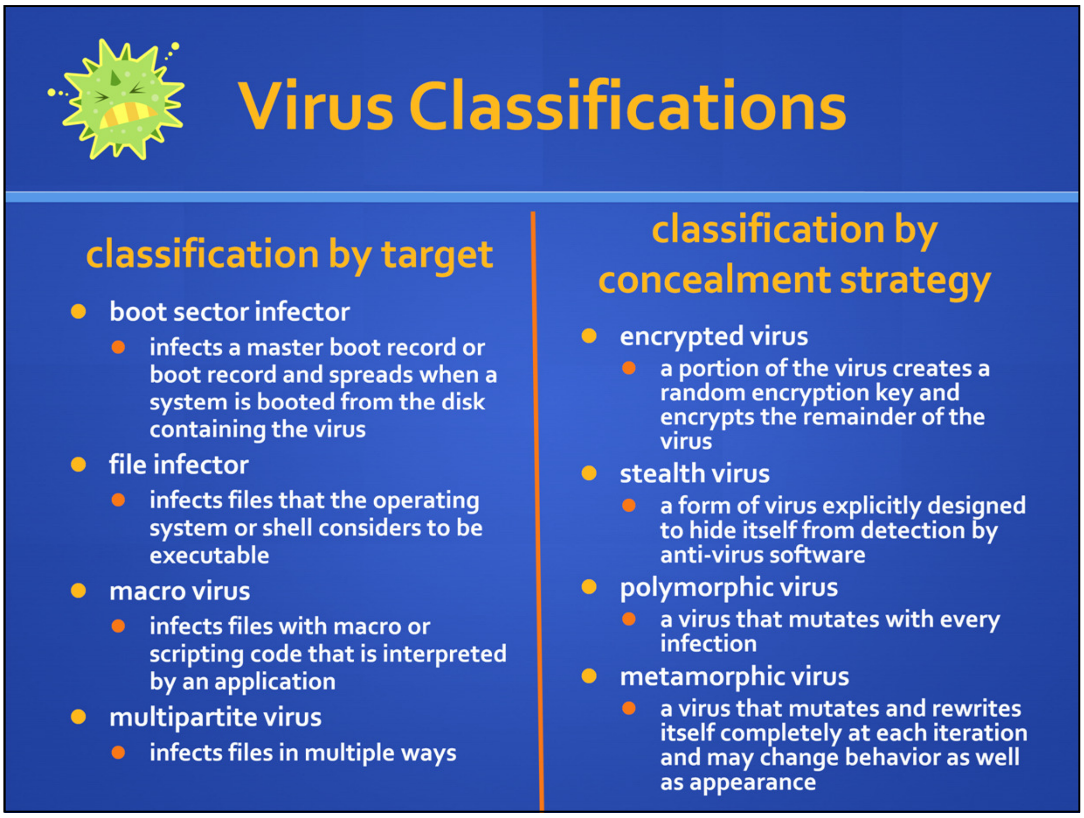
produce P'_2, which is
shorter than the original program by the size of the virus.

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

3. The compressed version of the original infected program, P'_1, is uncompressed.

4. The uncompressed original program is executed.
In this example, the virus does nothing other than propagate. As previously
mentioned, the virus may also include one or more payloads.

Once a virus has gained entry to a system by infecting a single program, it is in
a position to potentially infect some or all other executable files on that system
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.



Virus Classifications

classification by target

- 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 considers 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

classification by concealment strategy

- encrypted virus
  - a portion of the virus creates a random encryption key and encrypts the remainder of the virus
- stealth virus
  - a form of virus explicitly designed to hide itself from detection by anti-virus software
- polymorphic virus
  - a virus that mutates with every infection
- metamorphic virus
  - a virus that mutates and rewrites itself completely at each iteration and may change behavior as well as appearance

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 that virus eradication
must deal with all of the possible sites of infection.

A virus classification by concealment strategy includes the following categories:

• **Encrypted virus: A typical approach is as follows. A portion of the virus creates**
a random encryption key and encrypts the remainder of the virus. The key is
stored with the virus. When an infected program is invoked, the virus uses the
stored random key to decrypt the virus. When the virus replicates, a different
random key is selected. Because the bulk of the virus is encrypted with a different
key for each instance, there is no constant bit pattern to observe.

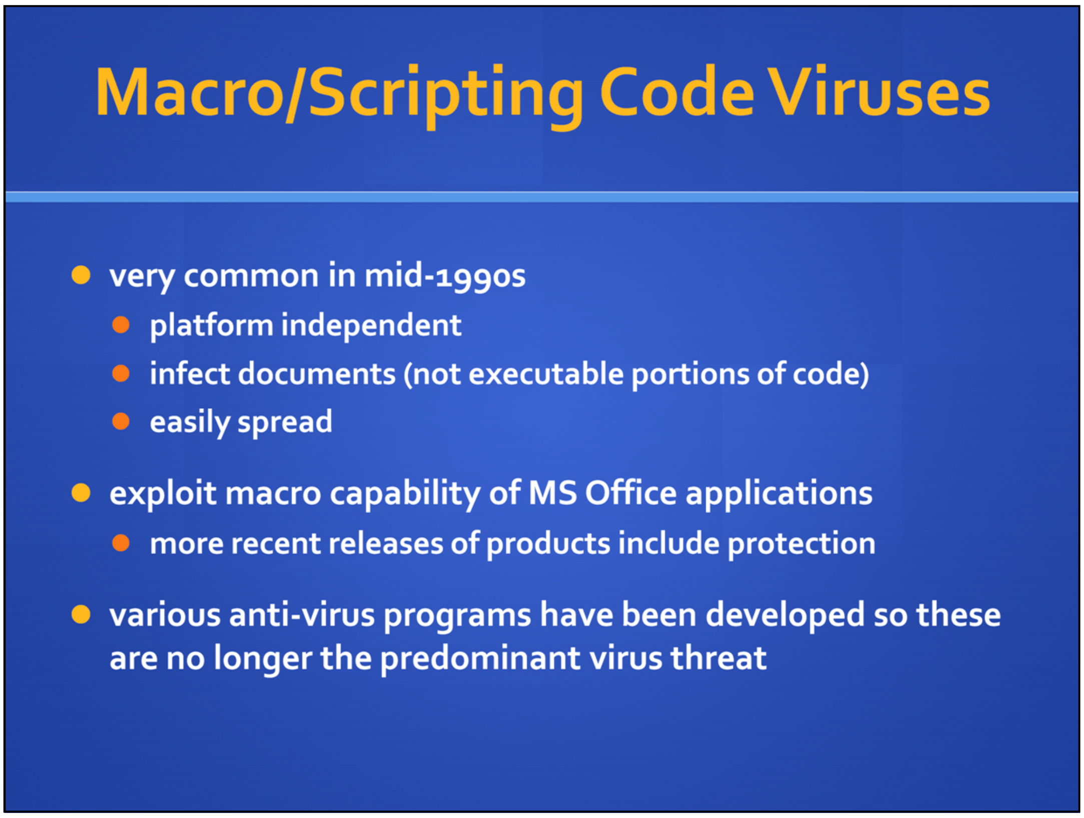
Stealth virus : A form of virus explicitly designed to hide itself from detection
by anti-virus software. Thus, the entire virus, not just a payload is hidden. It
may use both code mutation, for example compression, and rootkit techniques
to achieve this.

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

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

A **polymorphic virus 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.



Macro/Scripting Code Viruses

- very common in mid-1990s
  - platform independent
  - infect documents (not executable portions of code)
  - easily spread
- exploit macro capability of MS Office applications
  - more recent releases of products include protection
- various anti-virus programs have been developed so these are no longer the predominant virus threat

In the mid-1990s, macro or scripting code viruses became by far the most prevalent
type of virus. 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

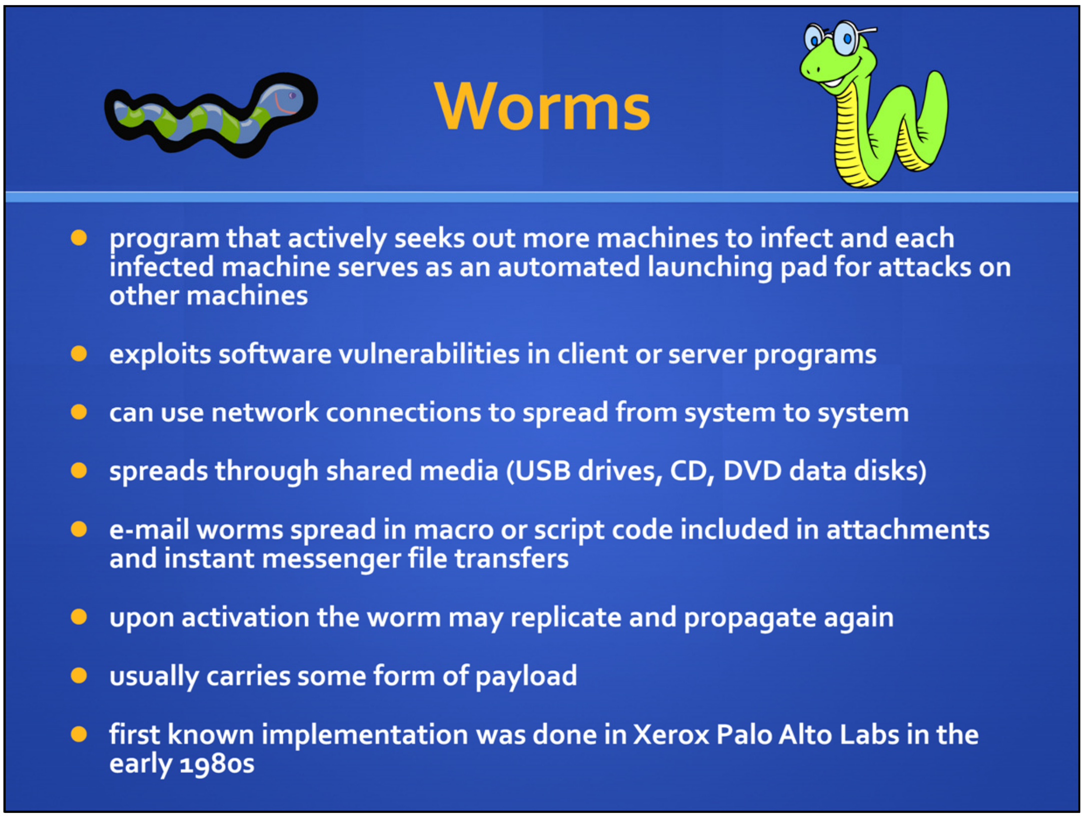
normal use. A very common method is by electronic mail.

4. Because macro viruses infect user documents rather than system programs,
traditional file system access controls are of limited use in preventing their spread, since users are expected to modify them.

Macro viruses take advantage of support for active content using a scripting or macro language, embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes.
They are also used to support dynamic content, form validation, and other useful tasks associated with these documents.

Successive releases of MS Office products provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various anti-virus product vendors have also developed tools to detect and remove macro viruses. As in other types of
viruses, the arms race continues in the field of macro viruses, but they no longer are
the predominant virus threat.

Another possible host for macro virus–style malware is in Adobe’s PDF documents.
These can support a range of embedded components, including Javascript and other types of scripting code. Although recent PDF viewers include measures to
warn users when such code is run, the message the user is shown can be manipulated
to trick them into permitting its execution. If this occurs, the code could potentially act as a virus to infect other PDF documents the user can access on their system. Alternatively, it can install a Trojan, or act as a worm, as we discuss later [STEV11].

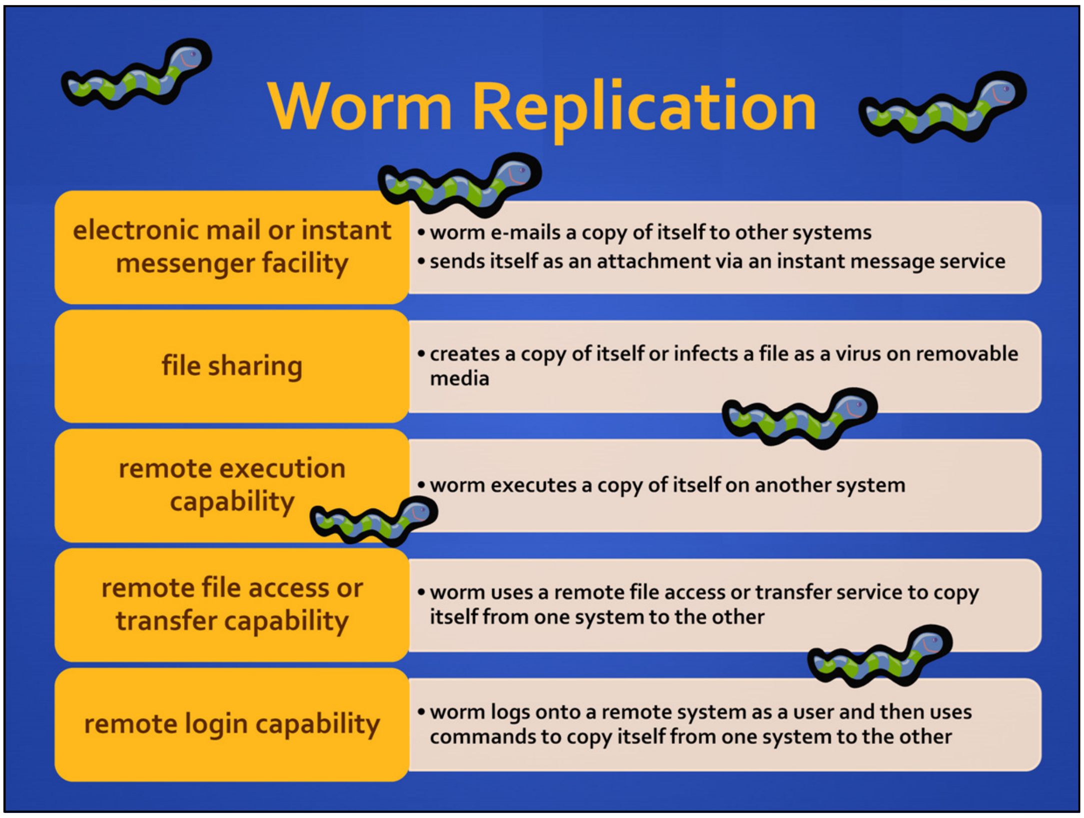


Worms

- program that actively seeks out more machines to infect and each infected machine serves as an automated launching pad for attacks on other machines
- exploits software vulnerabilities in client or server programs
- can use network connections to spread from system to system
- spreads through shared media (USB drives, CD, DVD data disks)
- e-mail worms spread in macro or script code included in attachments and instant messenger file transfers
- upon activation the worm may replicate and propagate again
- usually carries some form of payload
- first known implementation was done in Xerox Palo Alto Labs in the early 1980s

The next category of malware propagation concerns the exploit of software
vulnerabilities, such as those we discuss in Chapters 10 and 11 , which are commonly
exploited by computer worms. A worm is a program that actively seeks out
more machines to infect, and then each infected machine serves as an automated
launching pad for attacks on other machines. Worm programs exploit software
vulnerabilities in client or server programs to gain access to each new system. They
can use network connections to spread from system to system. They can also spread
through shared media, such as USB drives or CD and DVD data disks. E-mail
worms spread in macro or script code included in documents attached to e-mail or
to instant messenger file transfers. Upon activation, the worm may replicate and
propagate again. In addition to propagation, the worm usually carries some form of
payload, such as those we discuss later.

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.



Worm Replication

electronic mail or instant messenger facility	• worm e-mails a copy of itself to other systems • sends itself as an attachment via an instant message service
file sharing	• creates a copy of itself or infects a file as a virus on removable media
remote execution capability	• worm executes a copy of itself on another system
remote file access or transfer capability	• worm uses a remote file access or transfer service to copy itself from one system to the other
remote login capability	• worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other

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 [SYMA11]:

• **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 of 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 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 to other systems to infect by examining
host tables, address books, buddy lists, trusted peers, and other similar repositories of remote system access details; by scanning possible target host addresses; or by searching for suitable removable media devices to use.

• Use the access mechanisms found to transfer a copy of itself to the remote system, and cause the copy to be run.

The worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it can also disguise its presence by naming itself as a system process or using some
other name that may not be noticed by a system operator. More recent worms can even inject their code into existing processes on the system, and run using additional
threads in that process, to further disguise their presence.

Worm Propagation Model

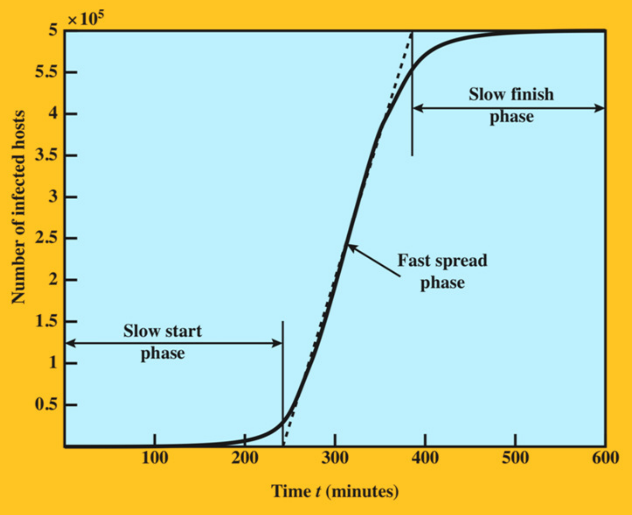
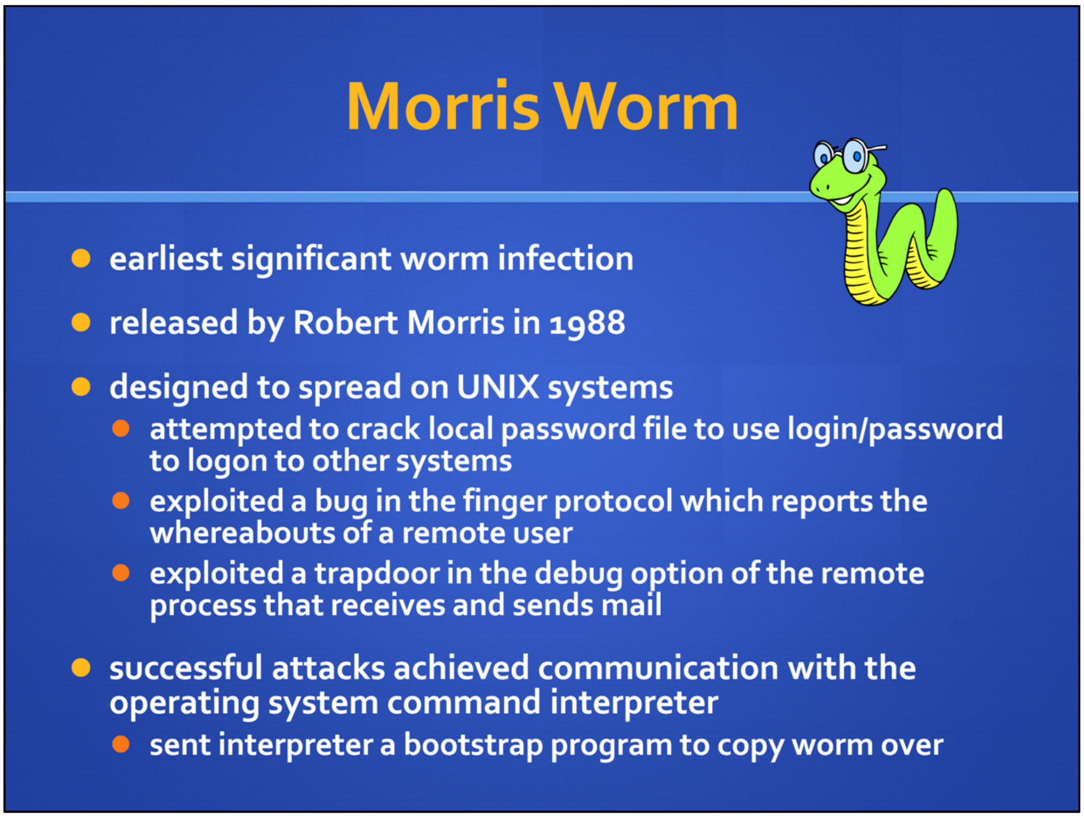


Figure 6.4 Worm Propagation Model

[ZOU05] describes 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. Figure 6.4 shows the dynamics for one typical set of parameters. 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.

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.



Morris Worm

- earliest significant worm infection
- released by Robert Morris in 1988
- designed to spread on UNIX systems
  - attempted to crack local password file to use login/password to logon to other systems
  - exploited a bug in the finger protocol which reports the whereabouts of a remote user
  - exploited a trapdoor in the debug option of the remote process that receives and sends mail
- successful attacks achieved communication with the operating system command interpreter
  - sent interpreter a bootstrap program to copy worm over

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 and then used the discovered
passwords and corresponding user IDs. The assumption was that many users

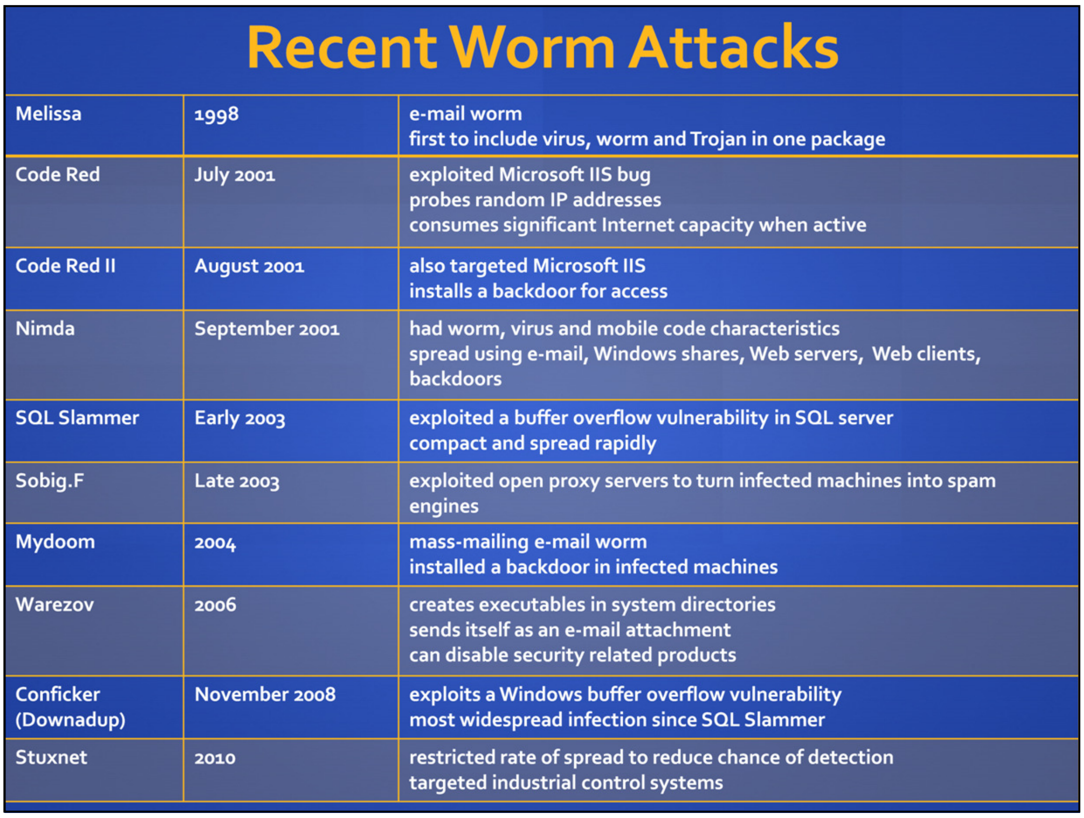
would
use the same password on different systems. To obtain the passwords, the worm ran a password-cracking program that tried

a. Each user's account name and simple permutations of it
b. A list of 432 built-in passwords that Morris thought to be likely candidates
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.



Recent Worm Attacks

Melissa	1998	e-mail worm first to include virus, worm and Trojan in one package
Code Red	July 2001	exploited Microsoft IIS bug probes random IP addresses consumes significant Internet capacity when active
Code Red II	August 2001	also targeted Microsoft IIS installs a backdoor for access
Nimda	September 2001	had worm, virus and mobile code characteristics spread using e-mail, Windows shares, Web servers, Web clients, backdoors
SQL Slammer	Early 2003	exploited a buffer overflow vulnerability in SQL server compact and spread rapidly
Sobig.F	Late 2003	exploited open proxy servers to turn infected machines into spam engines
Mydoom	2004	mass-mailing e-mail worm installed a backdoor in infected machines
Warezov	2006	creates executables in system directories sends itself as an e-mail attachment can disable security related products
Conficker (Downadup)	November 2008	exploits a Windows buffer overflow vulnerability most widespread infection since SQL Slammer
Stuxnet	2010	restricted rate of spread to reduce chance of detection targeted industrial control systems

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. If the

recipient opens the e-mail attachment, the Word macro is activated. Then it

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 Web site 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 and 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
a 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.

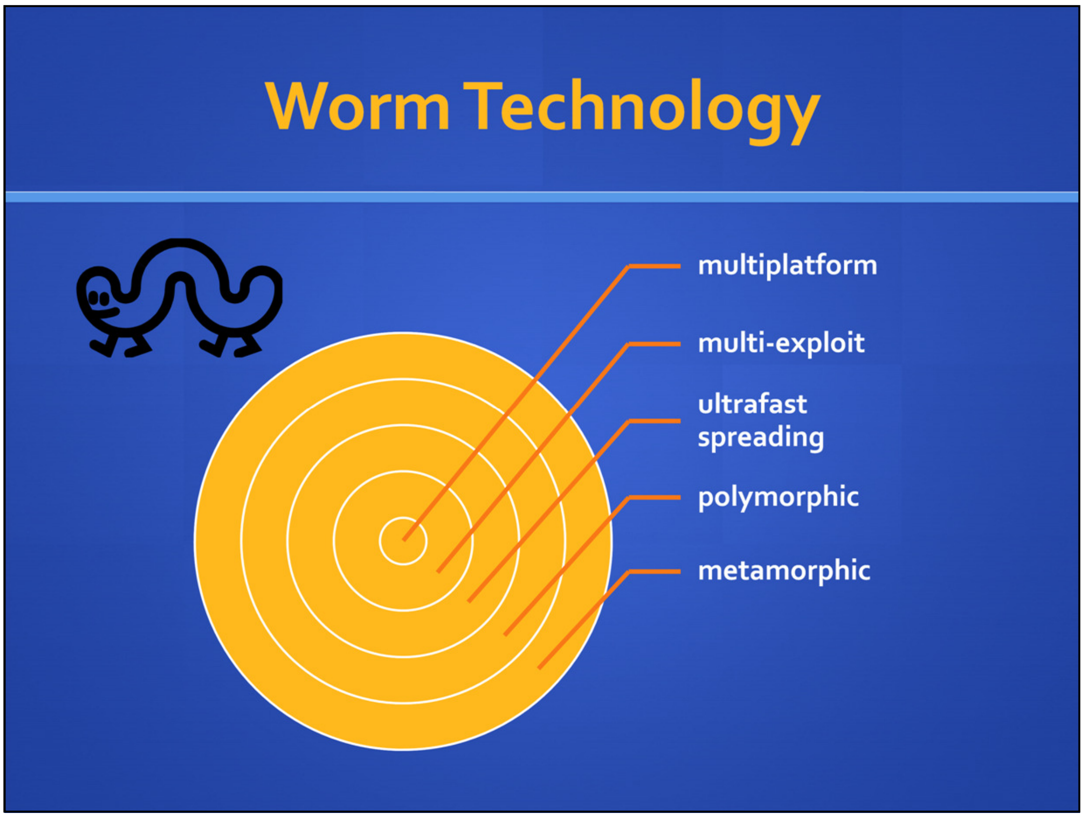
The Warezov family of worms appeared in 2006 [KIRK06]. When the worm is launched, it creates several executables in system directories and sets itself to run every time Windows starts by creating a registry entry. Warezov scans several types of files for e-mail addresses and sends itself as an e-mail 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. In 2010, it still comprised the second most common family of malware observed by Symantec [SYMA11], 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]. 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 at Symantec 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.



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.

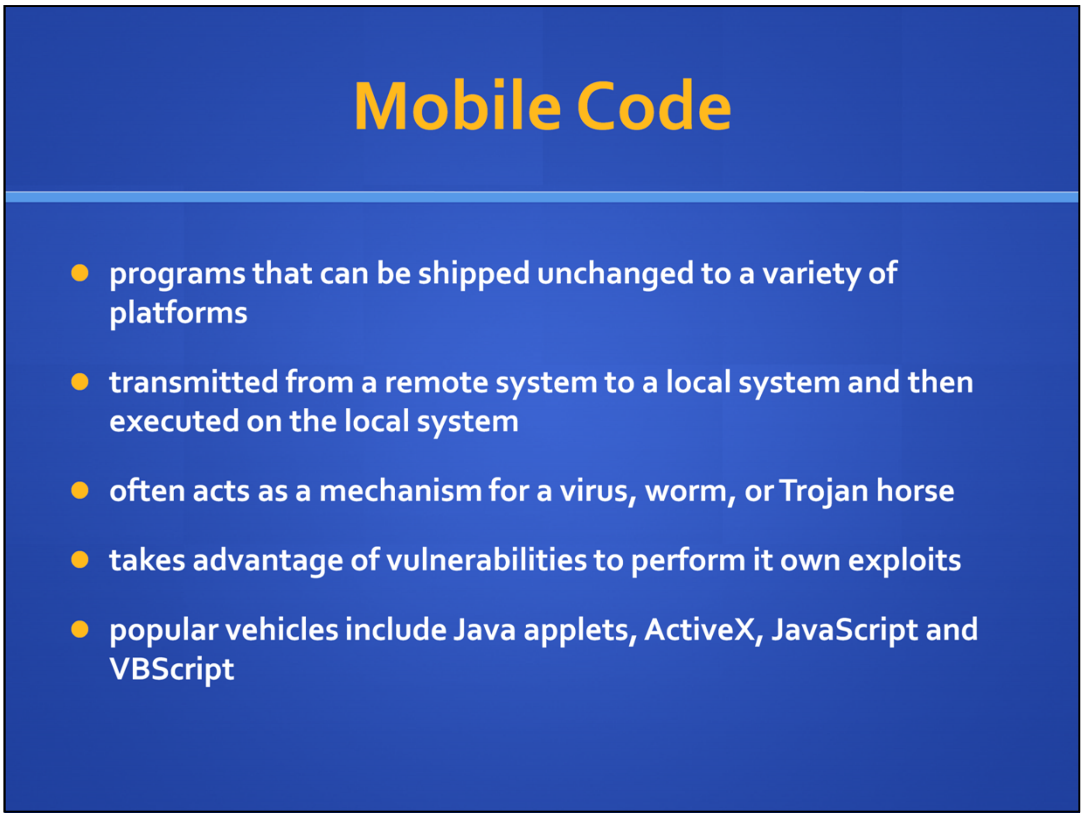
• **Polymorphic: To evade detection, skip past filters, and foil real-time analysis,**
worms adopt the virus polymorphic technique. Each copy of the worm has

new code generated on the fly using functionally equivalent instructions and encryption techniques.

• **Metamorphic: In addition to changing their appearance, metamorphic worms**
have a repertoire of behavior patterns that are unleashed at different stages of propagation.

• **Transport vehicles: Because worms can rapidly compromise a large number of**
systems, they are ideal for spreading a wide variety of malicious payloads, such as
distributed denial-of-service bots, rootkits, spam e-mail generators, and spyware.

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



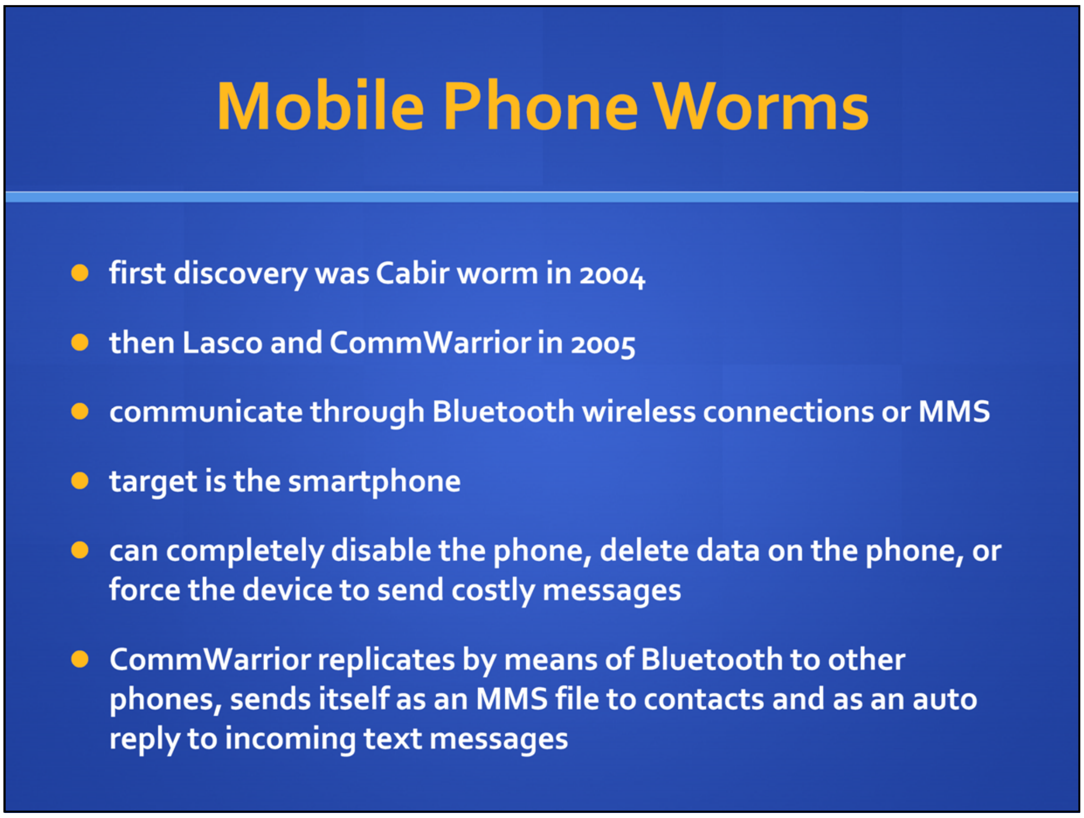
Mobile Code

- programs that can be shipped unchanged to a variety of platforms
- transmitted from a remote system to a local system and then executed on the local system
- often acts as a mechanism for a virus, worm, or Trojan horse
- takes advantage of vulnerabilities to perform it own exploits
- popular vehicles include Java applets, ActiveX, JavaScript and VBScript

Mobile code refers to programs (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics [JANS01].

Mobile code is transmitted from a remote system to a local system and then executed on the local system without the user's explicit instruction [NIST05]. Mobile code often acts as a mechanism for a virus, worm, or Trojan horse to be transmitted to the user's workstation. In other cases, mobile code takes advantage of vulnerabilities to perform its own exploits, such as unauthorized data access or root compromise.

Popular vehicles for mobile code include Java applets, ActiveX, JavaScript, and VBScript. The most common ways of using mobile code for malicious operations on local system are cross-site scripting, interactive and dynamic Web sites, e-mail attachments, and downloads from untrusted sites or of untrusted software.

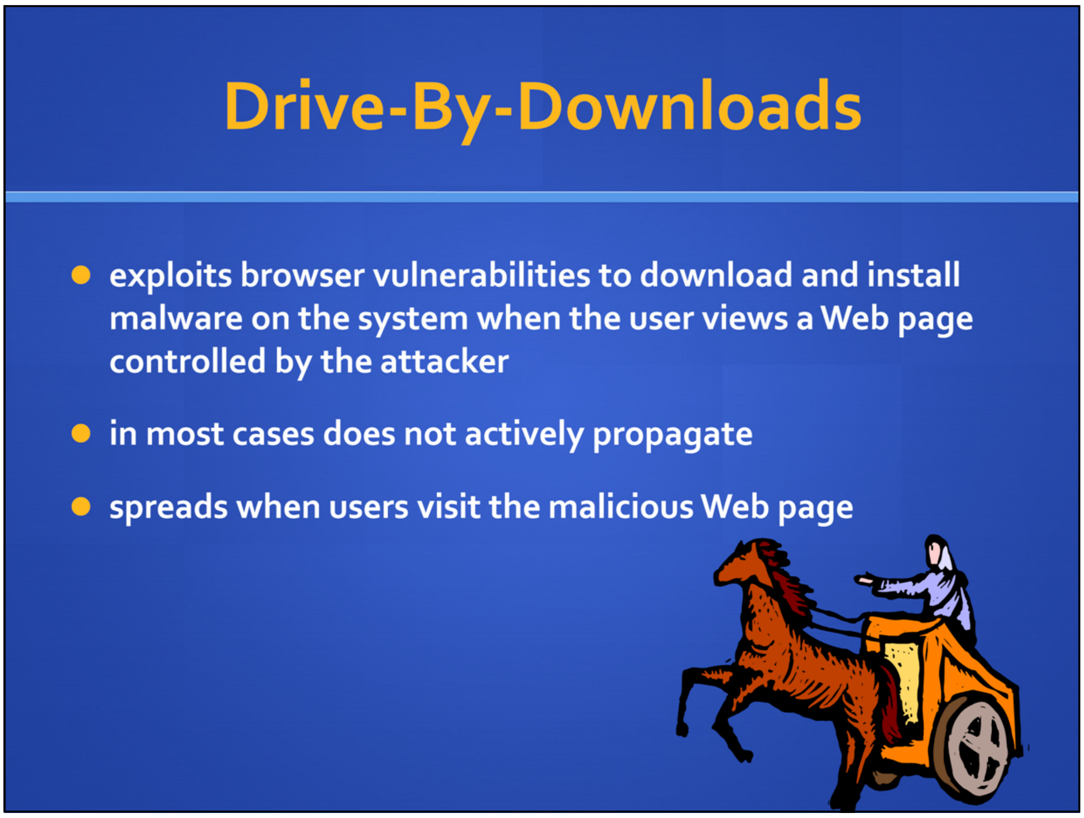


Mobile Phone Worms

- first discovery was Cabir worm in 2004
- then Lasco and CommWarrior in 2005
- communicate through Bluetooth wireless connections or MMS
- target is the smartphone
- can completely disable the phone, delete data on the phone, or force the device to send costly messages
- CommWarrior replicates by means of Bluetooth to other phones, sends itself as an MMS file to contacts and as an auto reply to incoming text messages

Worms first appeared on mobile phones with the discovery of the Cabir worm in 2004, and 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.



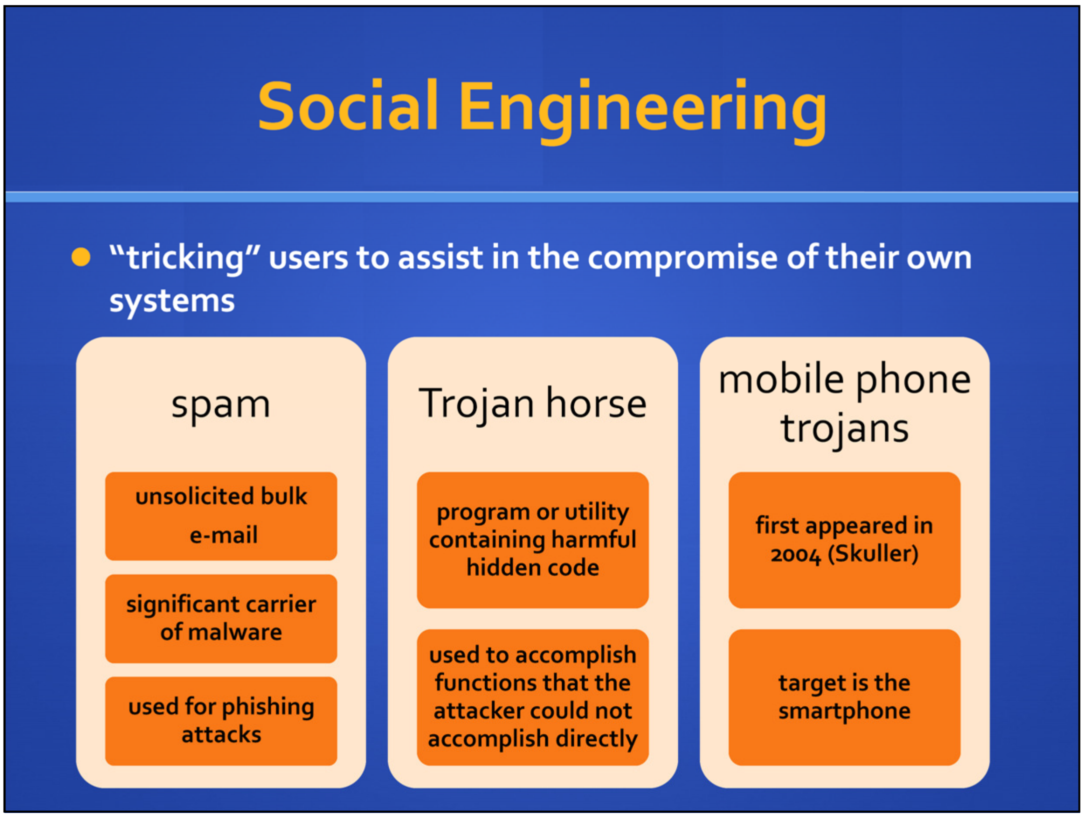
Drive-By-Downloads

- exploits browser vulnerabilities to download and install malware on the system when the user views a Web page controlled by the attacker
- in most cases does not actively propagate
- spreads when users visit the malicious Web page

Another approach to exploiting software vulnerabilities involves the exploit of
bugs in user applications to install malware. One common approach to this exploits
browser vulnerabilities so that when the user views a Web page controlled by the
attacker, it contains code that exploits the browser bug to download and install
malware on the system without the user's knowledge or consent. This is known as
a **drive-by-download and is a common exploit in recent attack kits. In most cases,**
this malware does not actively propagate as a worm does, but rather waits for unsuspecting
users to visit the malicious Web page in order to spread to their systems.

Related variants can exploit bugs in common e-mail clients, such as the Klez
mass-mailing worm seen in October 2001, which targeted a bug in the HTML
handling in Microsoft's Outlook and Outlook Express programs to automatically
run itself. Or, such malware may target common PDF viewers to also download and
install malware without the user's consent when they view a malicious PDF document

[STEV11]. Such documents may be spread by spam e-mail, or be part of a targeted phishing attack, as we discuss next.



Social Engineering

- "tricking" users to assist in the compromise of their own systems

spam
- unsolicited bulk e-mail
- significant carrier of malware
- used for phishing attacks

Trojan horse
- program or utility containing harmful hidden code
- used to accomplish functions that the attacker could not accomplish directly

mobile phone trojans
- first appeared in 2004 (Skuller)
- target is the smartphone

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.
A number of recent estimates suggest that spam e-mail may account for 90% or
more of all e-mail sent. 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].

While some spam is sent from legitimate mail servers, most recent spam is
sent by botnets using compromised user systems, as we discuss in Section 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 scams or money mule job ads. But spam is also a
significant
carrier of malware. The e-mail may have an attached document, which, if opened,
may exploit a software vulnerability to install malware on the user's system, as we
discussed in the previous section. Or, it may have an attached Trojan horse
program
or scripting code that, if run, also installs malware on the user's system. Some
trojans avoid the need for user agreement by exploiting a software vulnerability in
order to install themselves, as we discuss next. Finally the spam may be used in a
phishing attack, typically directing the user either to a fake Web site that mirrors
some legitimate service, such as an online banking site, where it attempts to
capture
the user's login and password details; or to complete some form with sufficient
personal details to allow the attacker to impersonate the user in an identity theft.
All of these uses make spam e-mails a significant security concern. However, in
many cases, it requires the user's active choice to view the e-mail and any
attached
document, or to permit the installation of some program, in order for the
compromise
to occur.

Trojan Horses

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

Trojan horse programs can be used to accomplish functions indirectly that
the attacker could not accomplish directly. For example, to gain access to
sensitive,
personal information stored in the files of a user, an attacker could create a Trojan
horse program that, when executed, scans the user's files for the desired sensitive
information and sends a copy of it to the attacker via a Web form or e-mail or text
message. The author could then entice users to run the program by incorporating

it
into a game or useful utility program, and making it available via a known software distribution site or app store. This approach has been used recently with utilities that “claim” to be the latest 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

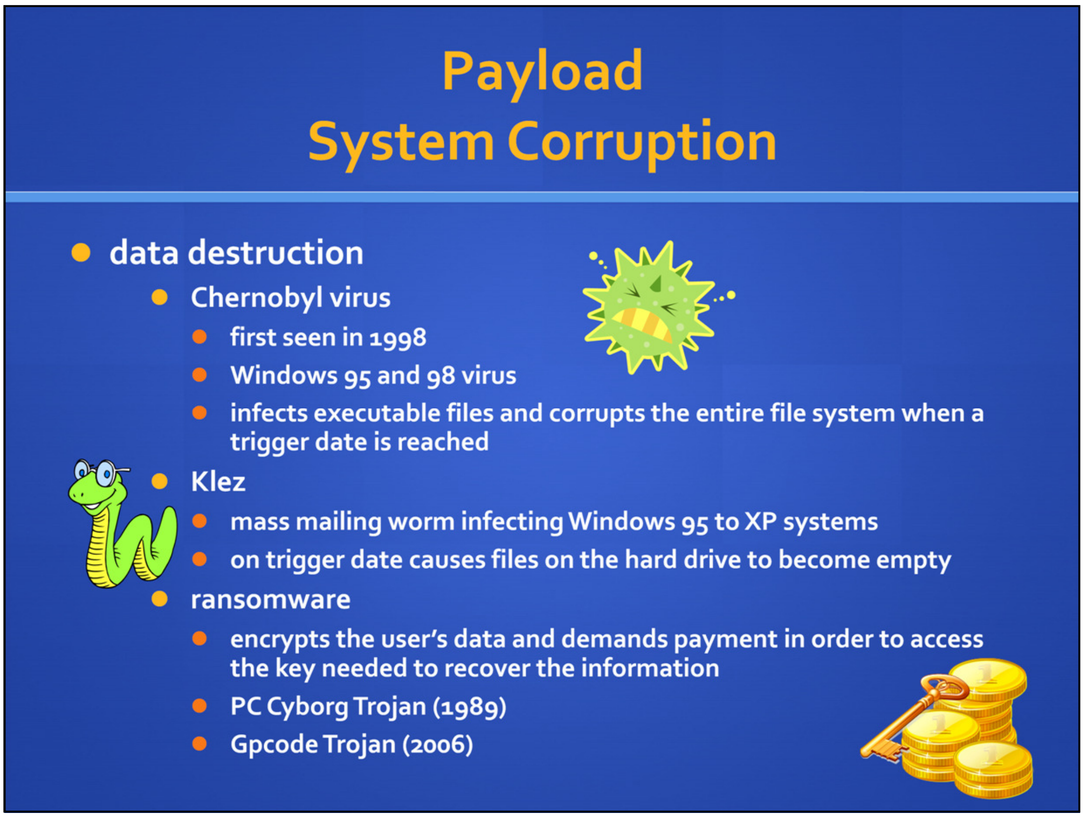
Some trojans avoid the requirement for user assistance by exploiting some software
vulnerability to enable their automatic installation and execution. In this they share some features of a worm, but unlike it, they do not replicate. A prominent example of such an attack was the Hydraq Trojan used in Operation Aurora in 2009 and early 2010. This exploited a vulnerability in Internet Explorer to install itself, and targeted several high-profile companies [SYMA11]. It was typically distributed using either spam e-mail or via a compromised Web site using a “driveby-
download.”

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 number of trojans have been detected that target Android phones and Apple iPhones.

In 2011, Google removed a number of apps from the Android Market that
were trojans containing the DroidDream malware. This is a powerful zombie agent that exploited vulnerabilities in some versions of Android used at this time to gain full access to the system to monitor data and install additional code.

The tighter controls that Apple impose on their app store, mean that most
iPhone trojans seen to date target "jail-broken" phones, and are distributed via unofficial sites. However a number of versions of the iPhone O/S included 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 return to this topic in Chapters 10 and 11 .



Payload
System Corruption

- data destruction
  - Chernobyl virus
    - first seen in 1998
    - Windows 95 and 98 virus
    - infects executable files and corrupts the entire file system when a trigger date is reached
  - Klez
    - mass mailing worm infecting Windows 95 to XP systems
    - on trigger date causes files on the hard drive to become empty
  - ransomware
    - encrypts the user's data and demands payment in order to access the key needed to recover the information
    - PC Cyborg Trojan (1989)
    - Gpcode Trojan (2006)

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.

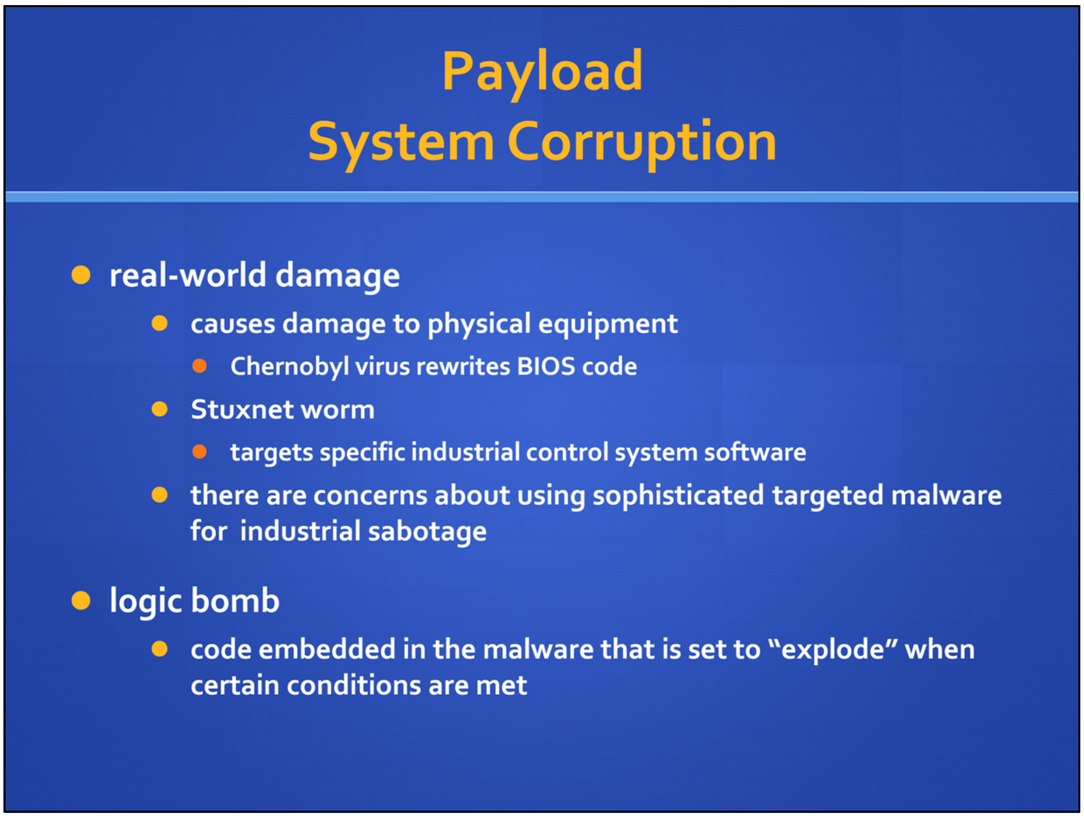
An early payload seen in a number of viruses and worms resulted in data destruction on the infected system when certain trigger conditions were met [WEAV03]. A related payload is one that displays unwanted messages or content on the user's system when triggered. More seriously, another variant attempts 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.

The Chernobyl virus is an early example of a destructive parasitic memory-resident

Windows-95 and 98 virus, that was first seen in 1998. It infects executable files when
they’re opened. And when a trigger date is reached, it 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 sometimes known as **ransomware . The PC Cyborg Trojan seen in**
1989 was an early example of this. However, around mid-2006, a number of worms
and trojans 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.



Payload
System Corruption

- real-world damage
  - causes damage to physical equipment
    - Chernobyl virus rewrites BIOS code
  - Stuxnet worm
    - targets specific industrial control system software
  - there are concerns about using sophisticated targeted malware for industrial sabotage
- logic bomb
  - code embedded in the malware that is set to "explode" when certain conditions are met

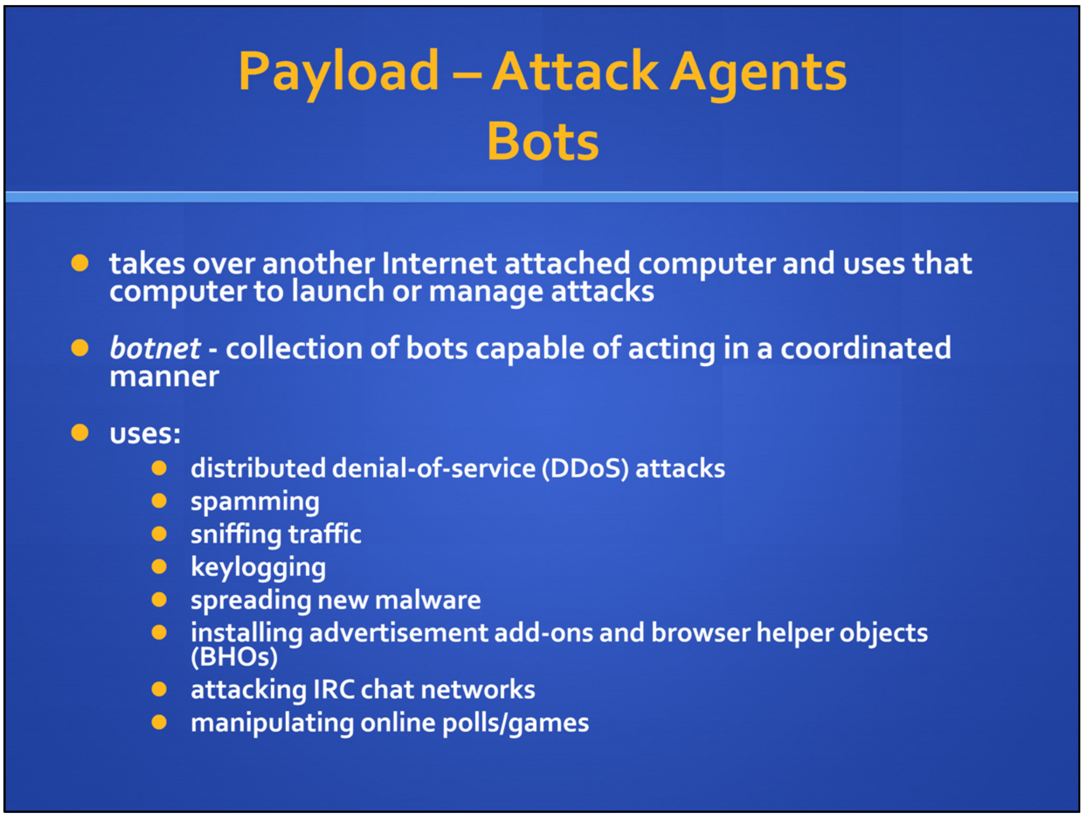
A further variant of system corruption payloads aims to cause damage to physical equipment. The infected system is clearly the device most easily targeted. The Chernobyl virus mentioned above not only corrupts data, but attempts to rewrite the BIOS code used to initially boot the computer. If it is successful, 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]. 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.

A key component of data corrupting malware is the logic bomb. The logic bomb is
code embedded in the malware that is set to “explode” when certain conditions are
met. Examples of conditions that can be used as triggers for a logic bomb are the presence
or absence of certain files or devices on the system, a particular day of the week
or date, a particular version or configuration of some software, or a particular user
running the application. Once triggered, a bomb may alter or delete data or entire files,
cause a machine halt, or do some other damage. All of the examples we describe in this
section include such code.

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.



Payload – Attack Agents Bots

- takes over another Internet attached computer and uses that computer to launch or manage attacks
- *botnet* - collection of bots capable of acting in a coordinated manner
- uses:
  - distributed denial-of-service (DDoS) attacks
  - spamming
  - sniffing traffic
  - keylogging
  - spreading new malware
  - installing advertisement add-ons and browser helper objects (BHOs)
  - attacking IRC chat networks
  - manipulating online polls/games

The next category of payload we discuss is where the malware subverts the computational

and network resources of the infected system for use by the attacker.

Such a system is known as a bot (robot), zombie or drone, and secretly takes over

another Internet-attached computer and then uses that computer to launch or manage

attacks that are difficult to trace to the bot's creator. The bot is typically planted

on hundreds or thousands of computers belonging to unsuspecting third parties.

The collection of bots often is capable of acting in a coordinated manner; such a

collection is referred to as a **botnet . This type of payload attacks the integrity and**

availability of the infected system.

Uses of Bots

[HONE05] lists the following uses of bots:

• **Distributed denial-of-service (DDoS) attacks: A DDoS attack is an attack on** a computer system or network that causes a loss of service to users. We examine DDoS attacks in 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 cleartext**
data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords.

Keylogging: If the compromised machine uses encrypted communication
channels (e.g. HTTPS or POP3S), then just sniffing the network packets on the victim's computer is useless because the appropriate key to decrypt the packets is missing. But by using a keylogger, which captures keystrokes on the infected machine, an attacker can retrieve sensitive information.

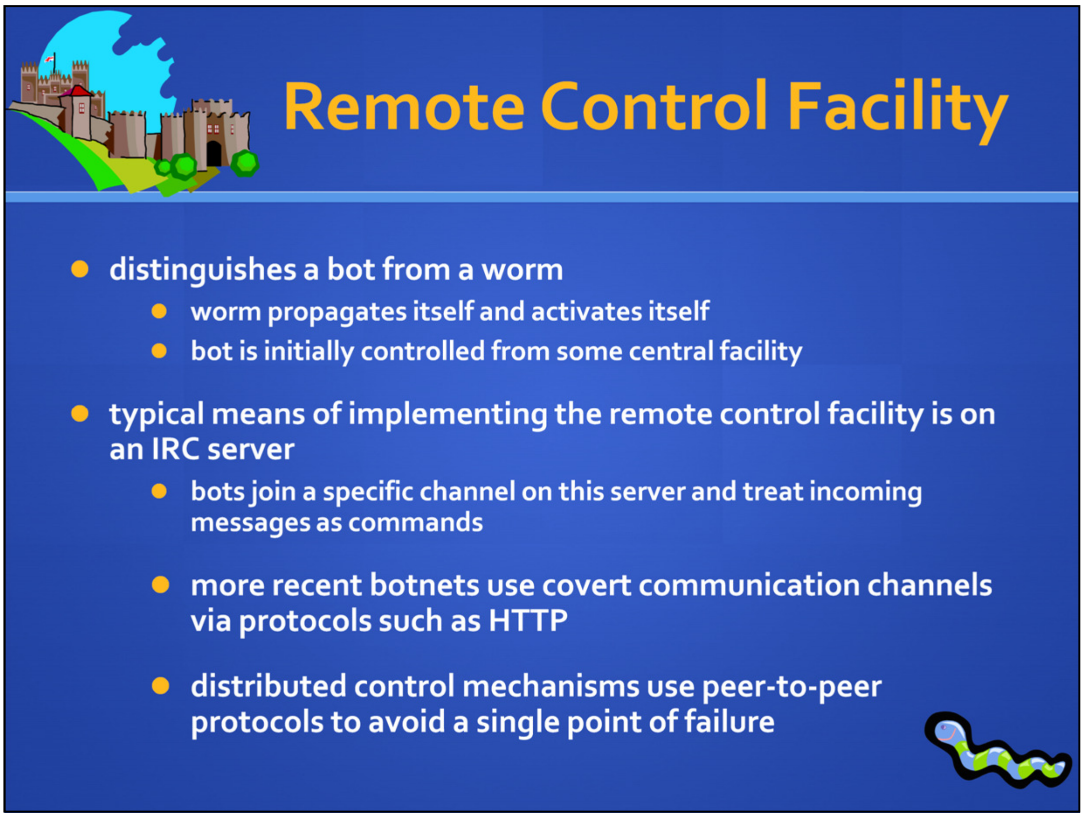
• **Spreading new malware: Botnets are used to spread new bots. This is very**
easy since all bots implement mechanisms to download and execute a file via HTTP or FTP. A botnet with 10,000 hosts that acts as the start base for a worm or mail virus allows very fast spreading and thus causes more harm.

• **Installing advertisement add-ons and browser helper objects (BHOs): Botnets**
can also be used to gain financial advantages. This works by setting up a fake Web site with some advertisements: The operator of this Web site negotiates a deal with some hosting companies that pay for clicks on ads. With the help of a botnet, these clicks can be "automated" so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start-page of a compromised machine so that the "clicks" are executed each time the victim uses the browser.

• **Attacking IRC chat networks: Botnets are also used for attacks against**
Internet Relay Chat (IRC) networks. Popular among attackers is 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 channeljoins
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

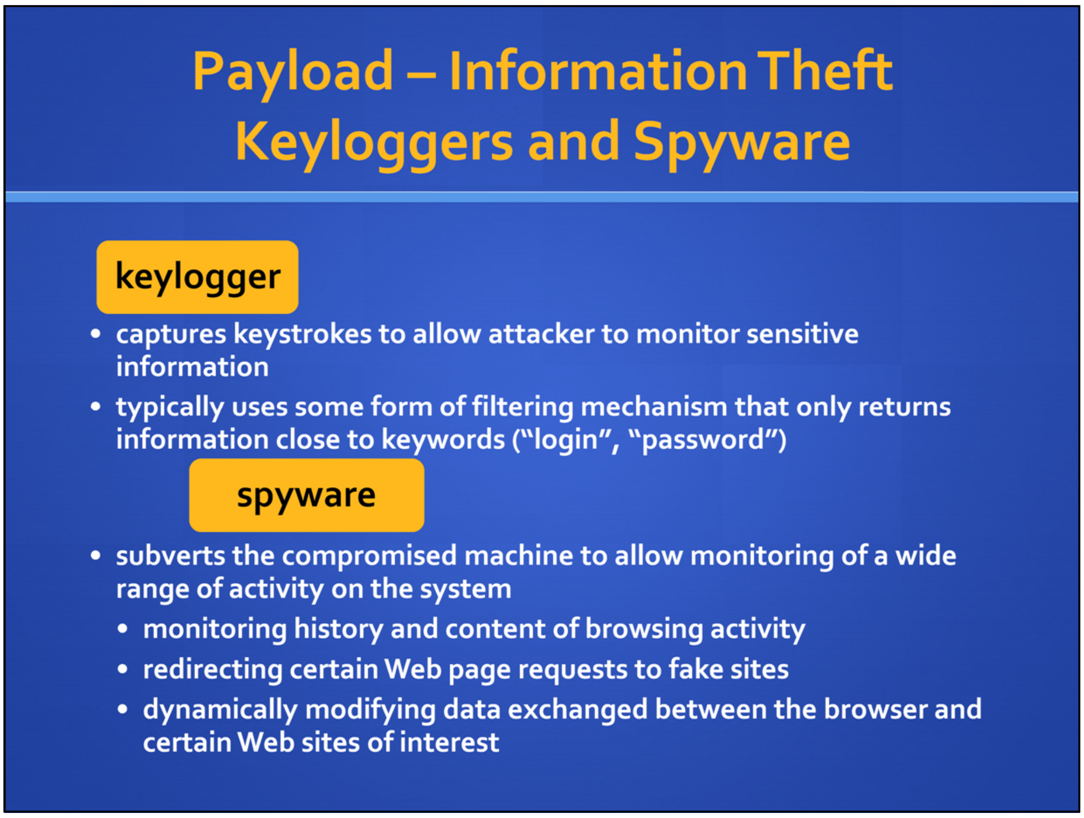
- distinguishes a bot from a worm
  - worm propagates itself and activates itself
  - bot is initially controlled from some central facility
- typical means of implementing the remote control facility is on an IRC server
  - bots join a specific channel on this server and treat incoming messages as commands
  - more recent botnets use covert communication channels via protocols such as HTTP
  - distributed control mechanisms use peer-to-peer protocols to avoid a single point of failure

The remote control facility is what distinguishes a bot from a worm. A worm propagates
itself and activates itself, whereas a bot is controlled from some central facility,
at least initially.

A typical means of implementing the remote control facility is on an IRC
server. All bots join a specific channel on this server and treat incoming messages
as commands. More recent botnets tend to avoid IRC mechanisms and use covert
communication channels via protocols such as HTTP. Distributed control mechanisms,
using peer-to-peer protocols, are also used, to avoid a single point of failure.

Once a communications path is established between a control module and
the bots, the control module can activate the bots. In its simplest form, the control
module simply issues command to the bot that causes the bot to execute routines
that are already implemented in the bot. For greater flexibility, the control module
can issue update commands that instruct the bots to download a file from some
Internet location and execute it. The bot in this latter case becomes a more general purpose

tool that can be used for multiple attacks.



Payload – Information Theft Keyloggers and Spyware

keylogger

- captures keystrokes to allow attacker to monitor sensitive information
- typically uses some form of filtering mechanism that only returns information close to keywords (“login”, “password”)

spyware

- subverts the compromised machine to allow monitoring of a wide range of activity on the system
  - monitoring history and content of browsing activity
  - redirecting certain Web page requests to fake sites
  - dynamically modifying data exchanged between the browser and certain Web sites of interest

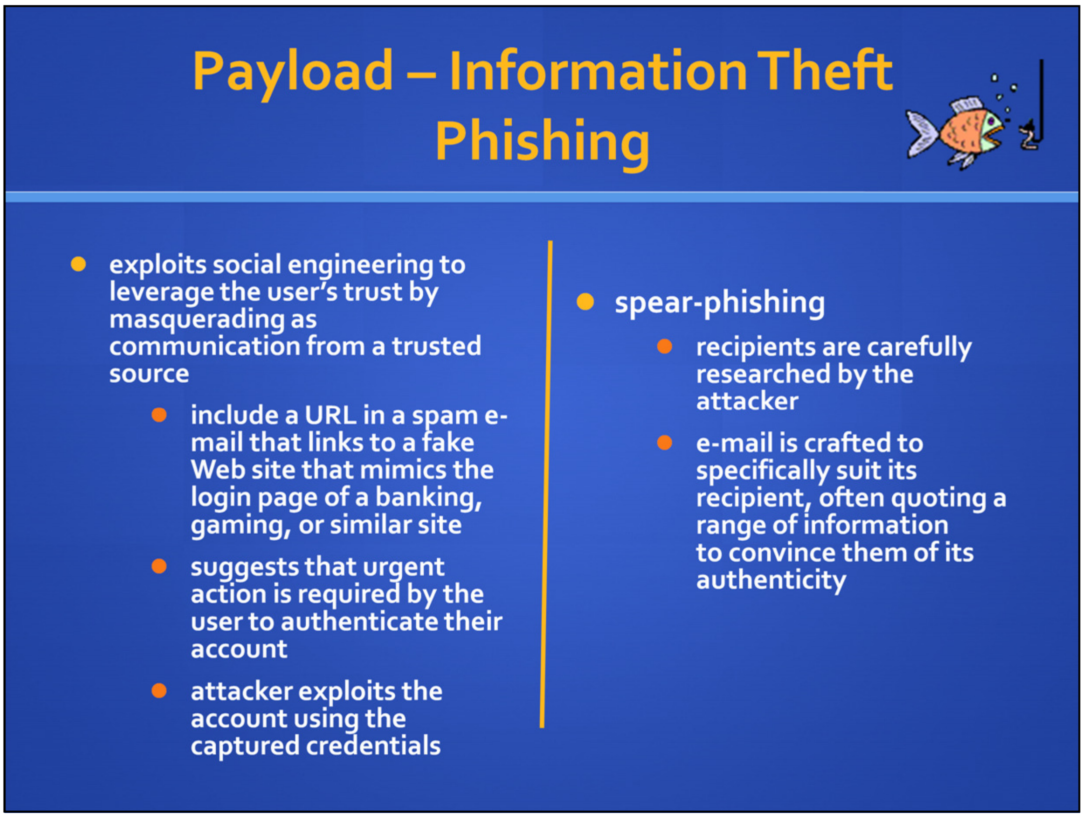
We now consider payloads where the malware gathers data stored on the infected system for use by the attacker. A common target is the user’s login and password credentials to banking, gaming, and related sites, which the attacker then uses to impersonate the user to access these sites for gain. Less 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 protects 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 typical implement some form of filtering mechanism that only returns information close to desired keywords (e.g., “login” or “password” or “paypal.com”).

In response to the use of keyloggers, some banking and other sites switched to using a graphical applet to enter critical information, such as passwords. Since these do not use text entered via the keyboard, traditional keyloggers do not capture this information. In response, attackers developed more general **spyware payloads,** which subvert the compromised machine to allow monitoring of a wide range of activity on the system. This may include monitoring the history and content of browsing activity, redirecting certain Web page requests to fake sites controlled by the attacker, and dynamically modifying data exchanged between the browser and certain Web sites of interest. All of which can result in significant compromise of the user’s personal information.

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



Payload – Information Theft Phishing

- exploits social engineering to leverage the user's trust by masquerading as communication from a trusted source
  - include a URL in a spam e-mail that links to a fake Web site that mimics the login page of a banking, gaming, or similar site
  - suggests that urgent action is required by the user to authenticate their account
  - attacker exploits the account using the captured credentials
- spear-phishing
  - recipients are carefully researched by the attacker
  - e-mail is crafted to specifically suit its recipient, often quoting a range of information to convince them of its authenticity

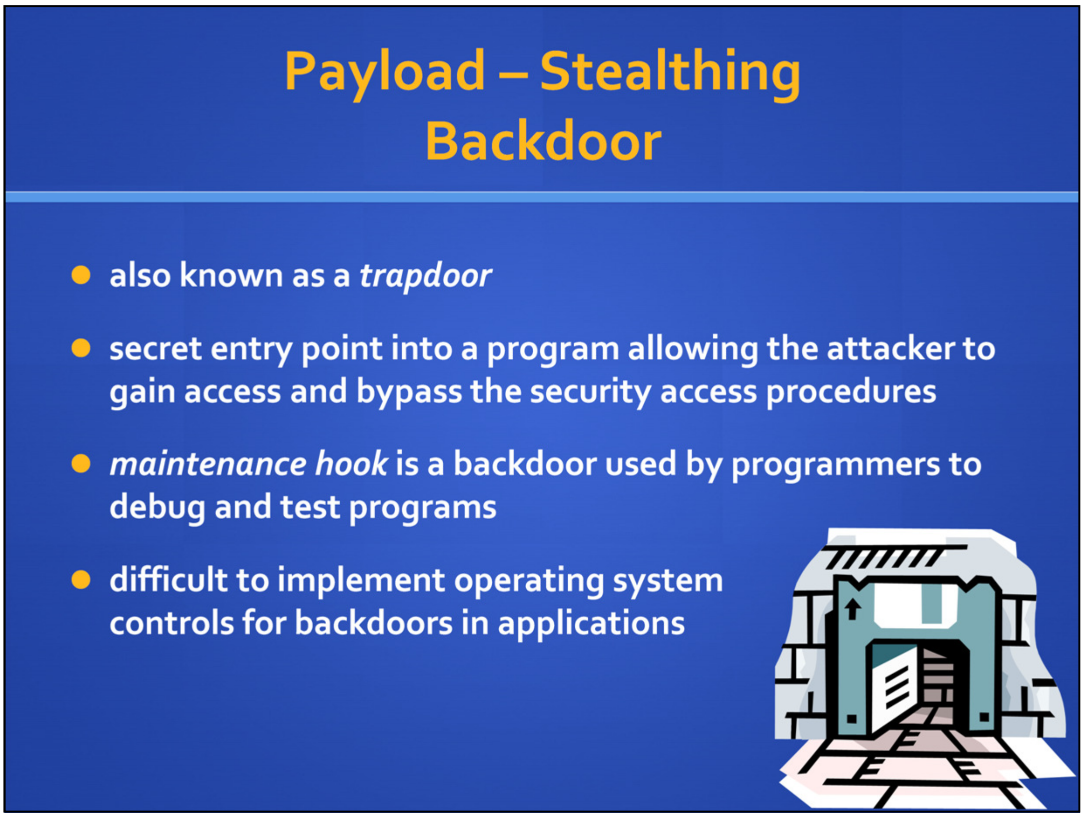
Another approach used to capture a user's login and password credentials is to include a URL in a spam e-mail that links to a fake Web site controlled by the attacker, but which mimics the login page of some banking, gaming, or similar site. This is normally included in some message suggesting that urgent action is required by the user to authenticate their account, to prevent it being locked. If the user is careless, and doesn't 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 Web site controlled by the attacker, or to complete some enclosed form and return to an e-mail accessible to the attacker, which is used to gather a range of private, personal, information on the user. Given sufficient details, the attacker can then "assume" the user's identity for the purpose of obtaining credit, or sensitive access to other resources. This is known as a **phishing attack 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.

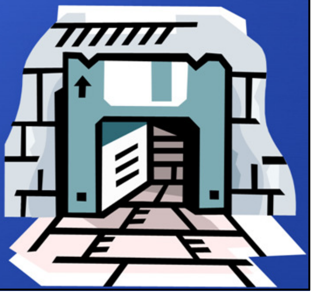
A more dangerous variant of this is the **spear-phishing attack. This again is an** e-mail claiming to be from a trusted source. However, the recipients are carefully researched by the attacker, and each e-mail is carefully crafted to suit its recipient specifically, often quoting a range of information to convince them of its authenticity. This greatly increases the likelihood of the recipient responding as desired by the attacker.

Reconnaissance and Espionage
Credential theft and identity theft are special cases of a more general reconnaissance payload, which aims to obtain certain types of desired information and return this to the attacker. These special cases are certainly the most common; however, other targets are known. Operation Aurora in 2009 used a Trojan to gain access to and potentially modify source code repositories at a range of high tech, security, and defense contractor companies [SYMA11]. 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].



Payload – Stealthing Backdoor

- also known as a *trapdoor*
- secret entry point into a program allowing the attacker to gain access and bypass the security access procedures
- *maintenance hook* is a backdoor used by programmers to debug and test programs
- difficult to implement operating system controls for backdoors in applications



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

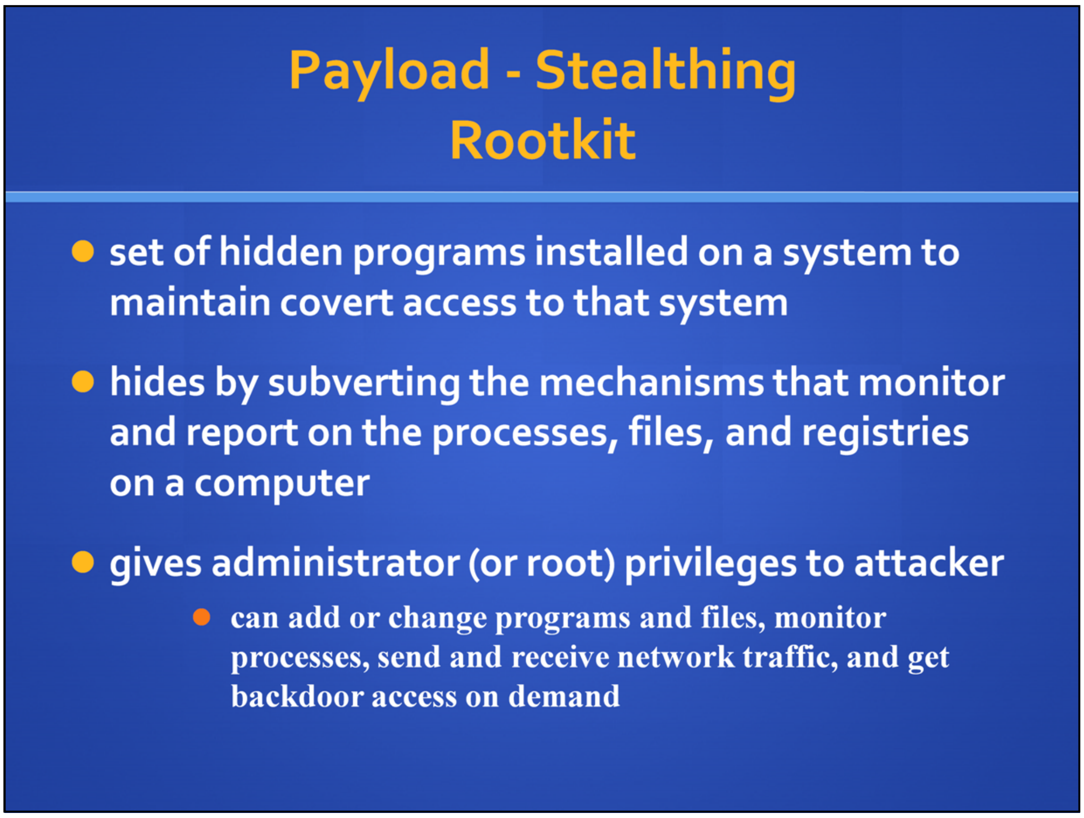
A **backdoor , also known as a trapdoor , is a secret entry point into a program**
that allows someone who is aware of the backdoor to gain access without going
through the usual security access procedures. Programmers have used backdoors
legitimately for many years to debug and test programs; such a 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 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.

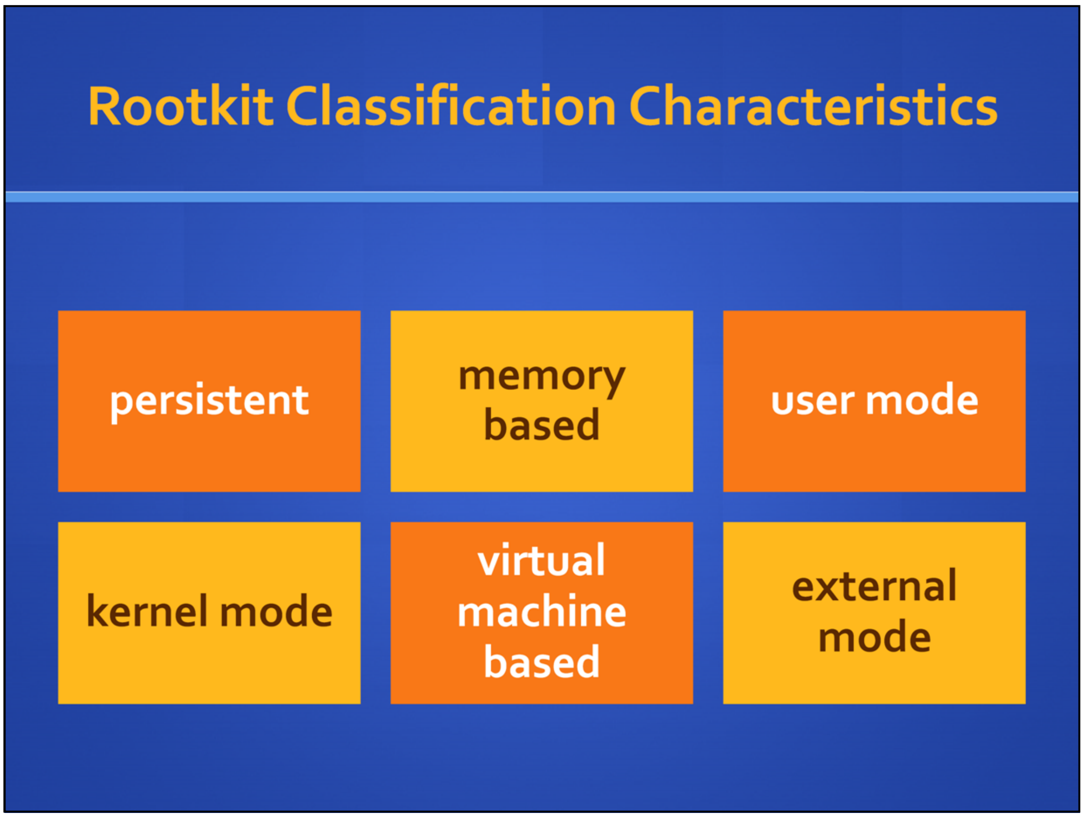
It is difficult to implement operating system controls for backdoors in
applications. Security measures must focus on the program development and
software update activities, and on programs that wish to offer a network service.



Payload - Stealthing Rootkit

- set of hidden programs installed on a system to maintain covert access to that system
- hides by subverting the mechanisms that monitor and report on the processes, files, and registries on a computer
- gives administrator (or root) privileges to attacker
  - can add or change programs and files, monitor processes, send and receive network traffic, and get backdoor access on demand

A rootkit is a set of programs installed on a system to maintain covert access to that
system with administrator (or root) 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.
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.



Rootkit Classification Characteristics

persistent	memory based	user mode
kernel mode	virtual machine based	external mode

A rootkit can be classified using the following characteristics:

• **Persistent: Activates each time the system boots. The rootkit must store code** in a persistent store, such as the registry or file system, and configure a method by which the code executes without user intervention. This means it is easier to detect, as the copy in persistent storage can potentially be scanned.

• **Memory based: Has no persistent code and therefore cannot survive a reboot.** However, because it is only in memory, it can be harder to detect.

• **User mode: Intercepts calls to APIs (application program interfaces) and modifies** returned results. For example, when an application performs a directory listing, the return results don't include entries identifying the files associated with the rootkit.

• **Kernel mode: Can intercept calls to native APIs in kernel mode. The rootkit** can also hide the presence of a malware process by removing it from the kernel's list of active processes.

• **Virtual machine based: This type of rootkit installs a lightweight virtual** machine monitor, and then runs the operating system in a virtual machine above it. The rootkit can then transparently intercept and modify states and events occurring in the virtualized system.

• **External mode: The malware is located outside the normal operation mode** of the targeted system, in BIOS or system management mode, where it can directly access hardware.

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

System Call Table Modification

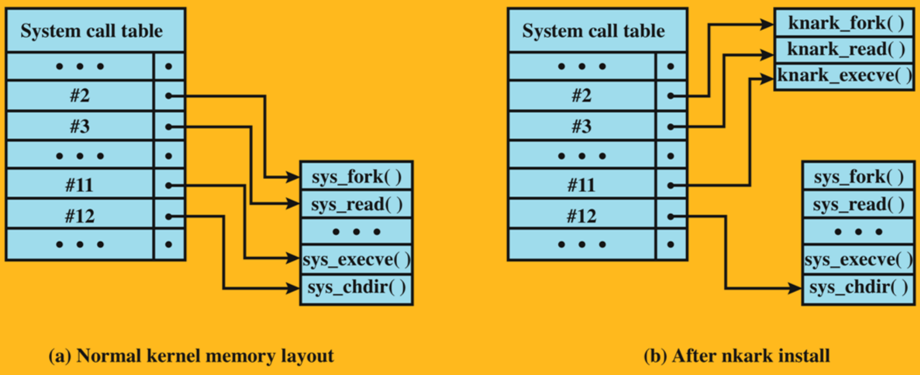


Figure 6.5 System Call Table Modification by Rootkit (based on [LEVI06])

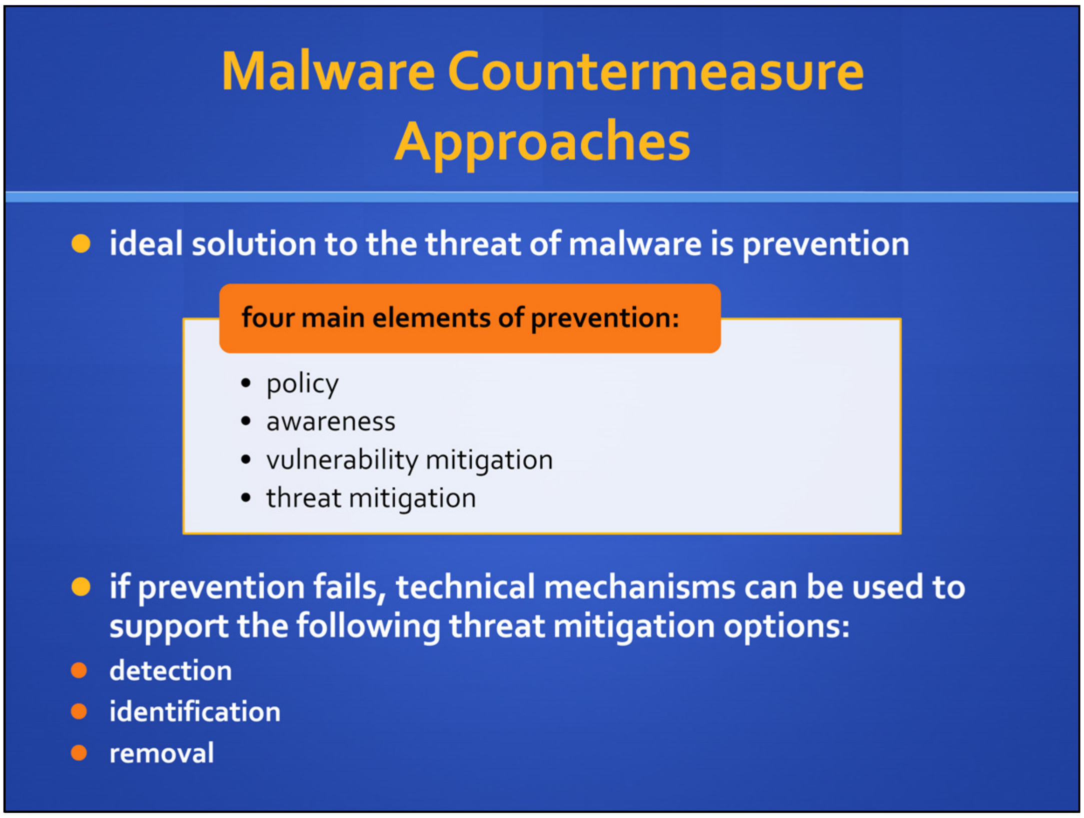
The next generation of rootkits moved down a layer, making changes inside the kernel and co-existing with the operating systems code, in order to make their detection much harder. Any "anti-virus" program would now be subject to the same "low-level" modifications that the rootkit uses to hide its presence. 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 6.5
shows how the knark rootkit achieves this.

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.



Malware Countermeasure Approaches

- ideal solution to the threat of malware is prevention

four main elements of prevention:

- policy
- awareness
- vulnerability mitigation
- threat mitigation

- if prevention fails, technical mechanisms can be used to support the following threat mitigation options:
- detection
- identification
- removal

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. [NIST05] 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 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. [NIST05] provides examples of suitable awareness issues. We 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 that it cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard any infected or malicious files and reload a clean backup version. In the case of some particularly nasty infections, this may require a complete wipe of all storage, and rebuild of the infected system from known clean media.

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

• **Generality: The approach taken should be able to handle a wide variety of attacks.**

• **Timeliness: The approach should respond quickly so as to limit the number of** infected programs or systems and the consequent activity.

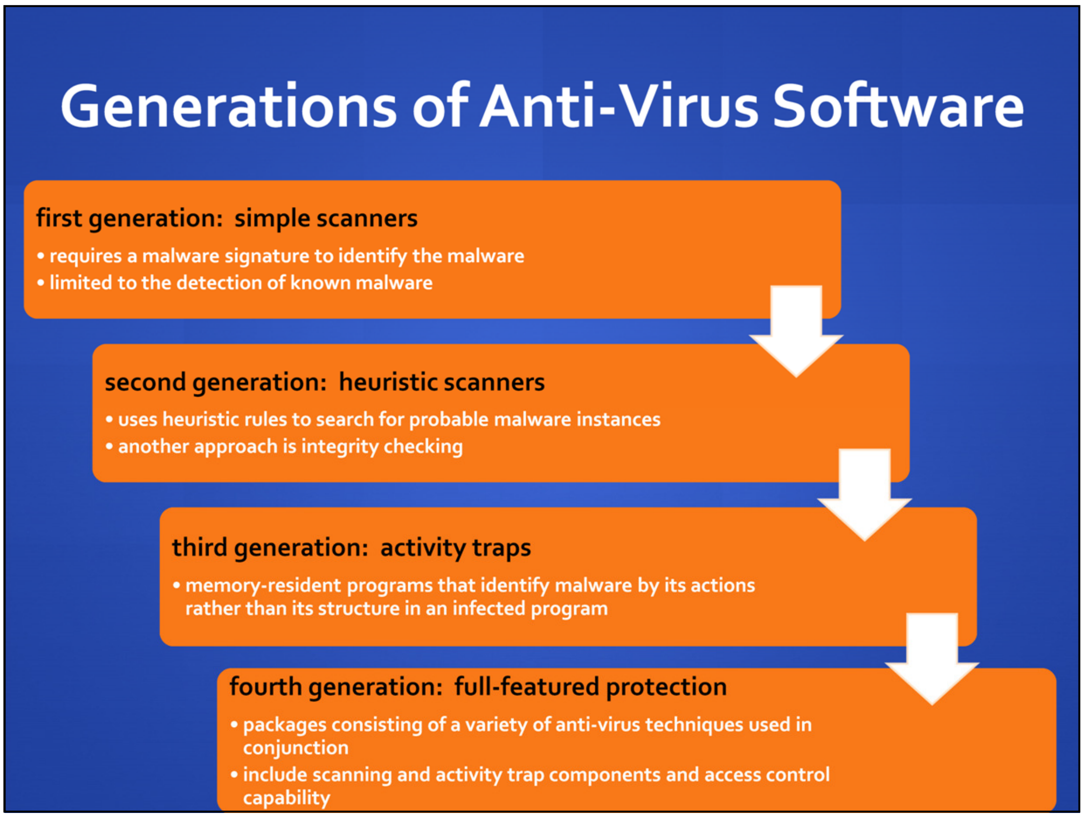
• **Resiliency: The approach should be resistant to evasion techniques employed** by attackers to hide the presence of their malware.

• **Minimal denial-of-service costs: The approach should result in minimal reduction** in capacity or service due to the actions of the countermeasure software, and should not significantly disrupt normal operation.

• **Transparency: The countermeasure software and devices should not require** modification to existing (legacy) OSs, application software, and hardware.

• **Global and local coverage: The approach should be able to deal with attack** sources both from outside and inside the enterprise network.
Achieving all these requirements often requires the use of multiple approaches.

Detection of the presence of malware can occur in a number of locations. It may occur on the infected system, where some host-based "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.



Generations of Anti-Virus Software

first generation: simple scanners

- requires a malware signature to identify the malware
- limited to the detection of known malware

second generation: heuristic scanners

- uses heuristic rules to search for probable malware instances
- another approach is integrity checking

third generation: activity traps

- memory-resident programs that identify malware by its actions rather than its structure in an infected program

fourth generation: full-featured protection

- packages consisting of a variety of anti-virus techniques used in conjunction
- include scanning and activity trap components and access control capability

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.
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.

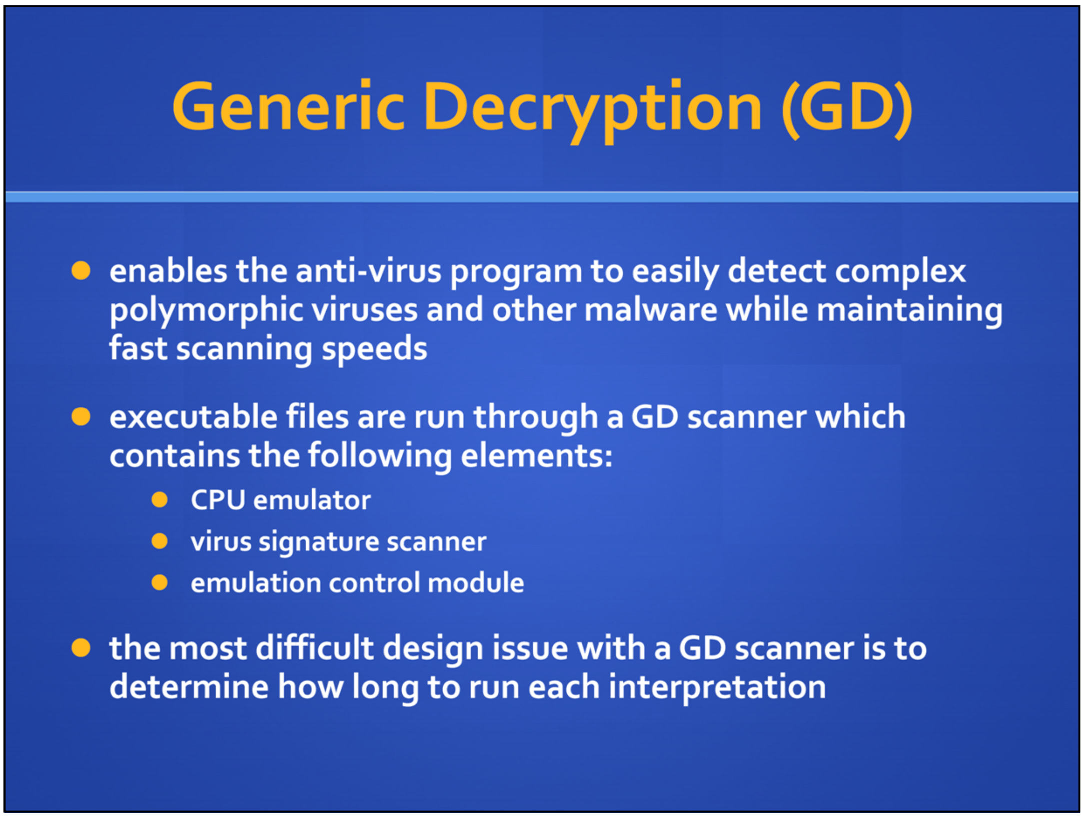
Another second-generation approach is integrity checking. A checksum can be appended to each program. If malware alters or replaces some program without changing the checksum, then an integrity check will catch this change. To counter malware that is sophisticated enough to change the checksum when it alters a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the malware cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum,
the malware is prevented from adjusting the program to produce the same hash code as before. If a protected list of programs in trusted locations is kept,

this
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.

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. We now highlight two of the most important.



Generic Decryption (GD)

- enables the anti-virus program to easily detect complex polymorphic viruses and other malware while maintaining fast scanning speeds
- executable files are run through a GD scanner which contains the following elements:
  - CPU emulator
  - virus signature scanner
  - emulation control module
- the most difficult design issue with a GD scanner is to determine how long to run each interpretation

Generic decryption (GD) technology enables the antivirus
program to easily detect even the most complex polymorphic viruses and other
malware, while maintaining fast scanning speeds [NACH97]. Recall that when a file
containing a polymorphic virus is executed, the virus must decrypt itself to activate. In order to detect such a structure, executable files are run through a GD scanner,
which contains the following elements:

• **CPU emulator: A software-based virtual computer. Instructions in an executable**
file are interpreted by the emulator rather than executed on the underlying
processor. The emulator includes software versions of all registers and other
processor hardware, so that the underlying processor is unaffected by programs
interpreted on the emulator.

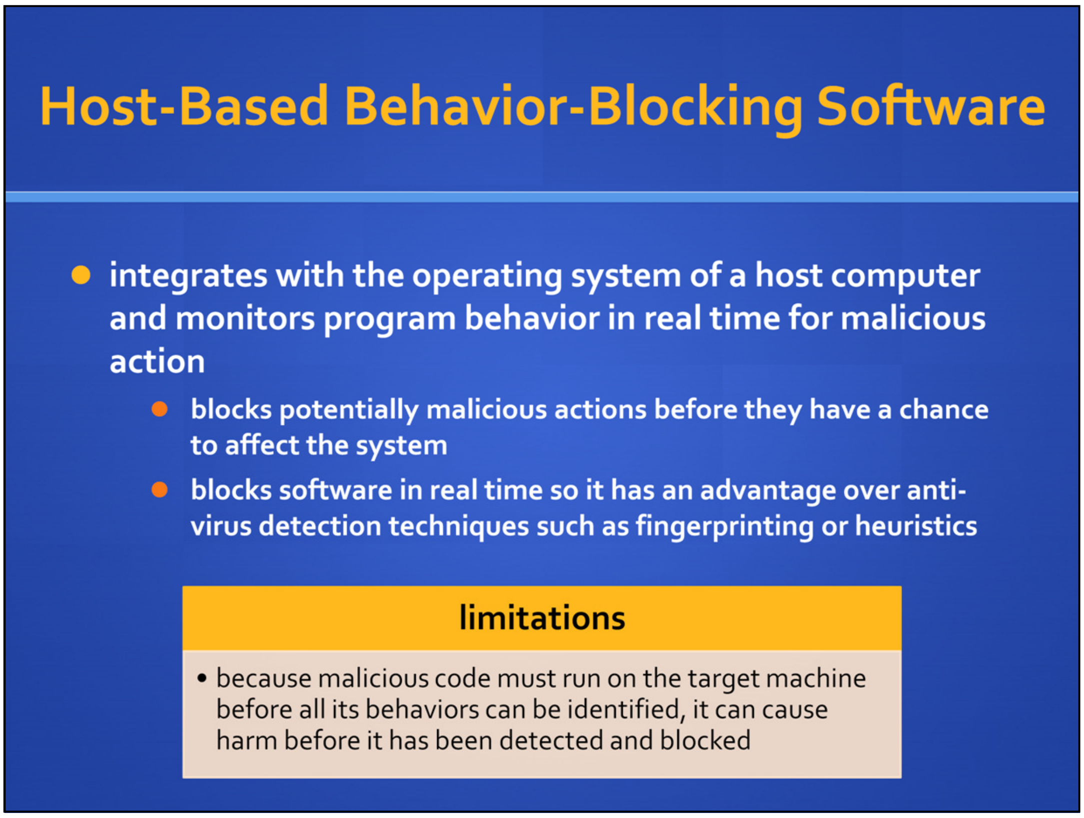
• **Virus signature scanner: A module that scans the target code looking for**
known malware signatures.

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

At the start of each simulation, the emulator begins interpreting instructions
in the target code, one at a time. Thus, if the code includes a decryption routine
that decrypts and hence exposes the malware, that code is interpreted. In effect, the
malware does the work for the anti-virus program by exposing itself. Periodically,
the control module interrupts interpretation to scan the target code for malware
signatures.

During interpretation, the target code can cause no damage to the actual
personal computer environment, because it is being interpreted in a completely
controlled environment.

The most difficult design issue with a GD scanner is to determine how long to run each interpretation. Typically, malware elements are activated soon after a program begins executing, but this need not be the case. The longer the scanner emulates a particular program, the more likely it is to catch any hidden malware. However, the anti-virus program can take up only a limited amount of time and resources before users complain of degraded system performance.


Host-Based Behavior-Blocking Software

- integrates with the operating system of a host computer and monitors program behavior in real time for malicious action
  - blocks potentially malicious actions before they have a chance to affect the system
  - blocks software in real time so it has an advantage over anti-virus detection techniques such as fingerprinting or heuristics

limitations

- because 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

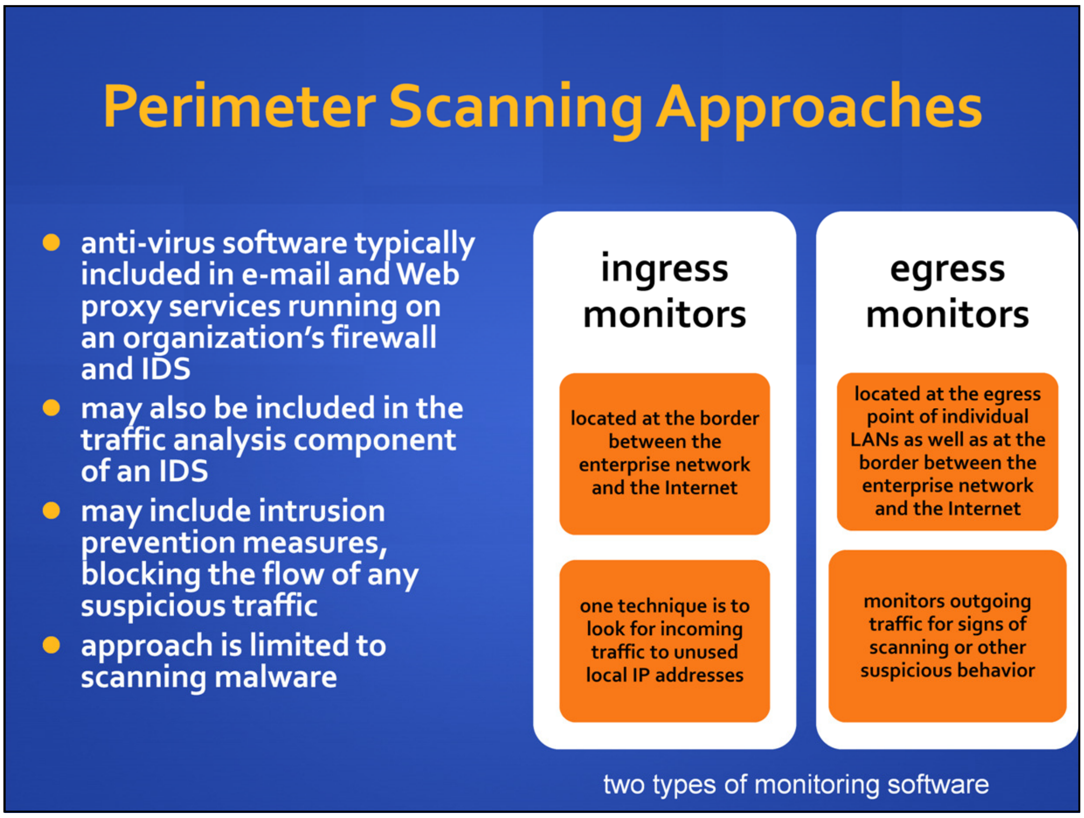
Unlike heuristics or fingerprint based
scanners, behavior-blocking software integrates with the operating system of
a host computer and monitors program behavior in real time for malicious actions
[CONR02, NACH02]. The behavior blocking software then blocks potentially
malicious actions before they have a chance to affect the system. Monitored
behaviors can include

• 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;
and
• Initiation of network communications.

Because a behavior blocker 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.

Behavior blocking alone has limitations. Because the malicious code must
run on the target machine before all its behaviors can be identified, it can cause
harm before it has been detected and blocked. For example, a new item of malware
might shuffle a number of seemingly unimportant files around the hard drive before
modifying a single file and being blocked. Even though the actual modification was
blocked, the user may be unable to locate his or her files, causing a loss to productivity
or possibly worse.



Perimeter Scanning Approaches

- anti-virus software typically included in e-mail and Web proxy services running on an organization's firewall and IDS
- may also be included in the traffic analysis component of an IDS
- may include intrusion prevention measures, blocking the flow of any suspicious traffic
- approach is limited to scanning malware

ingress monitors
- located at the border between the enterprise network and the Internet
- one technique is to look for incoming traffic to unused local IP addresses

egress monitors
- located at the egress point of individual LANs as well as at the border between the enterprise network and the Internet
- monitors outgoing traffic for signs of scanning or other suspicious behavior

two types of monitoring software

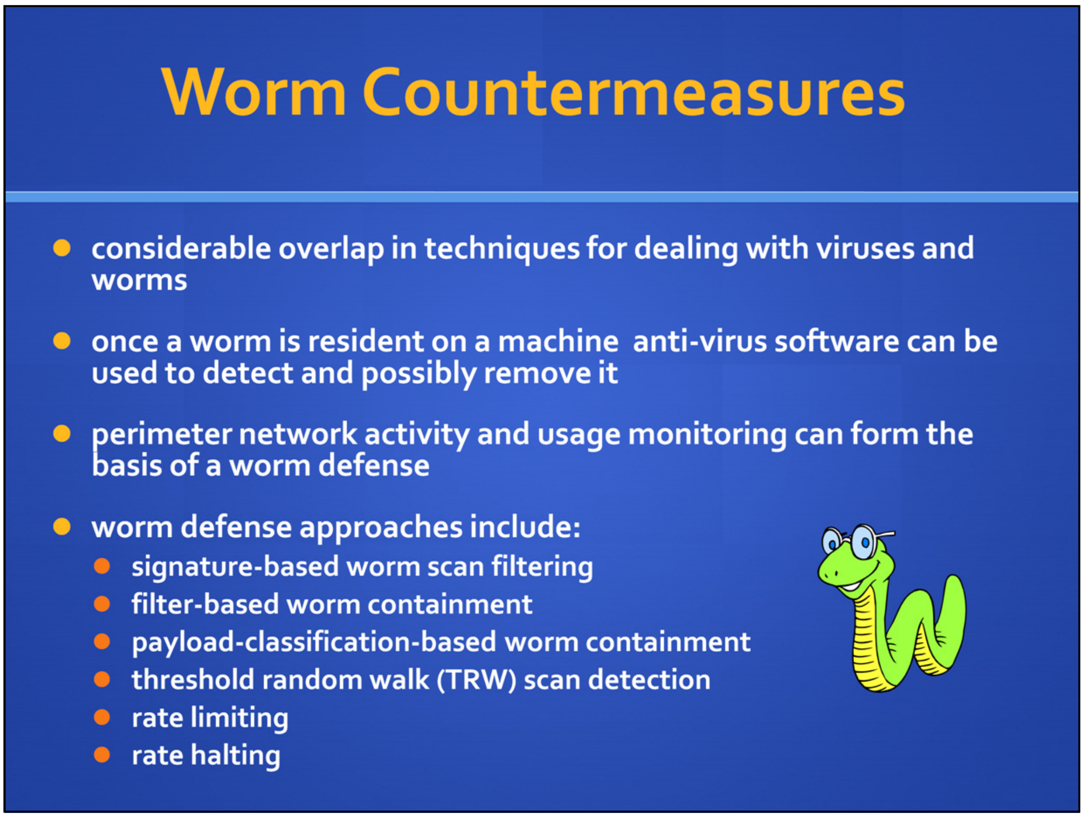
The next location where anti-virus software is used is on an organization's firewall and IDS. It is typically included in e-mail and Web proxy services running on these systems. It may also be included in the traffic analysis component of an IDS. This gives the anti-virus software access to malware in 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. 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 collocated. The egress monitor is designed to catch the source of a malware attack by monitoring outgoing traffic for signs of scanning or other suspicious behavior.

Perimeter monitoring can also assist in detecting and responding to botnet activity by detecting abnormal traffic patterns associated with this activity. Once bots are activated and an attack is underway, such monitoring can be used to detect the attack. However, the primary objective is to try to detect and disable the botnet during its construction phase, using the various scanning techniques we have just discussed, identifying and blocking the malware that is used to propagate this type of payload.



Worm Countermeasures

- considerable overlap in techniques for dealing with viruses and worms
- once a worm is resident on a machine anti-virus software can be used to detect and possibly remove it
- perimeter network activity and usage monitoring can form the basis of a worm defense
- worm defense approaches include:
  - signature-based worm scan filtering
  - filter-based worm containment
  - payload-classification-based worm containment
  - threshold random walk (TRW) scan detection
  - rate limiting
  - rate halting

There is considerable overlap in techniques for
dealing with viruses and worms. Once a worm is resident on a machine, anti-virus
software can be used to detect it, and possibly remove it. In addition, because worm
propagation generates considerable network activity, perimeter network activity
and usage monitoring can form the basis of a worm defense. Following [JHI07], we
list six classes of worm defense that address the network activity it may generate:

A. Signature-based worm scan filtering: This type of approach generates a worm
signature, which is then used to prevent worm scans from entering/leaving a
network/host. Typically, this approach involves identifying suspicious flows
and generating a worm signature. This approach is vulnerable to the use of
polymorphic worms: Either the detection software misses the worm or, if it
is sufficiently sophisticated to deal with polymorphic worms, the scheme may
take a long time to react. [NEWS05] is an example of this approach.

B. Filter-based worm containment: This approach is similar to class A but

focuses on
worm content rather than a scan signature. The filter checks a message to determine
if it contains worm code. An example is Vigilante [COST05], which relies
on collaborative worm detection at end hosts. This approach can be quite effective
but requires efficient detection algorithms and rapid alert dissemination.

C. Payload-classification-based worm containment: These network-based
techniques examine packets to see if they contain a worm. Various anomaly
detection techniques can be used, but care is needed to avoid high levels
of false positives or negatives. An example of this approach is reported in
[CHIN05], which looks for exploit code in network flows. This approach does
not generate signatures based on byte patterns but rather looks for control
and data flow structures that suggest an exploit.

D. Threshold random walk (TRW) scan detection: TRW exploits randomness in
picking destinations to connect to as a way of detecting if a scanner is in operation
[JUNG04]. TRW is suitable for deployment in high-speed, low-cost network
devices. It is effective against the common behavior seen in worm scans.

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

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

suitable for slow, stealthy worms.

Digital Immune System

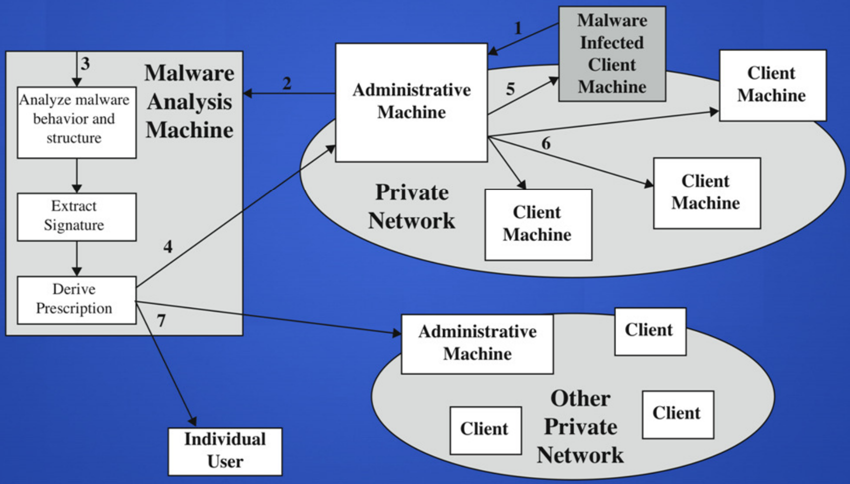


Figure 6.6 Digital Immune System

The digital immune system is a comprehensive
approach to virus protection developed by IBM [KEPH97a, KEPH97b, WHIT99]
and subsequently refined by Symantec [SYMA01]. In 2010, their resulting Global
Intelligence Network comprised more than 240,000 sensors, and gathered intelligence
on malicious code from more than 133 million client, server, and gateway systems
that have deployed Symantec anti-virus products [SYMA11]. The motivation for
this development has been the rising threat of Internet-based virus propagation, and
the need to acquire a global view of the situation.

Traditionally, the virus threat was characterized by the relatively slow spread
of new viruses and new mutations. Anti-virus software was typically updated on a
monthly basis, and this was sufficient to control the problem. Also traditionally, the
Internet played a comparatively small role in the spread of viruses. But as [CHES97]
points out, two major trends in Internet technology have had an increasing impact
on the rate of virus propagation over recent decades:

• **Integrated mail systems: Systems such as Lotus Notes and Microsoft Outlook**
make it very simple to send anything to anyone and to work with objects that
are received.

• **Mobile-program systems: Capabilities such as Java and ActiveX allow**
programs to move on their own from one system to another.

In response to the threat posed by these Internet-based capabilities, IBM
developed the original prototype digital immune system. This system expands on
the use of program emulation, discussed in the preceding subsection, and provides
a general-purpose emulation and malware detection system. The objective of this
system is to provide rapid response time so that malware can be stamped out almost
as soon as they are introduced. When new malware enters an organization, the
immune system automatically captures it, analyzes it, adds detection and shielding
for it, removes it, and passes information about it to client systems, so the malware
can be detected before it is allowed to run elsewhere.

Figure 6.6 illustrates the typical steps in early proposals for digital immune
system operation:

1. A monitoring program on each PC uses a variety of heuristics based on system
behavior, suspicious changes to programs, or family signature to infer that
malware may be present. The monitoring program forwards a copy of any
suspect program to an administrative machine within the organization.

The administrative machine encrypts the sample and sends it to a central
malware analysis system.

3. This machine creates an environment in which the suspect program can be
safely run for analysis. Techniques used for this purpose include emulation,
or the creation of a protected environment within which the suspect program

can be executed and monitored. The malware analysis system then produces a prescription for identifying and removing the malware.

4. The resulting prescription is sent back to the administrative machine.

5. The administrative machine forwards the prescription to the original client.

6. The prescription is also forwarded to other clients in the organization.

7. Subscribers around the world receive regular anti-virus updates that protect
them from the new malware.

Worm Countermeasure Architecture

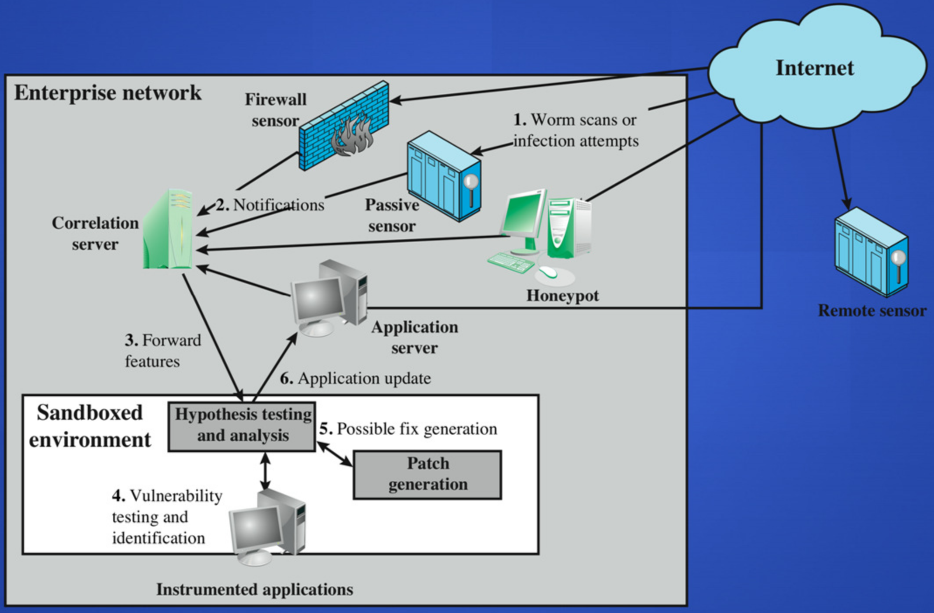


Figure 6.7 Placement of Worm Monitors (based on [SIDI05])

The success of the digital immune system depends on the ability of the malware analysis system to detect new and innovative malware strains. By constantly analyzing
and monitoring malware found in the wild, it should be possible to continually update the digital immune software to keep up with the threat.

This type of functionality may be further augmented by gathering intelligence from perimeter sensors as well. Figure 6.7 shows an example of a worm countermeasure
architecture [SIDI05]. The system works as follows (numbers in figure refer to numbers in the following list):

1. Sensors deployed at various network locations detect a potential worm. The
sensor logic can also be incorporated in IDS sensors.

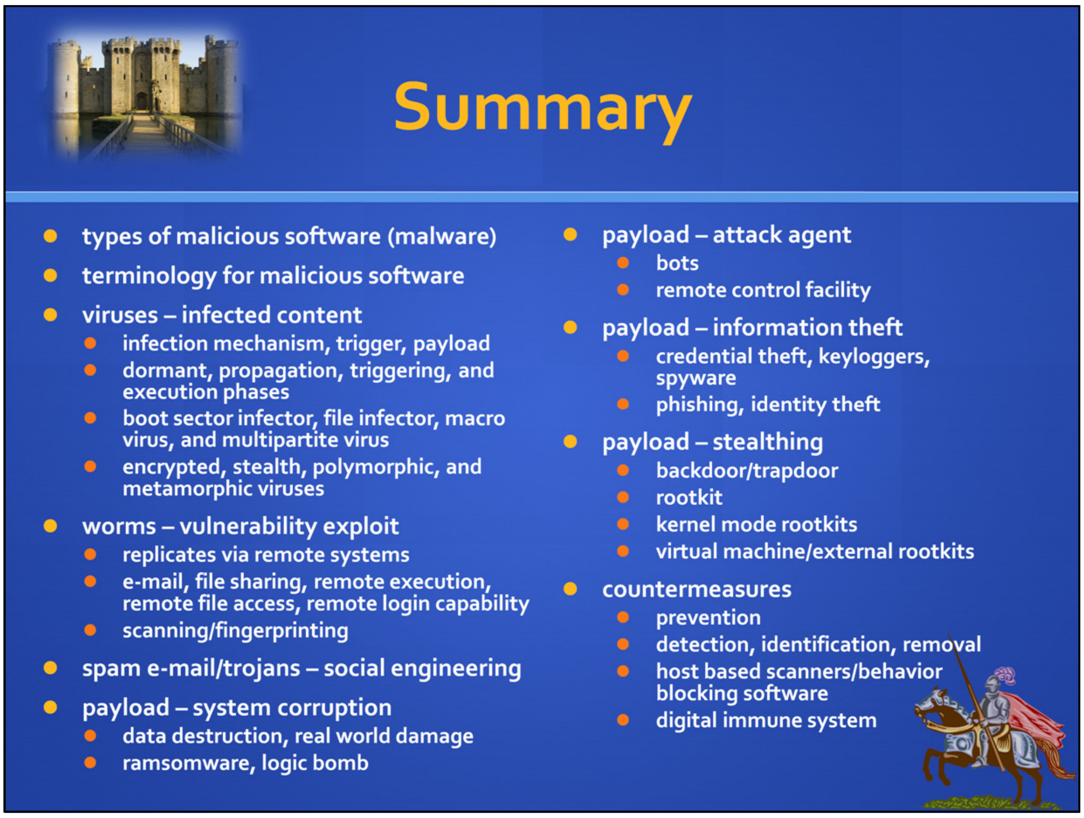
The sensors send alerts to a central server, which correlates and analyzes the incoming alerts. The correlation server determines the likelihood that a worm attack is being observed and the key characteristics of the attack.

3. The server forwards its information to a protected environment, where the potential worm may be sandboxed for analysis and testing.

4. The protected system tests the suspicious software against an appropriately
instrumented version of the targeted application to identify the vulnerability.

5. The protected system generates one or more software patches and tests these.

6. If the patch is not susceptible to the infection and does not compromise the
application’s functionality, the system sends the patch to the application host
to update the targeted application.



Summary

- types of malicious software (malware)
- terminology for malicious software
- viruses – infected content
  - infection mechanism, trigger, payload
  - dormant, propagation, triggering, and execution phases
  - boot sector infector, file infector, macro virus, and multipartite virus
  - encrypted, stealth, polymorphic, and metamorphic viruses
- worms – vulnerability exploit
  - replicates via remote systems
  - e-mail, file sharing, remote execution, remote file access, remote login capability
  - scanning/fingerprinting
- spam e-mail/trojans – social engineering
- payload – system corruption
  - data destruction, real world damage
  - ramsomware, logic bomb
- payload – attack agent
  - bots
  - remote control facility
- payload – information theft
  - credential theft, keyloggers, spyware
  - phishing, identity theft
- payload – stealthing
  - backdoor/trapdoor
  - rootkit
  - kernel mode rootkits
  - virtual machine/external rootkits
- countermeasures
  - prevention
  - detection, identification, removal
  - host based scanners/behavior blocking software
  - digital immune system

Chapter 6 summary.