Information Security
CENG418
week-9

- Malwares and Software Security

- Malwares

Malware Taxonomy

- Software that has a malicious purpose is called *malware*.
- There are different types of malware:
- A ***computer virus*** is a piece of self-replicating code attached to some other piece of code, with a ***payload***.
  - The payload can range from the non-existent via the harmless, e.g. displaying a message or playing a tune, to the harmful, e.g. deleting and modifying files.
  - A computer virus ***infects*** a program by inserting itself into the program code.
- A ***worm*** is a replicating but not infecting program.

Malware Taxonomy

- A ***Trojan horse*** is a program with hidden side effects that are not specified in the program documentation and are not intended by the user executing the program.
- A ***logic bomb*** is a program that is only executed when a specific trigger condition is met.

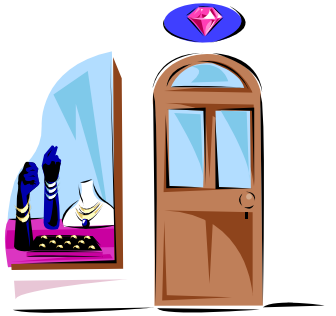
Malware Taxonomy

- In history, a **hacker** was an expert who could use the system in a way beyond the grasp of ordinary users.
- Over time, **the term hacker describes** a person who illicitly breaks into computer systems.

Malware Taxonomy

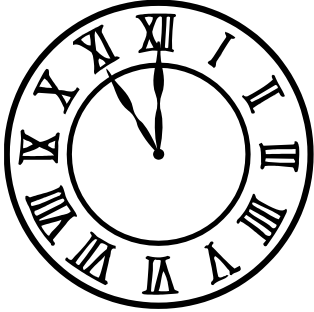
- There is a distinction between **white hats** that use their skills to help software developers and **black hats** that break into systems.
- Anti-virus researchers separate experimental (physically) from operational systems.
  - Otherwise, there is the danger of code escaping out of the laboratory into the wild.
- **Distributing code that performs actions on other people's machines is a crime and brings you into conflict with the law.**

Trapdoor



- Secret entry point into a system
  - Specific user identifier or password that circumvents normal security procedures.
- Commonly used by developers
  - Could be included in a compiler.

Logic Bomb



- Embedded in legitimate programs
- Activated when specified conditions met
  - E.g., presence/absence of some file; Particular date/time or particular user
- When triggered, typically damages system
  - Modify/delete files/disks

Example of Logic Bomb

- In 1982, the Trans-Siberian Pipeline incident occurred.
- A KGB operative was to steal the plans for a sophisticated control system and its software from a Canadian firm, for use on their Siberian pipeline.
- The CIA was tipped off by documents in the Farewell Dossier and had the company insert a logic bomb in the program for sabotage purposes.
- This eventually resulted in "the most monumental non-nuclear explosion and fire ever seen from space“.

Trojan Horse

- Program with an overt (expected) and covert effect
  - Appears normal/expected
  - Covert effect violates security policy
- User tricked into executing Trojan horse
  - Expects (and sees) overt behavior
  - Covert effect performed with user's authorization

Example: Attacker:

Place the following file

cp /bin/sh /tmp/.xxsh

chmod u+s,o+x /tmp/.xxsh

rm ./ls

ls $*

as /homes/victim/ls

- *Victim*

ls

Virus

- Self-replicating code
  - Like replicating Trojan horse
  - Alters normal code with “infected” version
- No *overt* action
  - Generally tries to remain undetected
- Operates when infected code executed

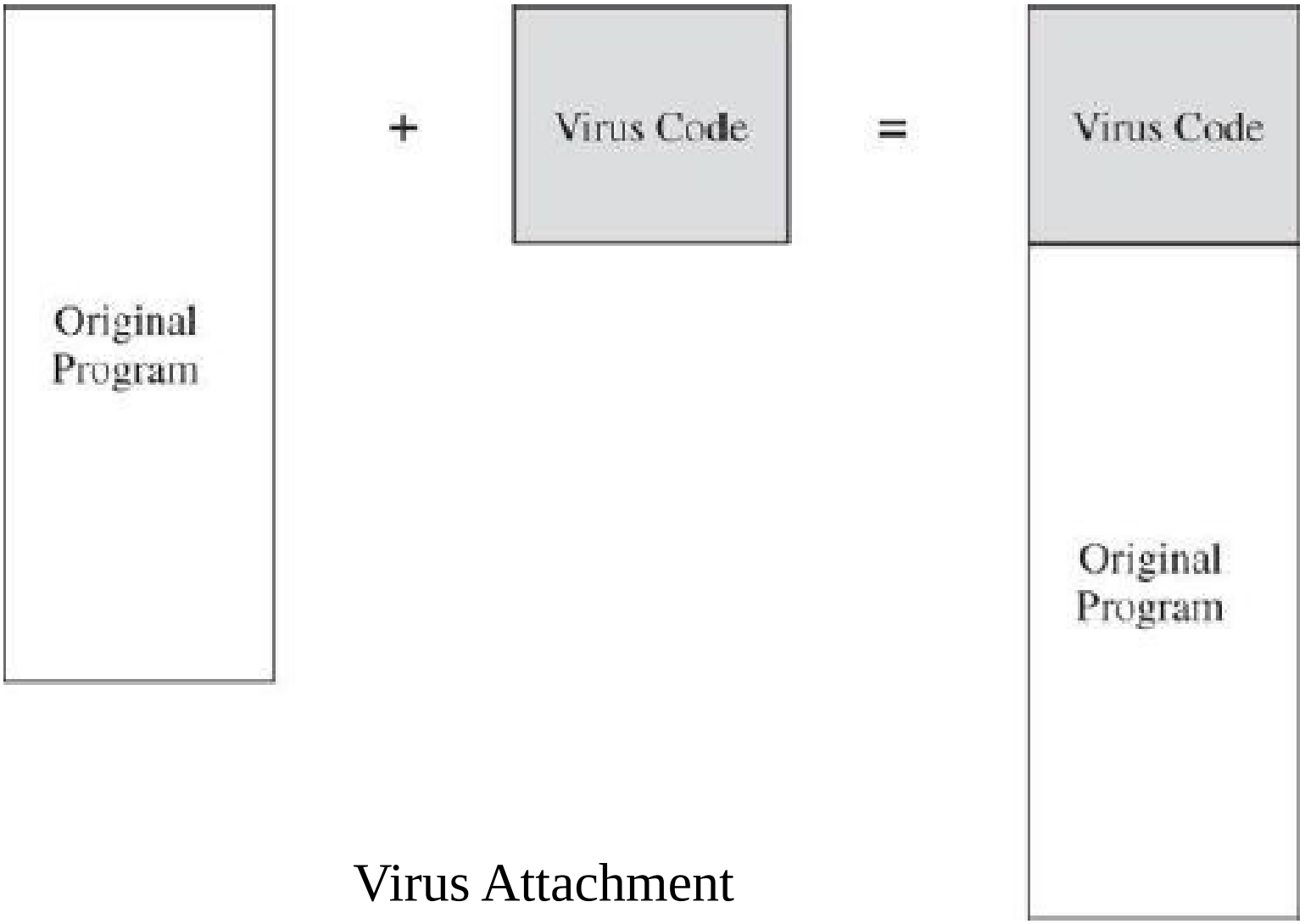
If *spread condition* then

For *target files*

if *not infected* then *alter to include virus*

Perform malicious action

Execute normal program



Virus Attachment

Virus Infection Vectors

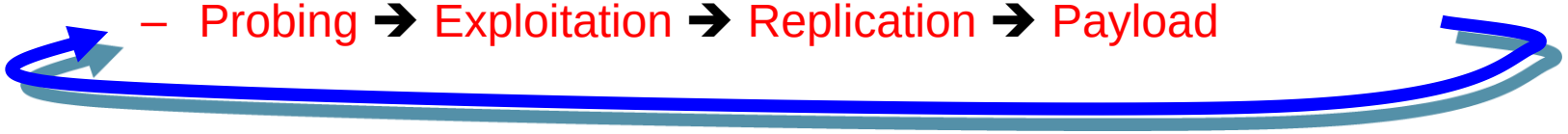
- Boot Sector (USB drives)
  - Problem: How to ensure the virus “carrier” is executed?
  - Solution: Place in boot sector of disk
  - Run on any boot
  - Similar concepts now being used for flash drive
- Executable
  - Malicious code placed at beginning of the legitimate program
  - Runs when application run
  - The application then runs normally
- Macro files
  - Relies on something “executed” inside application data
  - Architecture-independent
  - Application-dependent

Virus Properties

- Terminate and Stay Resident
  - Stays active in memory after application complete
  - Allows infection of previously unknown files
    - Trap calls that execute a program
- Stealth
  - Conceal Infection
    - Trap read and disinfect
    - Let execute call infected file
  - Encrypt virus
    - Prevents “signature” to detect virus
  - Polymorphism
    - Change virus code to prevent signature

Worm

- Runs independently
  - Does not require a host program
- Propagates a fully working version of itself to other machines
- Carries a payload performing hidden tasks
  - Backdoors, spam relays, DDoS agents; …
- Phases
  - Probing ➔ Exploitation ➔ Replication ➔ Payload



Examples of Worm attacks

- Morris worm, 1988
  - Exploits buffer overflow in fingerd, and other vulnerabilities
  - Infected approximately 6,000 machines
    - 10% of computers connected to the Internet
  - cost ~ $10 million in downtime and cleanup
- Code Red I & II worms, 2001
  - Direct descendant of Morris' worm; Exploit buffer overflow in IIS
  - Infected more than 500,000 servers
  - Caused ~ $2.6 Billion in damages,

More Examples of Worm Attacks

- Nimda Worm (2001) Fast spreading
  - Uses five different ways to propagate
    - Including using backdoors left by other worms
- SQL Slammer (2003) Fast spreading
  - Exploits Microsoft SQL server
  - Infects 75,000 hosts within 10 minutes
- Conficker (2008,2009) Evolving &
  - Exploits Windows server service (and other vectors in variants)
  - Infects between 9 and 15 million computers
  - Evolver, persists, self-update, and eventually install a spambot & a scareware

Email Worms: Spreading as Email Attachments

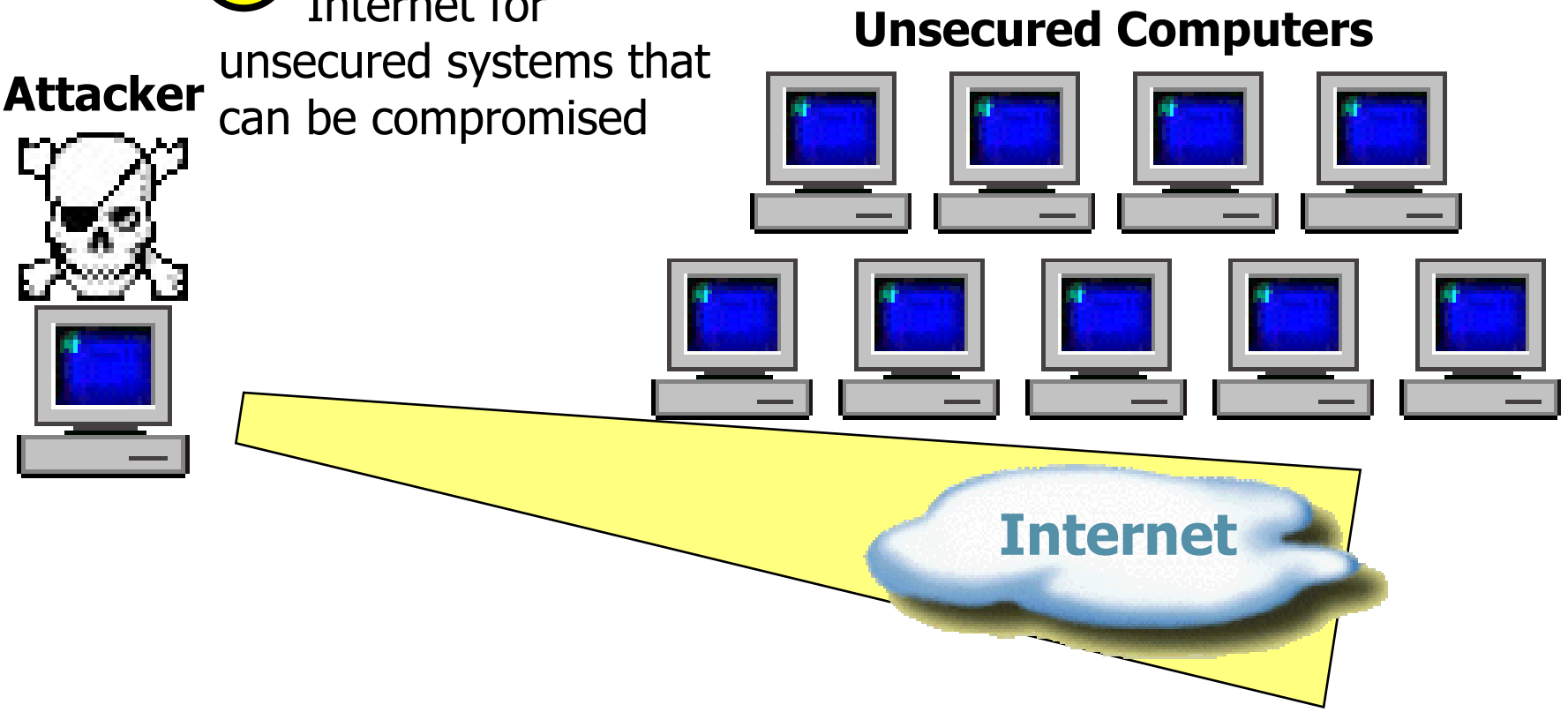
- Love Bug worm (ILOVEYOU worm) (2000):
  - May 3, 2000: 5.5 to 10 billion dollars in damage
- MyDoom worm (2004)
  - First identified in 26 January 2004:
  - On 1 February 2004, about 1 million computers infected with Mydoom begin a massive DDoS attack against the SCO group
- Storm worm & Storm botnet (2007)
  - Identified on January 17
  - gets its name from the subjects of initial emails “230 dead as storm batters Europe”
  - gathering infected computers into the Storm botnet.
  - By around June 30th infected 1.7 million computers,
  - By September, has between 1 and 10 million bots

Zombie & Botnet

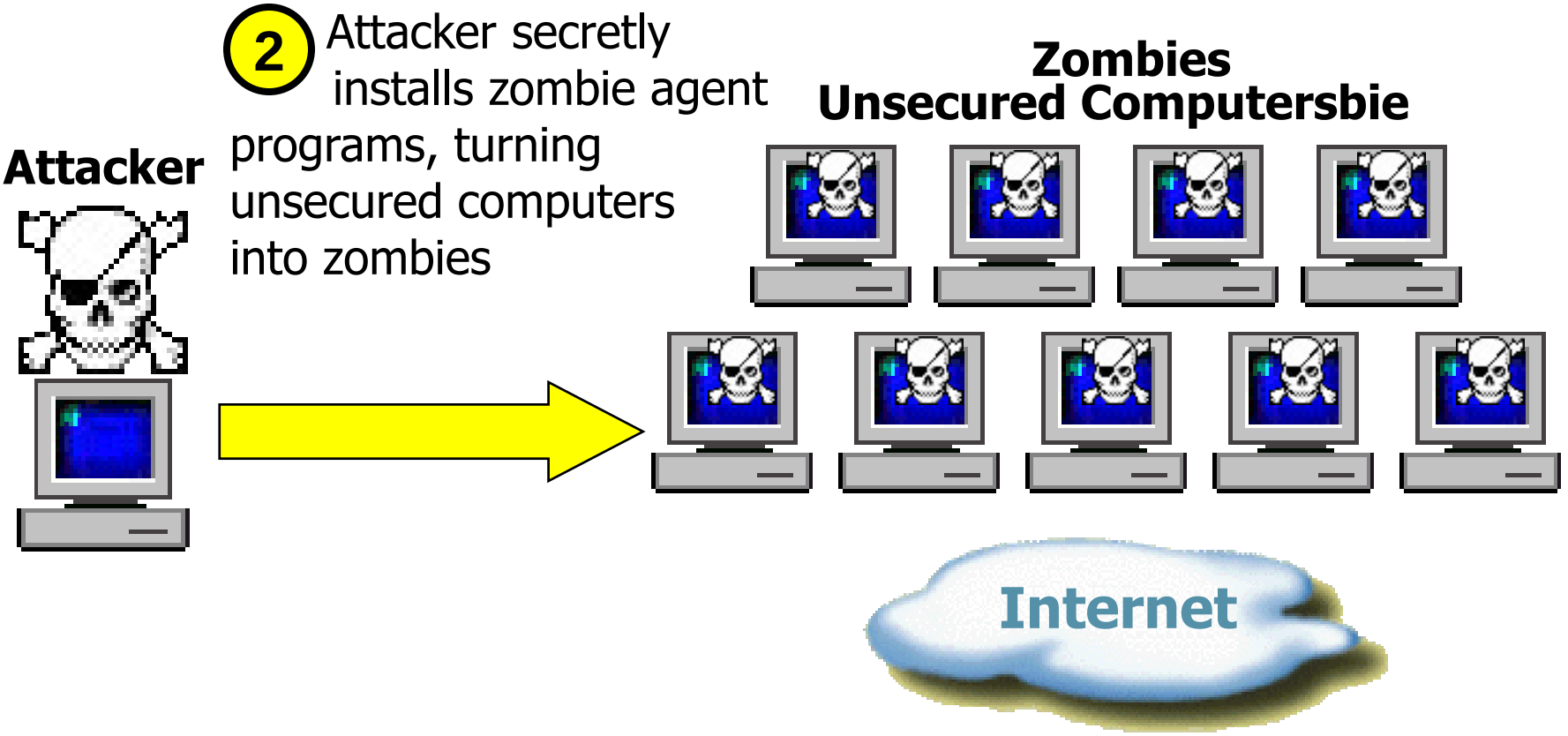
- Secretly takes over another networked computer by exploiting software flows
- Builds the compromised computers into a zombie network or botnet
  - A collection of compromised machines running programs, usually referred to as worms, Trojan horses, or backdoors, under a joint command and control infrastructure.
  - Uses it to indirectly launch attacks
  - E.g., DDoS, phishing, spamming, cracking

Detailed Steps (1)

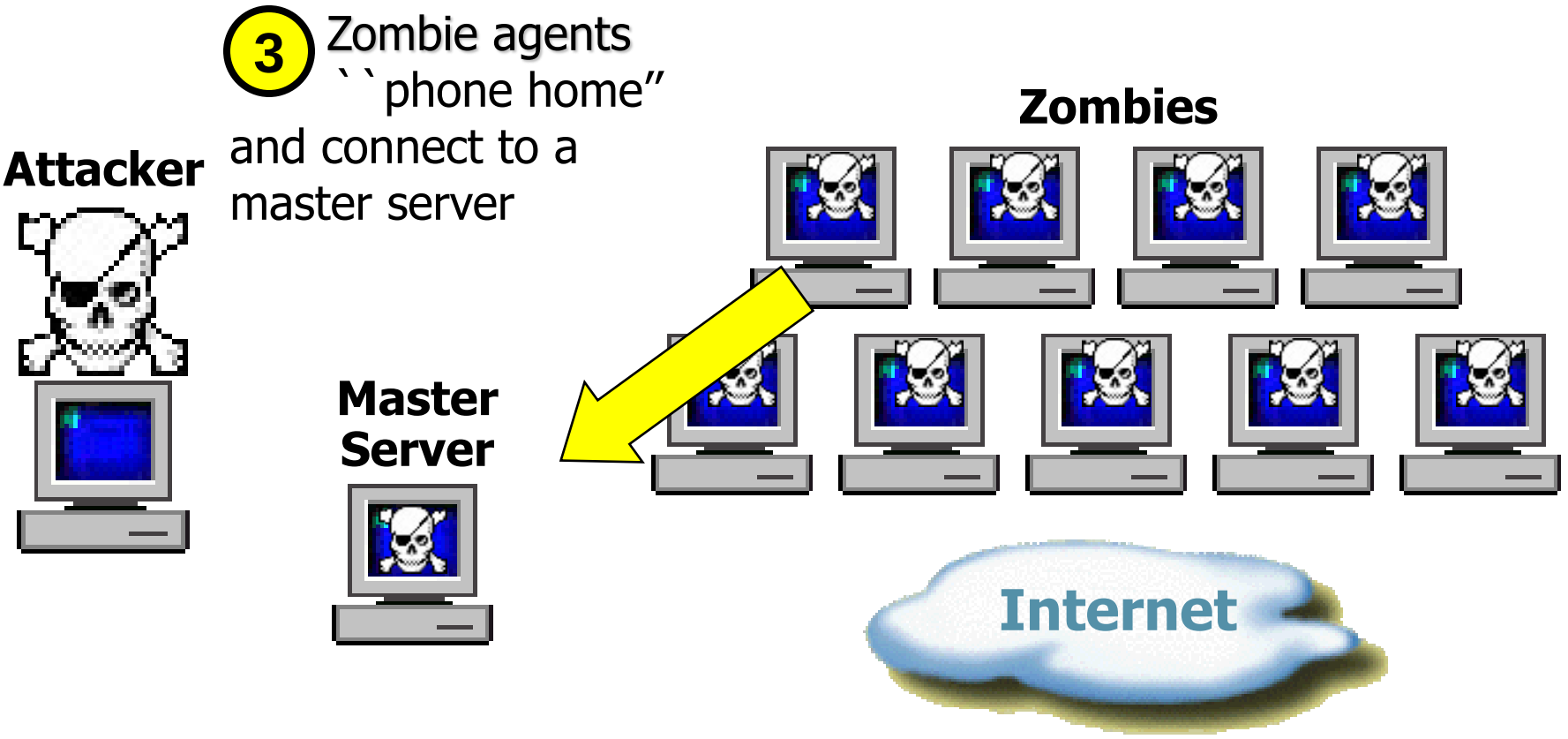
1 Attacker scans Internet for unsecured systems that can be compromised



Detailed Steps (2)

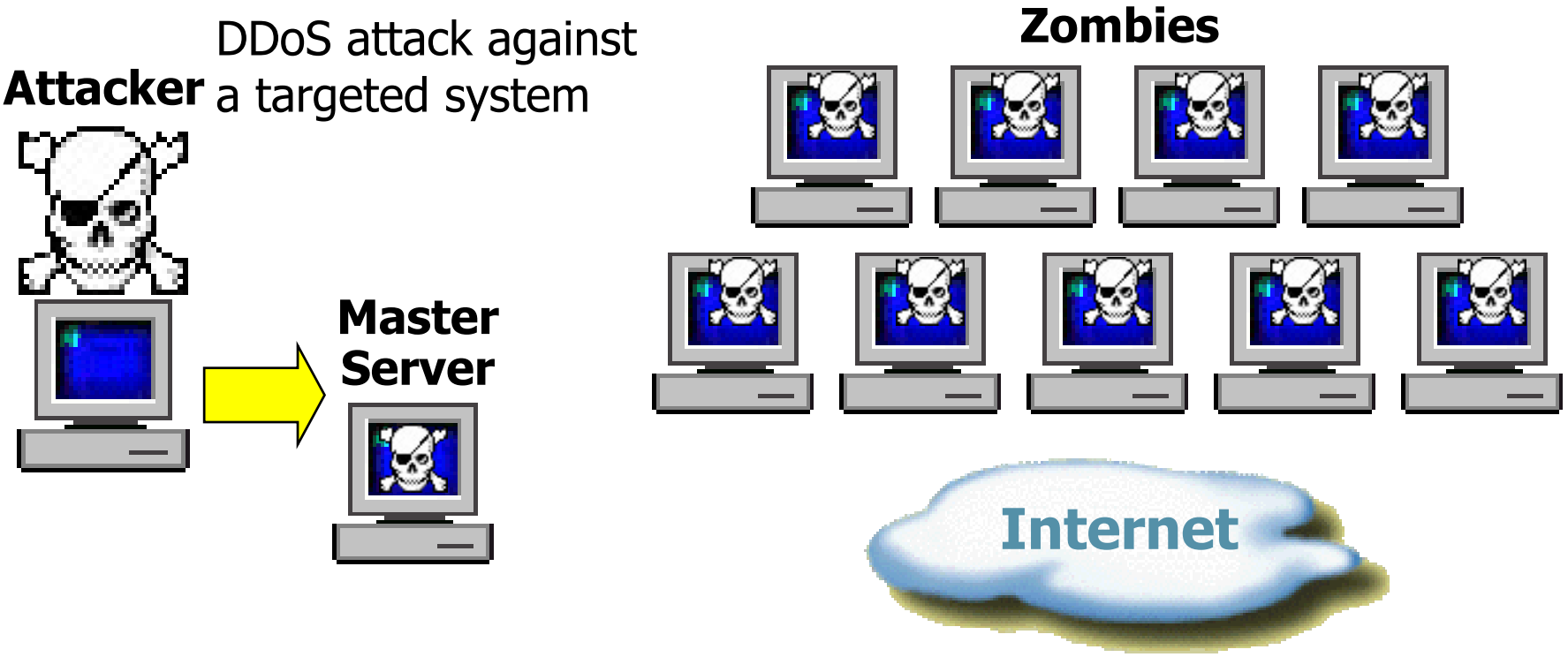


Detailed Steps (3)

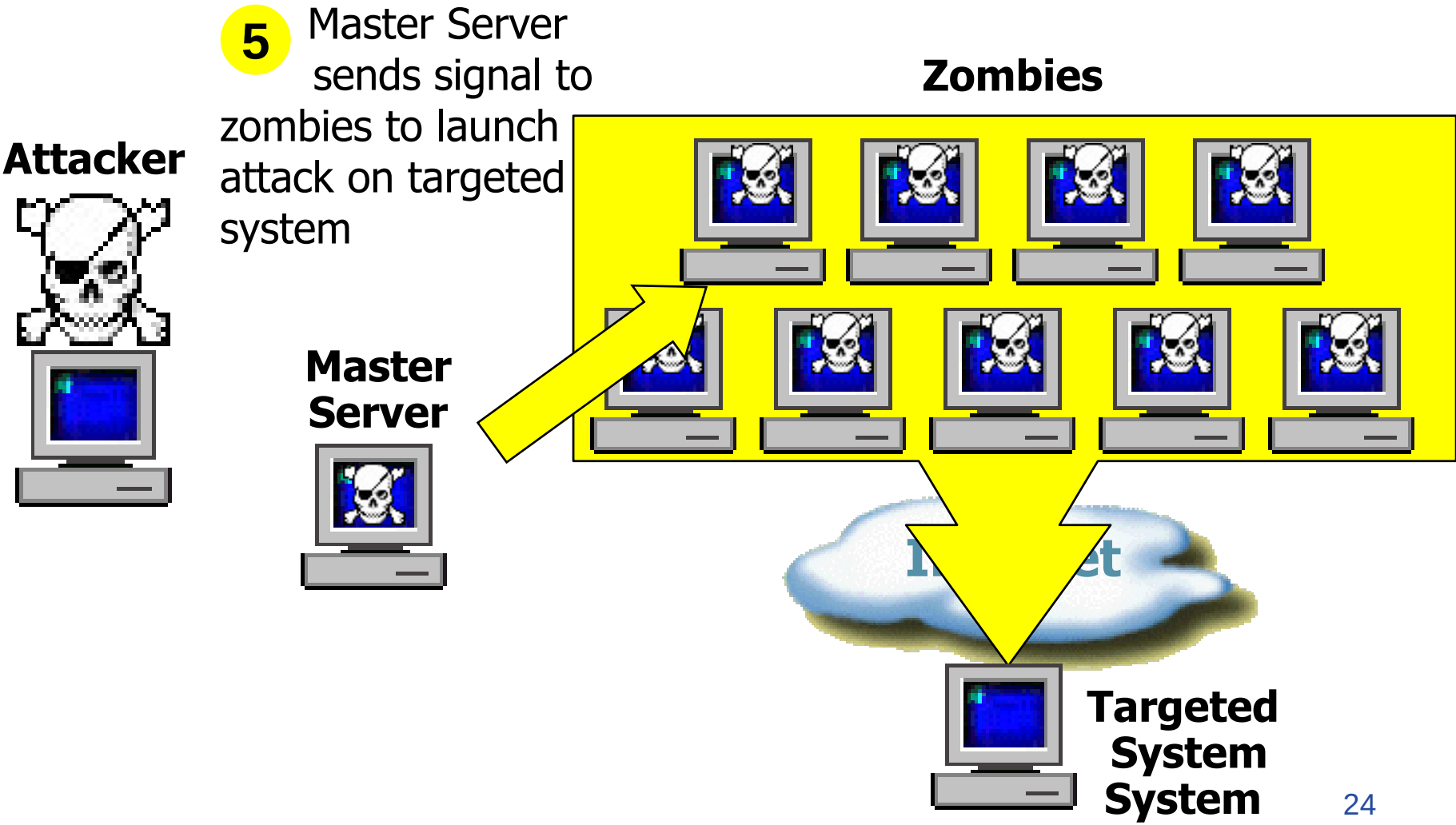


Detailed Steps (4)

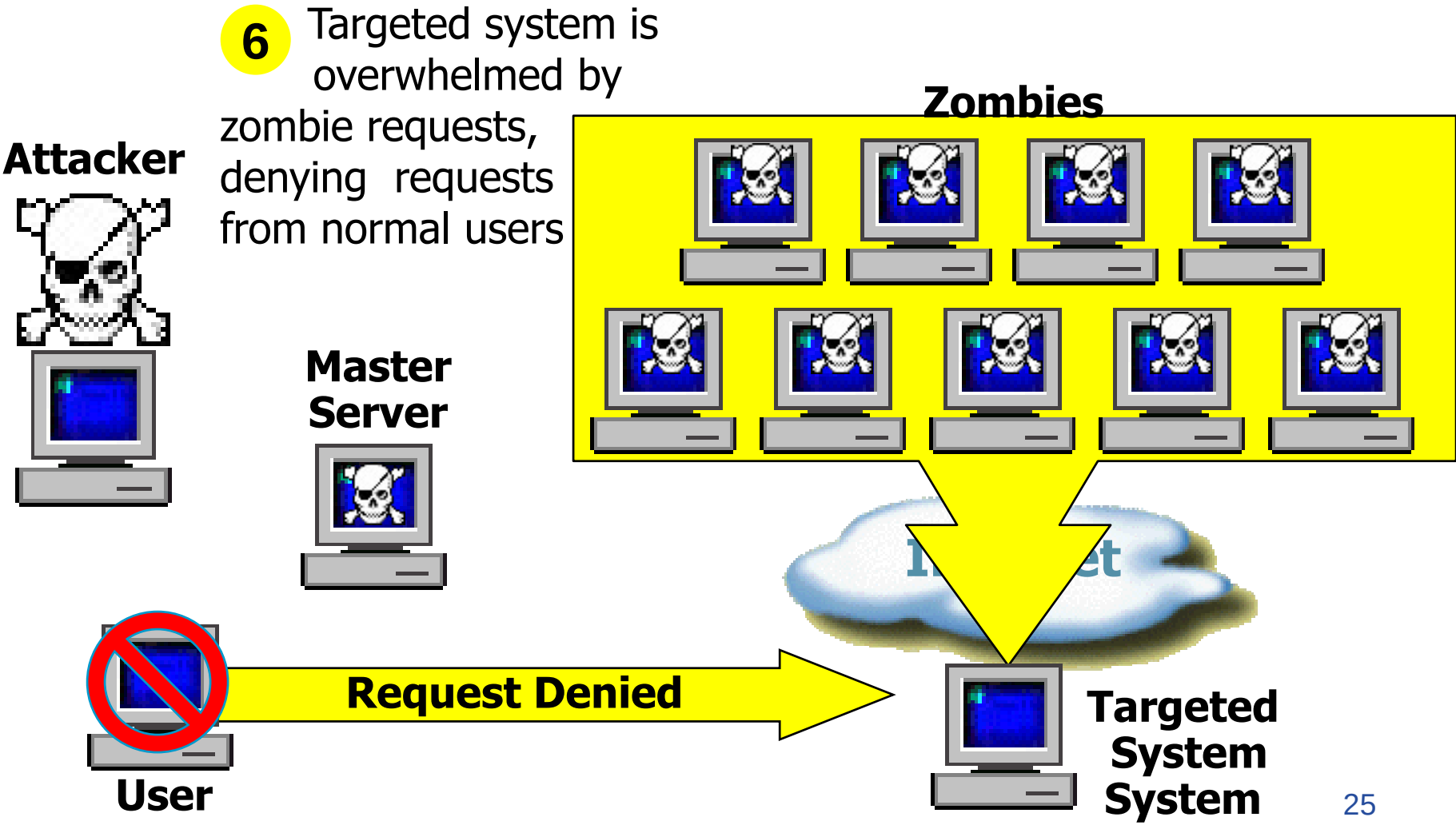
4 Attacker sends commands to Master Server to launch a DDoS attack against a targeted system



Detailed Steps (5)



Detailed Steps (6)



Botnet

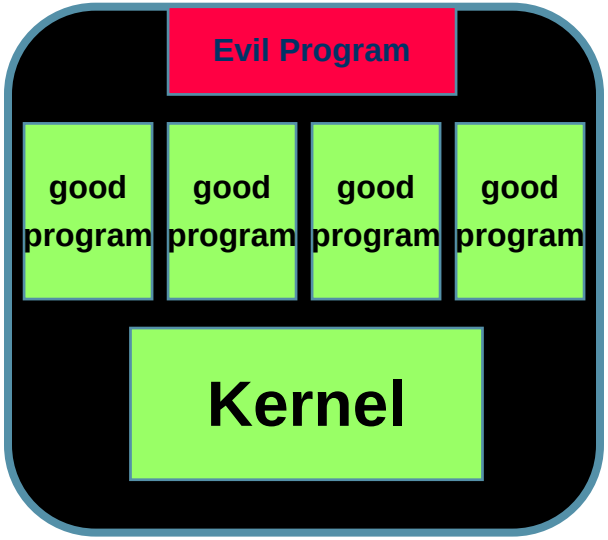
- Using peer-to-peer structure, rather than a central command & control
- Encrypting/authenticating communications

Rootkit

- Software used after system compromise to:
  - Hide the attacker's presence
  - Provide backdoors for easy reentry
- Simple rootkits:
  - Modify user programs (ls, ps)
  - Detectable by tools like Tripwire
- Sophisticated rootkits:
  - Modify the kernel itself
  - Hard to detect from userland

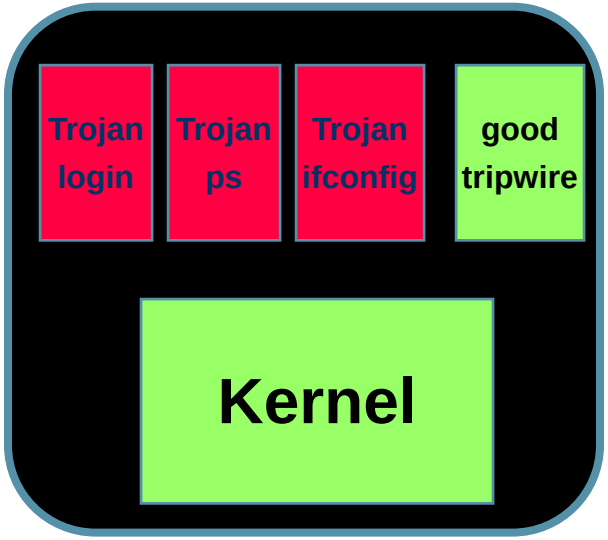
Rootkit Classification

Application-level Rootkit



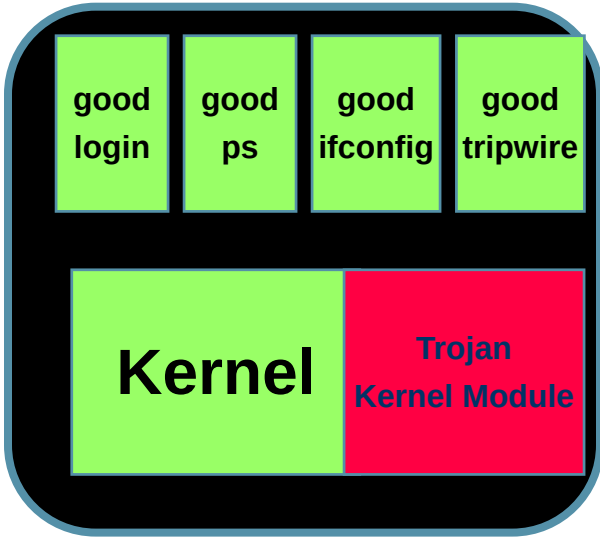
Hxdef, NTIllusion

Traditional RootKit



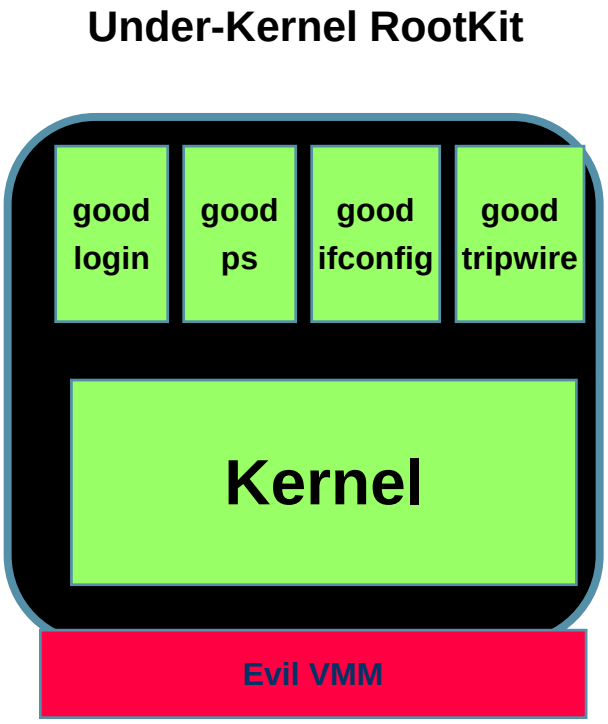
Lrk5, t0rn

Kernel-level RootKit



Shadow Walker, adore

Rootkit Classification



SubVirt, ``Blue Pill''

Hypervisor
Hardware/firmware

Spyware

- Malware that collects little bits of information at a time about users without their knowledge
  - Keyloggers:
  - May also tracking browsing habit
  - May also re-direct browsing and display ads
- Typically do not self-propagate

Scareware

- Software
  - Malicious programs designed to trick a user into buying and downloading unnecessary and potentially dangerous software, such as fake antivirus protection.
  - Sold by social engineering to cause shock, anxiety, or the perception of a threat
- Rapidly increasing
  - Anti-Phishing Working Group: # of scareware packages rise from 2,850 to 9,287 in 2nd half of 2008.

SECURITY WARNING!

serious security threat detected

Your computer is infected with Spyware.
Your Security and Privacy are in DANGER.

Spyware programs can steal your credit card numbers and bank information details. The computer can be used for sending spam and you may get popups with adult or any other unwanted content.

If

- *You have visited adult or warez websites during past 3 days.*
- *Your homepage has changed and does not change back.*
- *Your computer performance has dropped down dramatically.*
- *You are suspecting someone is watching you.*

Then your computer is most likely
INFECTED WITH SPYWARE.

We are sorry, but the trial version is unable to remove these threats.
We strongly recommend you to purchase Full version.

You will get 24x7 friendly support and unlimited protection.

Continue Unprotected

Get Full version of SpyShelll Now!

Ransomware

- Holds a computer system, or the data it contains, hostage against its user by demanding a ransom.
  - Disable an essential system service or lock the display at system startup
  - Encrypt some of the user's personal files, originally referred to as **cryptoviruses**, **cryptotrojans** or **cryptoworms**
- Victim user has to
  - enter a code obtainable only after wiring payment to the attacker or sending an SMS message
  - buy a decryption or removal tool

Countermeasures for Users

- *Use only commercial software acquired from reliable, well-established vendors*.
- *Test all new software on an isolated computer*.
- *Open attachments—and other potentially infected data files—only when you know them to be safe*.
- *Install software—and other potentially infected executable code files—only when you really, really know them to be safe.*
- *Recognize that any web site can be potentially harmful.*
- *Make a recoverable system image and store it safely.*
- *Make and retain backup copies of executable system files.*
- *Virus detectors are powerful but not all-powerful.*
- *Virus writers and antivirus tool makers engage in a battle to conceal patterns and find those regularities.*

- Software Security

How does a computer get infected with malware or being intruded?

- Executes malicious code via user actions (email attachment, download and execute Trojan horses)
- Buggy programs accept malicious input
  - daemon programs that receive network traffic
  - client programs (e.g., web browser, mail client) that receive input data from network
  - Programs Read malicious files with buggy file reader program
- Configuration errors (e.g., weak passwords, guest accounts, DEBUG options, etc)
- Physical access to computer

Why Software Has (or appear to have) So Many Bugs?

- Software is complicated, and created by human
- Software is no more buggy, is just more targeted?
- Unique nature of software
  - Near-zero marginal cost
- Market failure for secure software
  - Market failure: a scenario in which individuals' pursuit of self-interest leads to bad results for society as a whole
  - Users cannot just vote for security with their money.
    - lack of measurement for security
  - Vendor has no incentives to produce higher quality software.

Guy Kawasaki: "**The Art of Innovation**"

- Don't worry, be crappy.
  - An innovator doesn't worry about shipping an innovative product with elements of crappiness if it's truly innovative.
- Churn, baby, churn.
  - I'm saying it's okay to ship crap--I'm not saying that it's okay to stay crappy. A company must improve version 1.0 and create version 1.1, 1.2, ... 2.0.

Why Vendors Lack Incentive to Produce More Secure Software

- Cash flows when product starts shipping.
- Market dominance is key to success
  - being first often means becoming de facto standard
- No liability.
- Bugs can be patched with little cost. No expensive recall.
- Thorough testing is inefficient. Let the users test it and fix only the bugs that affect users

The Perversity of Patching

- Releasing a patch costs little
- Buggy software can force users to upgrade
  - Achieving market dominance means competing with previous versions
  - Stop releasing patches for old versions can force users to upgrade
- Patching provide an opportunity of offering new licensing terms

Software Vulnerabilities

- Attacks may target **memory management flaws** using a *buffer overrun* to manipulate the control flow at a layer below the programming abstractions used by the software developer.
- Attacks may target applications written in a *scripting language* and insert their commands via user input.
- In both cases, the attacker manages to execute code with elevated privileges.

- This chapter will analyze the causes of software vulnerabilities and defense options at a general level.

Security and Reliability

- Software security is related to software quality and software reliability.
- Reliability deals with accidental failures.
  - Failures are assumed to occur according to some given probability distribution.
  - Improvements in reliability can be calculated based on this distribution.
  - ‘It does not matter how many bugs there are, it matters how often they are triggered.’
- In security the attacker picks the distribution of the inputs.
  - It has to be tested against a typical usage patterns (but there are typical attack patterns).

Change in Environment

- Change is one of the biggest enemies of security.
- You can assume that you have a system that offers perfectly adequate security.
- But, you change a part of the system. You may not be aware of the security implications of this change.

Dangers of Abstraction

- **Abstraction** is an important concept we cannot do without designing and understanding complex systems.
- High-level descriptions of a system hide details.
- Software developers use abstractions when writing code in high-level programming languages.
- **Software security problems arise when the intuitive properties of abstraction do not match its concrete implementation.**

Characters and Numbers

- Characters and integers; are described with binary representation in memory or in messages by use hexadecimal values.
- Hexadecimal values in C have the prefix 0x;
- Hexadecimal values in a URL have the prefix %.

Lesson

Beware of mistranslations that change the meaning of texts. Decoding UTF-8 is a translation between different levels of abstraction.

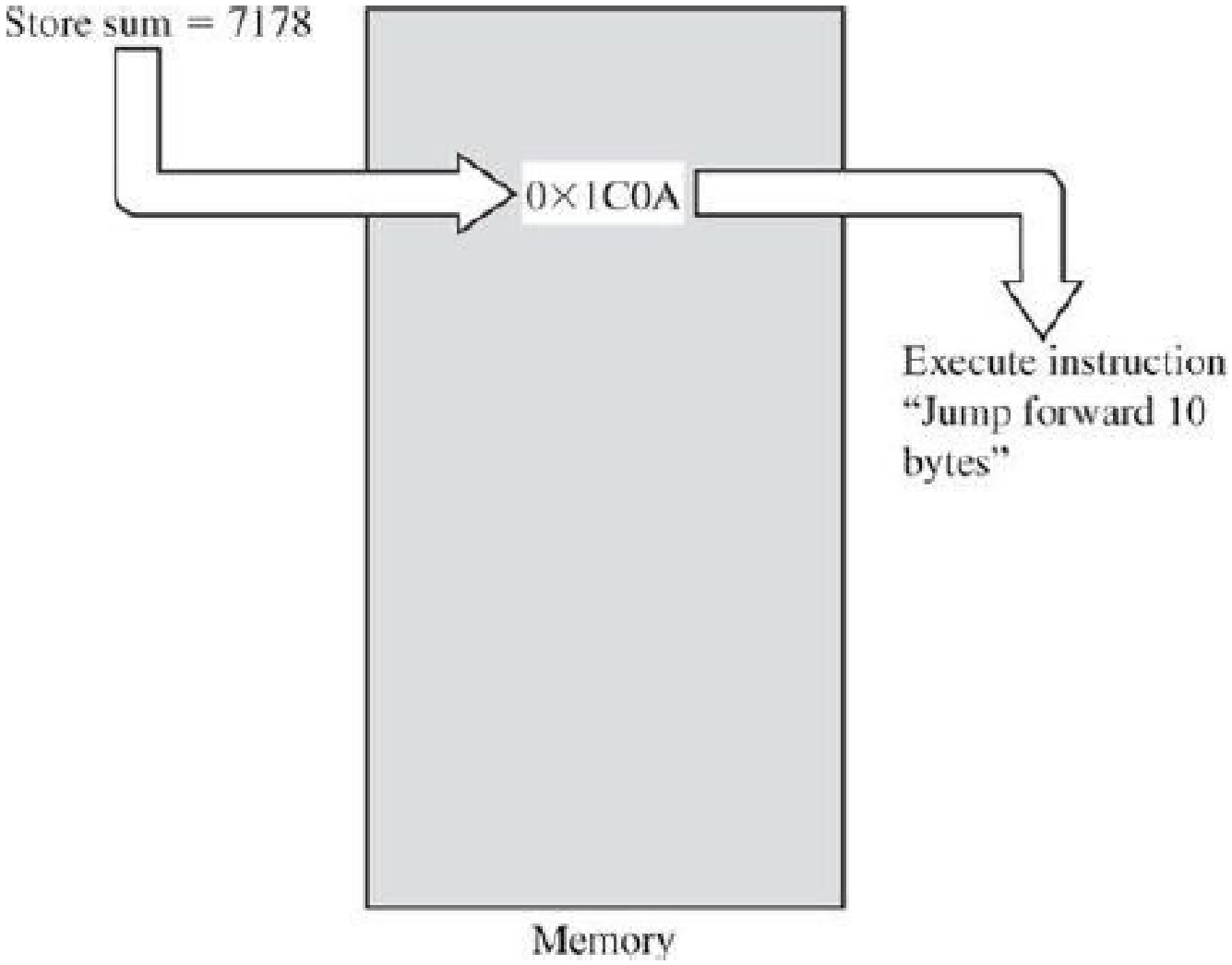
Buffer Overflow

- Buffer overflows often come from innocent programmer oversights or failures to document and check for excessive data.
- A string overruns its assigned space or one extra element is added into an array; what's the big deal, you ask?
- To understand why buffer overflows are a major security issue, you need to understand how an operating system stores code and data.

Buffer Overflow

- Code, data, instructions, the operating system, complex data structures, user programs, strings, downloaded utility routines, hexadecimal data, decimal data, character strings, code libraries, photos, and everything else in memory are just strings of 0s and 1s
- An Add instruction implies the data item is interpreted as a number, a Jump instruction assumes the target is an instruction.
- But at the machine level, nothing prevents a Jump instruction from transferring into a data field or an Add command operating on an instruction, although the results may be unpleasant. Code and data are bit strings.

Buffer Overflow



Bit Patterns Can Represent Data or Instructions

Buffer Overflow – Example 1
Microsoft Dialer program, dialer.exe

- Many people shared one line between voice and computer (data) communication.
- Microsoft provided Dialer, a simple utility program to dial a number with the modem.
- As of 2014, it was still part of Windows 10, although the buffer overflow described here was patched shortly after David reported it.
- Dialer had to accept phone numbers of different lengths, given country variations, outgoing access codes, and remote signals.
  - David suspected there would be an upper limit and tried dialer.exe with a 20-digit phone number and 25 and 50, and the program still worked fine.
  - When he tried a 100-digit phone number, the program crashed.
- The programmer had probably made an undocumented and untested decision that nobody would ever try to dial a 100-digit phone number … except David.

Buffer Overflow – Example 2

- An alternative style of buffer overflow occurs when parameter values are passed into a routine, especially when the parameters are passed to a web server on the Internet.
- Parameters are passed in the URL line, with a syntax similar to:

  http://www.somesite.com/subpage/userinput.asp?

  parm1=(808)555-1212
- The application script user input receives one parameter, parm1 with value (808)555-1212.
- The web browser on the caller's machine will accept values from a user who probably completes fields on a form.
- The browser encodes those values and transmits them back to the server's web site.
- Passing a very long string to a web server is a slight variation on the classic buffer overflow, but no less effective.

Overwriting Memory

- If you write an element past the end of an array or you store an 11-byte string in a 10-byte area, that extra data has to go somewhere; often it goes immediately after the last assigned space for the data.
- The memory is finite, a buffer's capacity is finite. For this reason, in many programming languages the programmer must declare the buffer's maximum size so that the compiler can set aside that amount of space.

Overwriting Memory - Example

The compiler sets aside 10 bytes to store this buffer. `char sample[10];`

We execute the statement `sample[10] = 'B';`

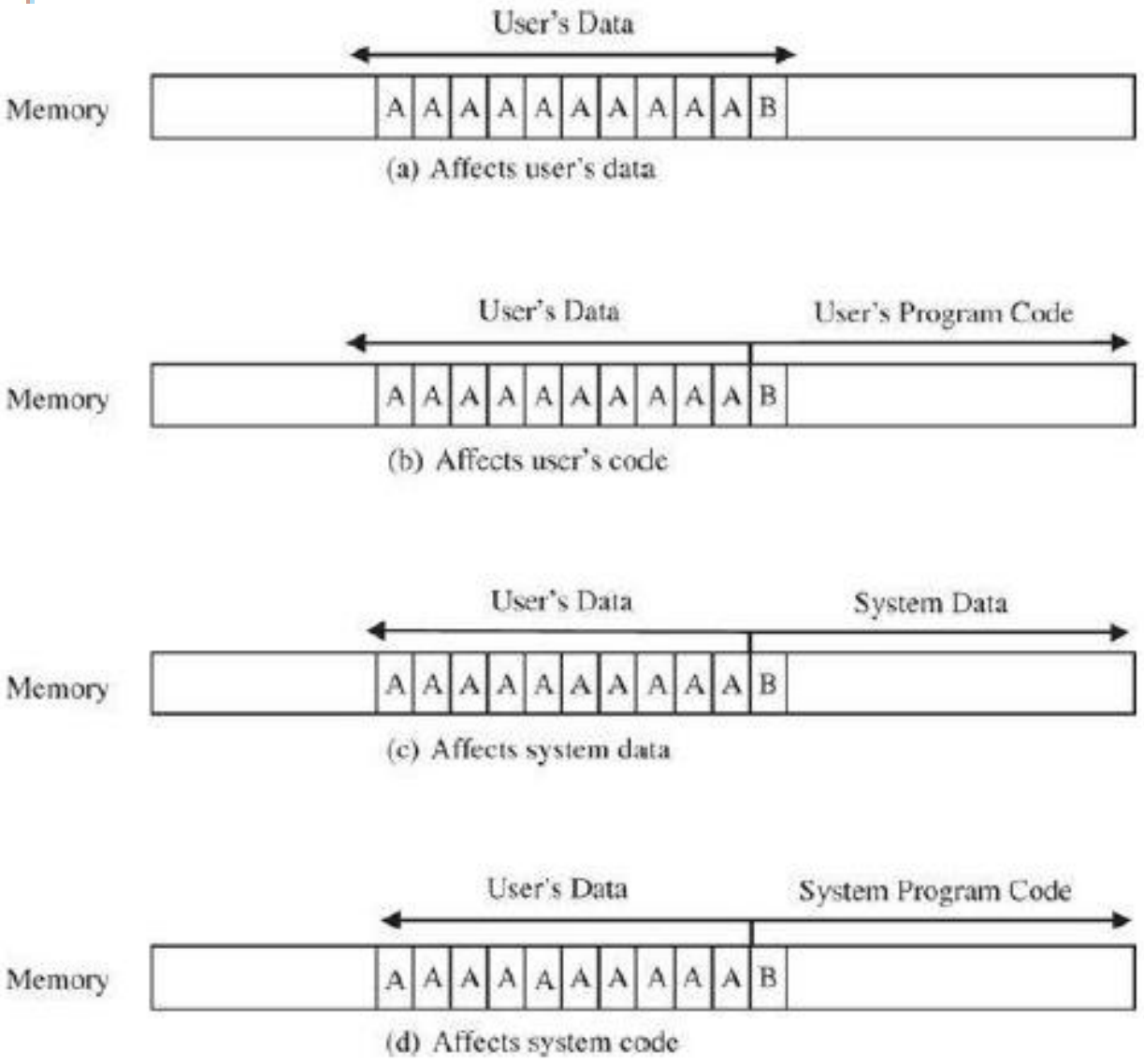
The subscript is out of bounds (that is, it does not fall between 0 and 9), so we have a problem. The nicest outcome (from a security perspective) is for the compiler to detect the problem and mark the error during compilation.

if the statement were `sample[i] = 'B';`

the compiler could not identify the problem until i was set during execution either to a proper value (between 0 and 9) or to an out-of-bounds subscript (less than 0 or greater than 9).

Even if the compiler were careful in analyzing the buffer declaration and use, this same problem can be caused with pointers, for which there is no reasonable way to define a proper limit.

```
for (i=0; i<=9; i++)
      sample[i] = 'A';
sample[10] = 'B'
```



Overflow Countermeasures

- The most obvious countermeasure to overwriting memory is to stay within bounds.
- Maintaining boundaries is a shared responsibility of the programmer, operating system, compiler, and hardware.
  - Check lengths before writing.
  - Confirm that array subscripts are within limits.
  - Double-check boundary condition code to catch possible off-by-one errors.
  - Monitor input and accept only as many characters as can be handled.
  - Use string utilities that transfer only a bounded amount of data.
  - Check procedures that might overrun their space.
  - Limit programs' privileges, so if a piece of code is overtaken maliciously, the violator does not acquire elevated system privileges as part of the compromise.

Code Analyzers

- Software developers hope for a simple tool to find security errors in programs.
- Such a tool, called a **static code analyzer**, analyzes source code to detect unsafe conditions.
- Although such tools are not, and can never be, perfect, several good ones exist.

Ref: *The US-CERT Build Security In website a related article: https://www.cisa.gov/uscert/bsi/articles/tools/source-code-analysis/source-code-analysis-tools---overview.*

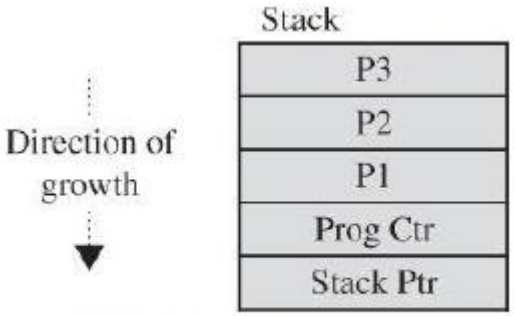


FIGURE 3-7 A Stack Frame

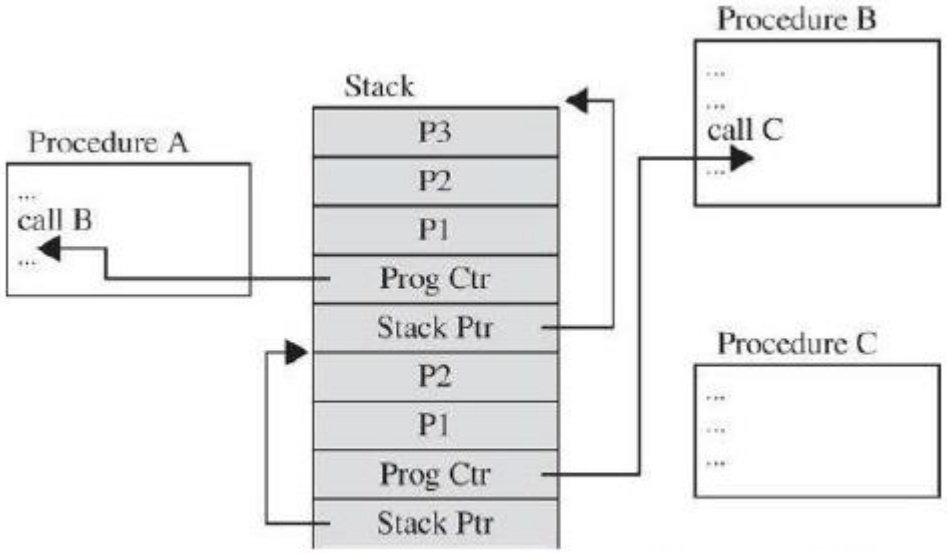
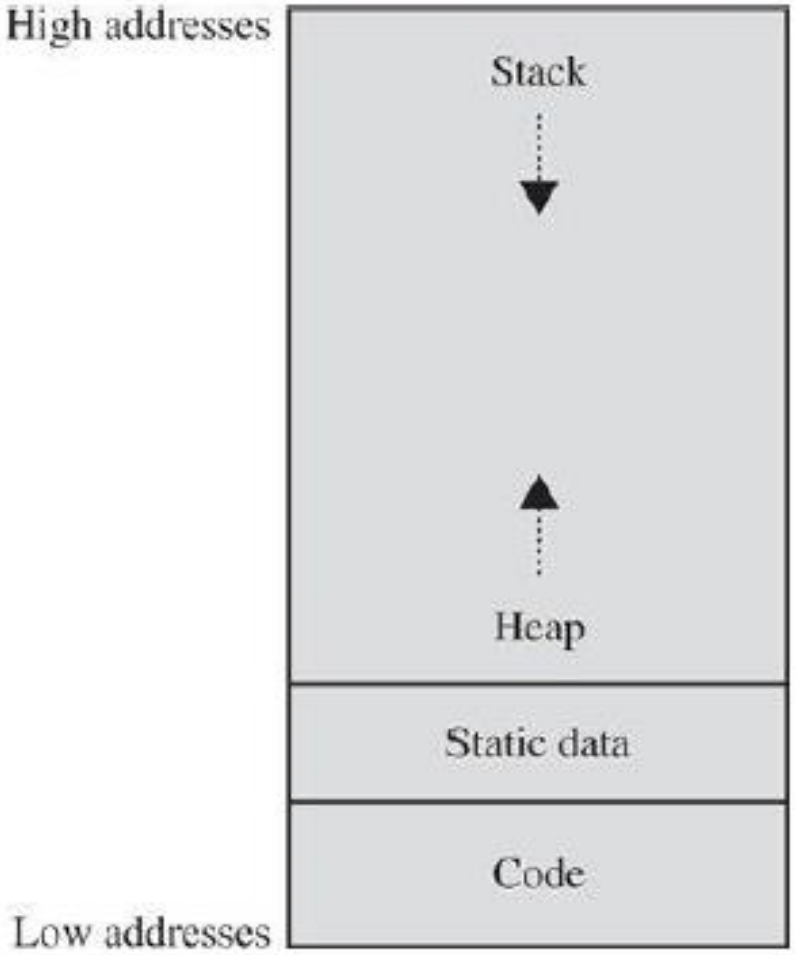


FIGURE 3-9 The Stack after Nested Procedure Calls



Typical Memory Organization

The stack is at the top of memory, growing downward, and something else, called the heap, is at the bottom growing up. As you have just seen, the stack is mainly used for nested calls to procedures. The **heap** provides space for dynamic data, that is, data items whose size is not known when a program is compiled.

- **Overflow into system space can redirect execution immediately or on exit from the current called procedure.**
- The attacker wants to overwrite stack memory, sometimes called **stack smashing**.
- **Overwrite the program counter stored in the stack** so that when this routine exits, control transfers to the address pointed at by the modified program counter address.
- **Overwrite part of the code in low memory**, substituting the attacker's instructions for previous program statements.
- **Overwrite the program counter and data in the stack** so that the program counter now points into the stack, causing the data overwritten into the stack to be executed.

Separation

- Separating sensitive areas from the running code and its buffers and data space. To a certain degree, hardware can separate code from data areas and the operating system.

Incomplete Mediation

- Verifying that the subject is authorized to perform the operation on an object is called **mediation**. Incomplete mediation is a security problem.
- The solution is complete mediation. The standard security tools is access control. These three properties combine to give us solid, complete mediation:

(1) small and simple enough to give confidence of correctness,

(2) unbypassable, and

(3) always invoked.

Validate All Input

- The programmer may have written code to check for correctness on the *client*'s side (that is, the user's browser).
- To prevent the use of nonsense data, the program can restrict choices to valid ones only.

Guard Against Users' Fingers

- The user can edit the URL line, change any parameter values, and send the revised line.
- On the server side, the server has no way to tell if the response line came from the client's browser or as a result of the user's editing the URL directly.
- We say in this case that the data values are not completely mediated: The sensitive data (namely, the parameter values) are in an exposed, uncontrolled condition.

Backdoors

- An undocumented access point is called a **backdoor** or **trapdoor**.
- Such an entry can transfer control to any point with any privileges the programmer wanted.
- An intruder who obtains access to a machine wants to return later.
  - The attacker can create a new account on the compromised machine, under a user name and password that only the attacker knows.
- **Secret backdoors are eventually found. Security cannot depend on such secrecy.**

Software Engineering Techniques for SW-Security

- Simplicity of software design improves correctness and maintainability.
- **Modularity** offers advantages for program development in general and security in particular.
- If a component is isolated from the effects of other components, then the system is designed in a way that limits the damage any fault causes. If the component is isolated. We call this isolation **encapsulation**.
- **Security testing** tries to anticipate the hundreds of ways a program can fail.
  - **module testing**, **component testing**, or **unit testing**, verifies that the component functions properly with the types of input expected from a study of the component's design.