Computer Security

CS433

Chapter 3

Programs Security

Objectives

Learn	Learn about memory organization, buffer overflows, and relevant countermeasures
Introduce	Common programming bugs, such as off-by-one errors, race conditions, and incomplete mediation
Survey	Survey of past malware and malware capabilities
Learn	Learn about Virus detection
Show	Tips for programmers on writing code for security

Terminology

Error Human mistake

Fault Incorrect step caused by an error

- Human error in understanding the requirement results on a fault in the design
- A single error may result on multiple faults

Failure Wrong system behavior

- Dividing by zero
- Failure can be discovered during system testing phase.

Flaw Although there is a distinction between fault and failure, security engineers use this term to refer to both.

- Weakness in the system

Programming Oversights

- ✓ **Three ways to obtain a program:**
  1. Use the **pre-installed** one in your system (e.g. text editor)
  2. Purchase an **over-the-counter** (e.g. MS Word)
  3. **Build** your own
- ✓ **Programs are built and used by human; hence, flaws may appear regularly.**
- ✓ **Program flaws has 2 security implications:**
  1. **Integrity problem**. Harmful output or actions.
     - Program flaw may result into fault the leads to failure.
     - It may also result on modifying, delete, overwriting correct information.
  2. **Exploitation by malicious actors**.
     - When attackers learn the flaw, they can manipulate the behavior of the program.

Programming Oversights

Security failures can result from

- Unintentional Causes
- Intentional Causes

Unintentional Programming Oversights

Buffer Overflow

Time-of-Check to Time-of-Use

Incomplete Mediation

Race Condition

There are also

- ✓ *Undocumented Access Point*
- ✓ *Off-by-One Error*
- ✓ *Integer Overflow*
- ✓ *Unterminated Null-Terminated String*
- ✓ *Parameter Length, Type, and Number*
- ✓ *Unsafe Utility Program*

Buffer Overflow

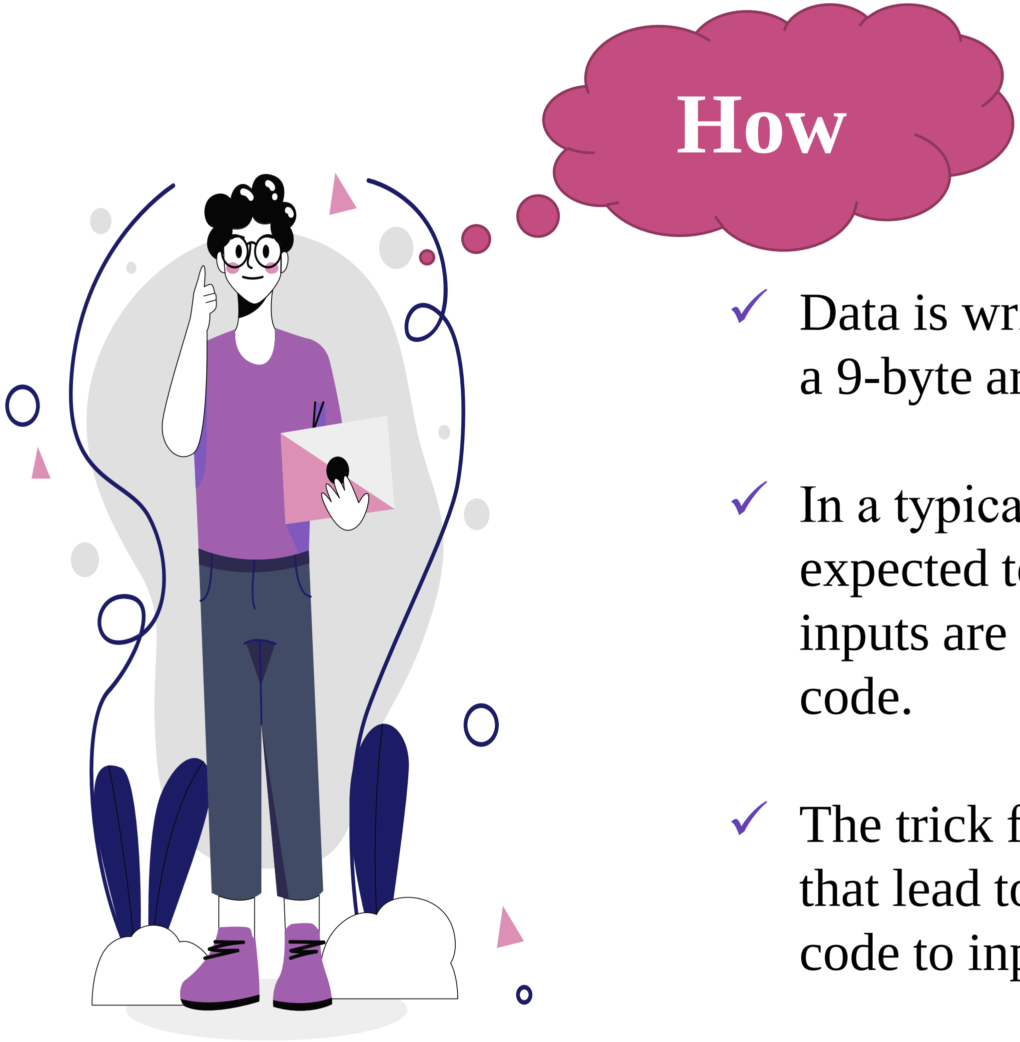
Buffer Overflows

- One of the **most common vulnerabilities** in software
- Particularly **problematic** when present in **system libraries** and other code that runs with **high execution privileges**
- It is an example on **boundary condition violation**

Definition

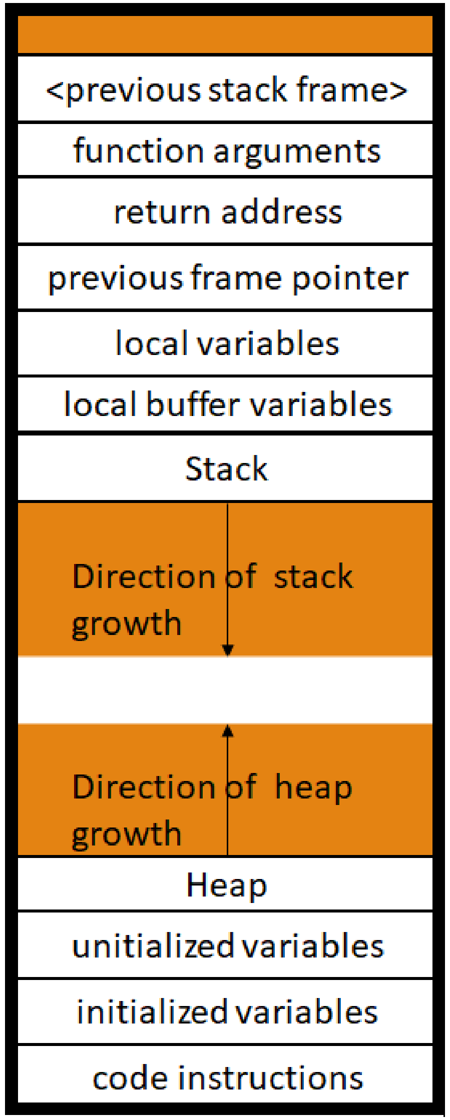
  - A buffer (or array or string) is a space in which data can be held
  - A buffer resides in memory.
    - Because memory is finite, a buffer's capacity is finite

Buffer Overflows



- ✓ Data is written beyond the space allocated for it, such as a 10th byte in a 9-byte array
- ✓ In a typical exploitable buffer overflow, an attacker’s inputs are expected to go into regions of memory allocated for data, but those inputs are instead allowed to overwrite memory holding executable code.
- ✓ The trick for an attacker is **finding buffer overflow opportunities** that lead to overwritten memory being executed, and finding the right code to input.

Memory Allocation

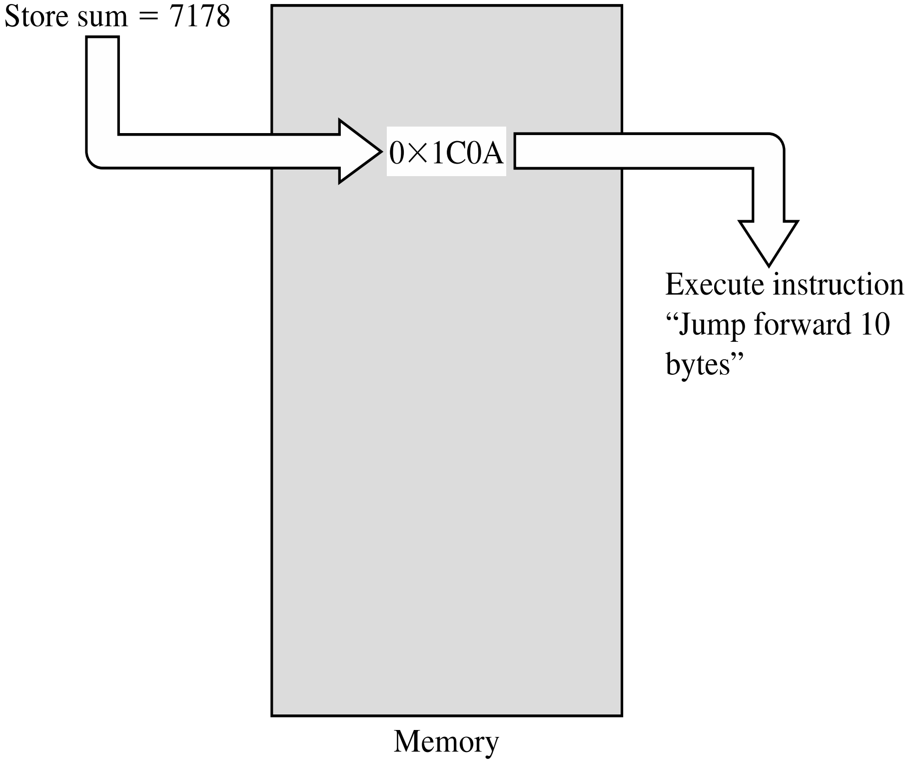

<previous stack frame>
function arguments
return address
previous frame pointer
local variables
local buffer variables
Stack
Direction of stack growth
Direction of heap growth
Heap
unitialized variables
initialized variables
code instructions

- ✓ **Memory is a limited but flexible resource**
- ✓ **Any memory location can hold any piece of code or data.**
- ✓ **Typical Memory Organization**
  - Executable cod
  - Static data: whose size is known at compile time
  - Heap (dynamic data); whose size can change during execution
  - Stack: Used to interchange of data between procedures

- ✓ Computers use a pointer or register known **as a program counter** that indicates the next instruction

Data vs. Instructions

- ✓ In memory, code is indistinguishable from data.
- ✓ You do not execute data values or perform arithmetic on instructions.
- ✓ The origin of code (respected source or attacker) is not visible.
- ✓ The attacker’s trick is to cause data to spill over into executable code
- ✓ Attacker plan:
  - ▪ Cause the overflow
  - ▪ Experiment with the ensuing action to cause a desired event



FROM *SECURITY IN COMPUTING, FIFTH EDITION*, BY CHARLES P. PFLEEGER, ET AL. (ISBN: 9780134085043). COPYRIGHT 2015 BY PEARSON EDUCATION, INC. ALL RIGHTS RESERVED.

Data vs. Instructions

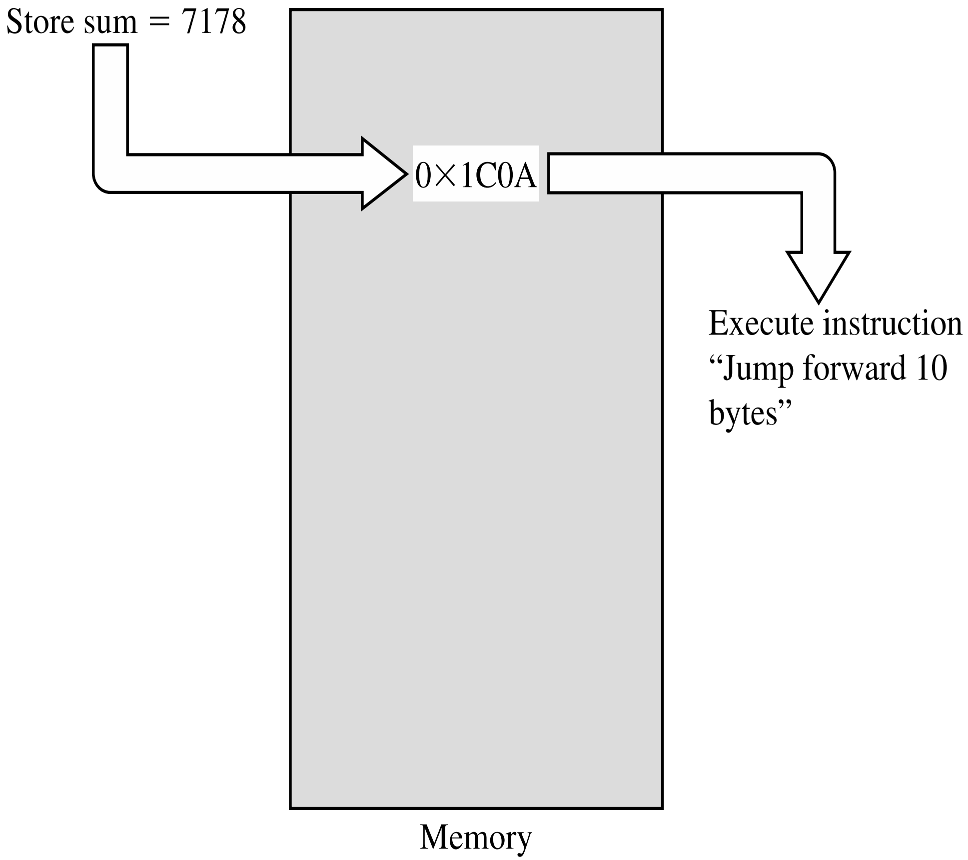
0X1C0A

Instruction

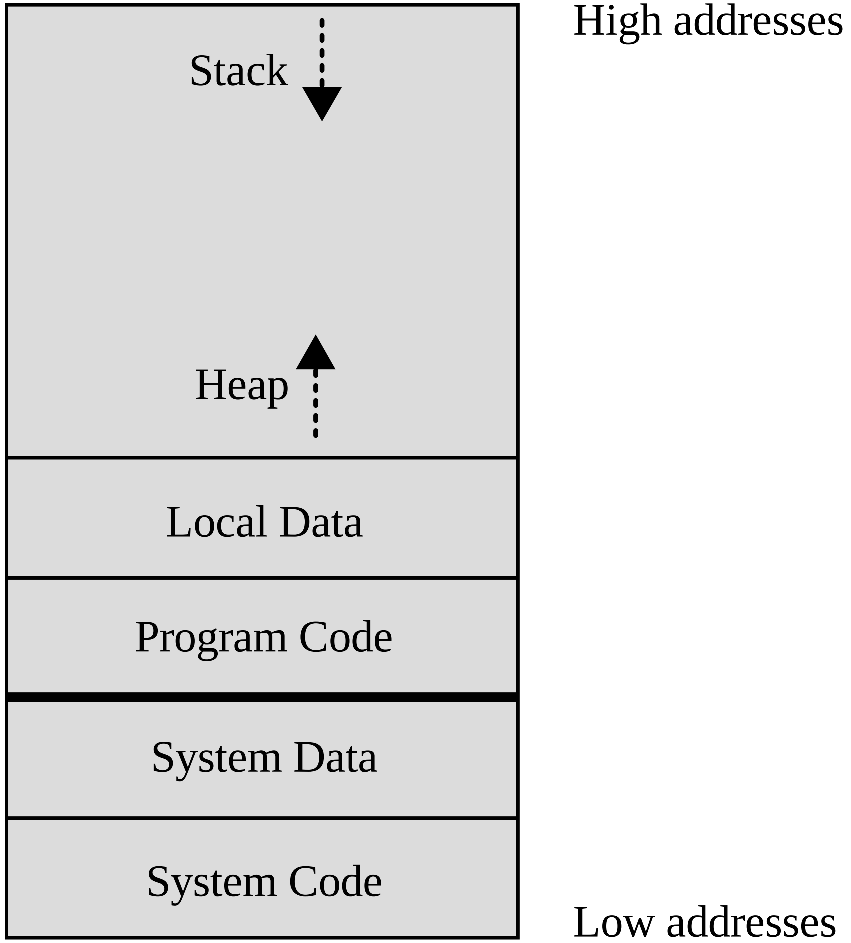
- Is the operation code for a Jump instruction.
- The string 0x1C0A is interpreted as jump forward 10 bytes

Data

Same bit pattern represent 7178



Memory Organization



- ✓ The figure shows where the system data/code reside vs. where the program code and its local data reside.
- ✓ This context is important to understand how an attack that takes place inside a given program can affect that program vs. how it can affect the rest of the system

How Buffer Overflows Happen

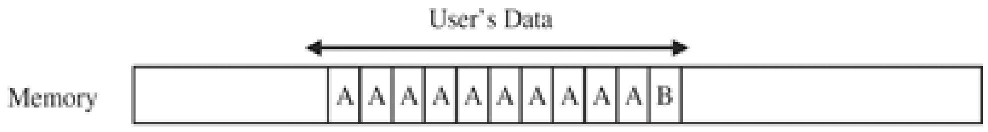
- ✓ Buffer overflows often come from **innocent programmer oversights** or failures to document and check for excessive data
- ✓ Sample code with buffer overflow vulnerability

```
char sample[10];
int i;
for (i=0; i<=9; i++)
  sample[i] = ‘A’;
sample[10] = ‘B’;
```

Where a Buffer Can Overflow

User’s data

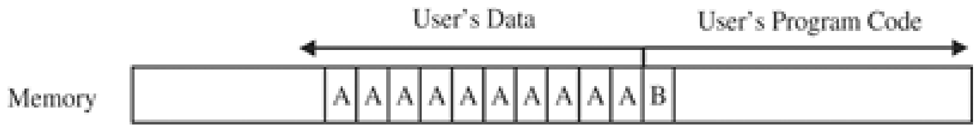
overwrites an existing variable value



(a) Affects user’s data

User’s code

overwrites an instruction that is not yet executed



(b) Affects user’s code

System’s data

computing with a faulty value

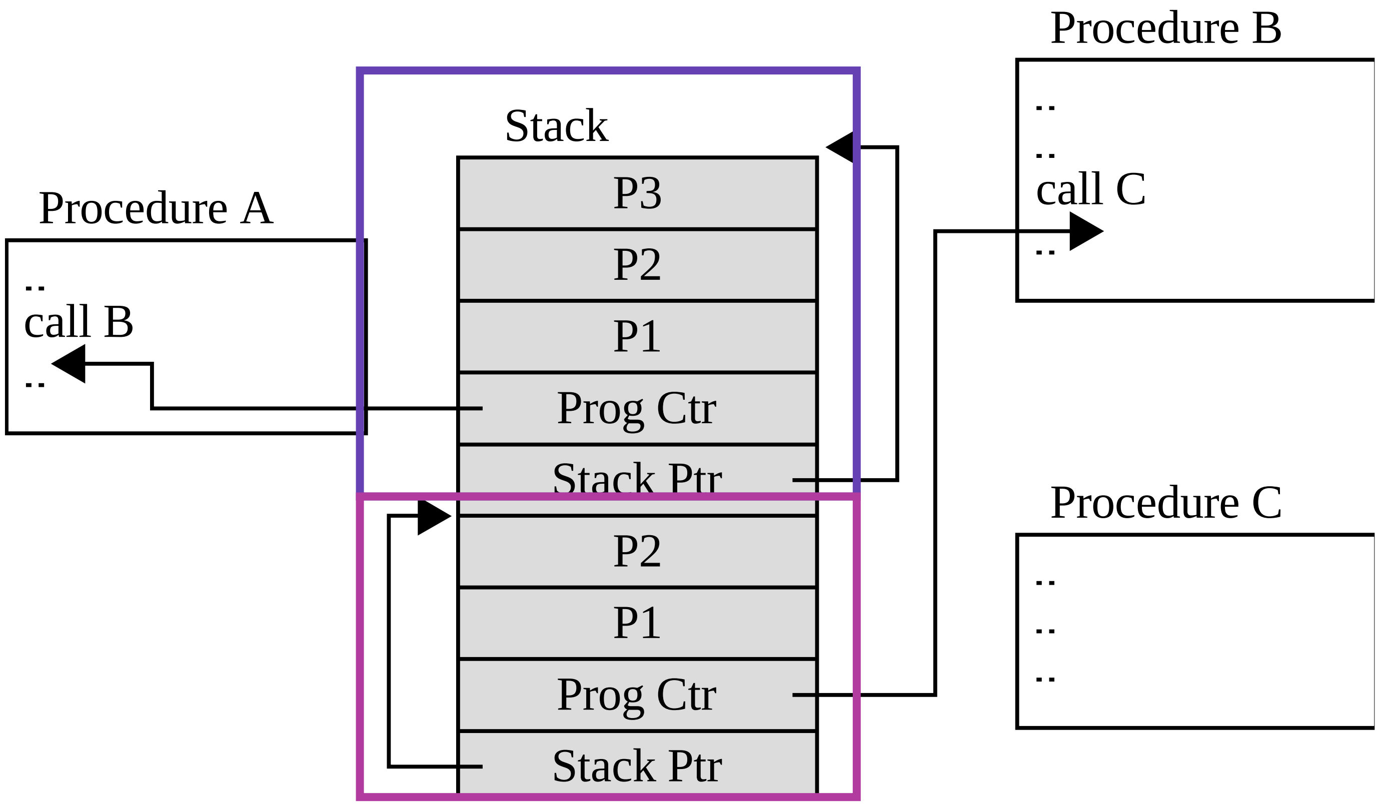
(c) Affects system data

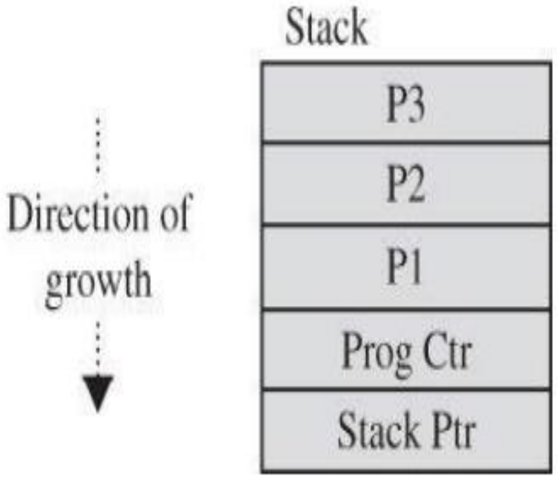
System’s code

trying to execute an improper operation.

(d) Affects system code

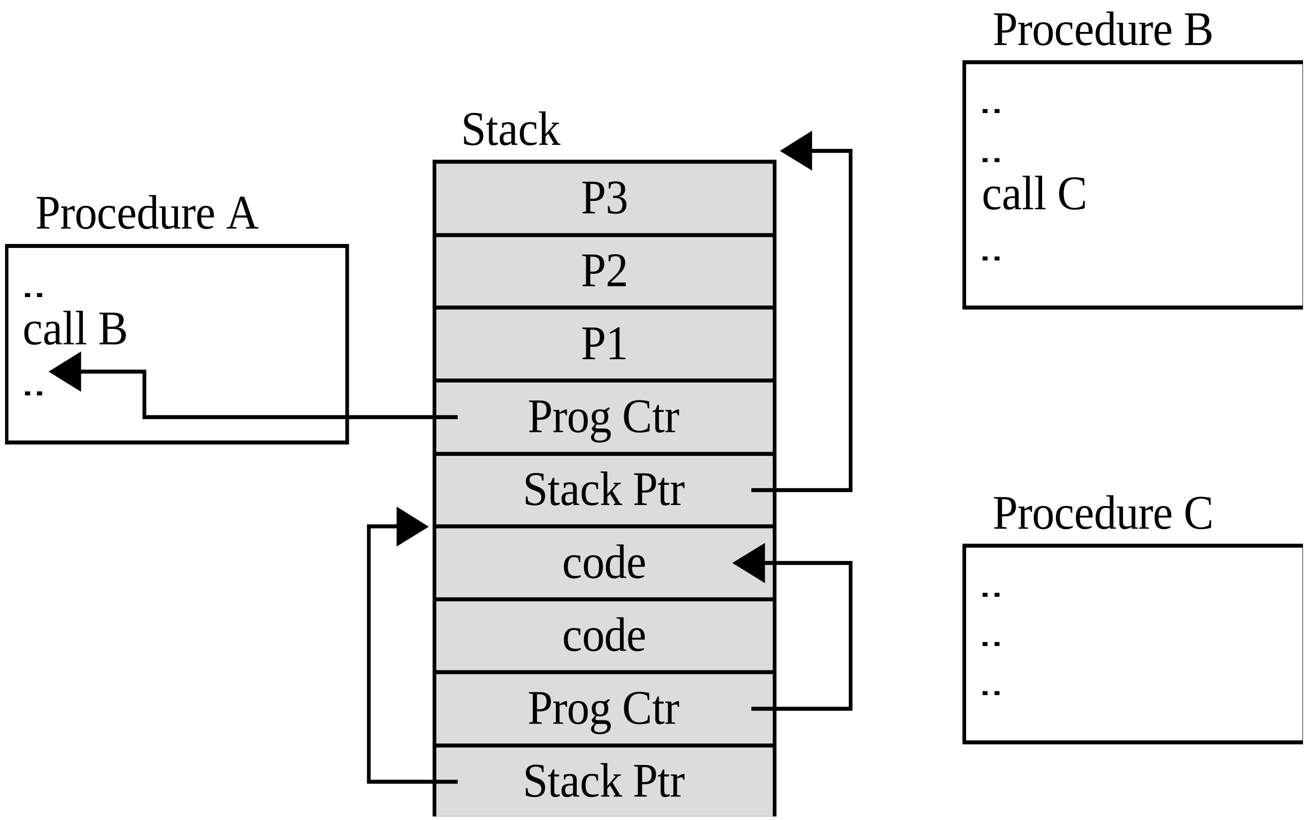
The Stack after Procedure Calls


Procedure A
..
call B
..
Stack
P3
P2
P1
Prog Ctr
Stack Ptr
P2
P1
Prog Ctr
Stack Ptr
Procedure B
..
..
call C
..
Procedure C
..
..
..



Stack Frame

Compromised Stack



A common buffer overflow stack modification
The two parameters P1 and P2 have been **overwritten** with code to which the program counter has been redirected

Stack Smashing

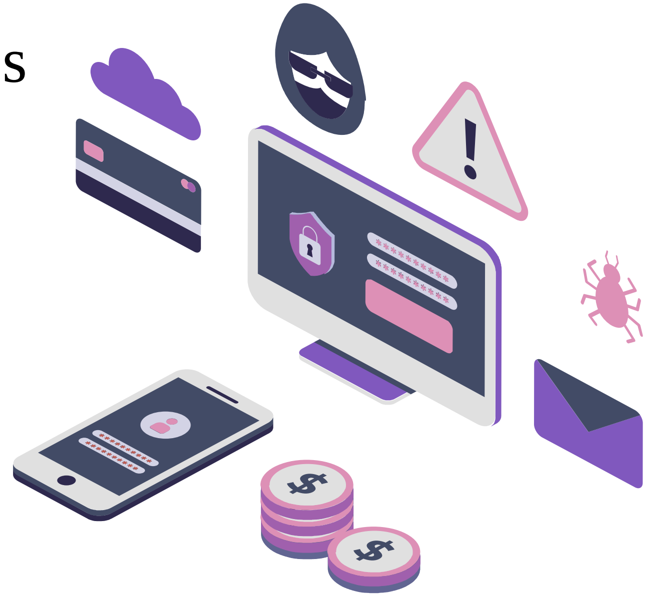
Stack Smashing: Overwriting stack memory

The Effects can be

- Overwriting the program counter stored in the stack
- Overwriting part of the code in low memory, substituting new instructions
- Overwriting the program counter and data in the stack so that the program counter points to the stack

Harm from Buffer Overflows

- ✓ **Overwrite**
  - Another piece of your program’s data
  - An instruction in your program
  - Data or code belonging to another program
  - Data or code belonging to the operating system
- ✓ Overwriting a program’s instructions gives attackers that program’s execution privileges
- ✓ Overwriting operating system instructions gives attackers the operating system’s execution privileges



Overflow Countermeasures

Staying within bounds

- Check lengths before writing
- Confirm that array subscripts are within limits
- Double-check boundary condition code for off-by-one errors
- Limit input to the number of acceptable characters
- Limit programs' privileges to reduce potential harm

Use programming languages that provide overflow protections

Overflow Countermeasures

Separation

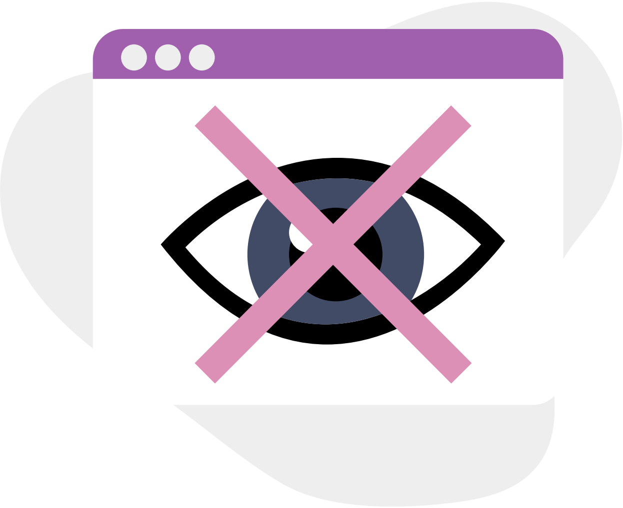
Enforce containment: Separating sensitive areas from the running code and its buffers and data space

Static Code analyzers

Analyze the code of the program and examine all possible execution paths

Stumbling blocks

- ✓ Canary values in stack to signal modification. Like the legendary canary-in-the-mine, it detects stack smash attacks.
- ✓ Inserts a “Canary value” just below the return address (Stack Guard) or just below the previous frame pointer (Stack Smashing Protector). This value gets checked right before a function returns.



Incomplete Mediation

Incomplete Mediation

Incomplete mediation is a security problem

- ✓ Attackers exploit incomplete mediation to cause security problems
- ✓ Users make errors from ignorance, misunderstanding, distraction
  - *User errors should not cause program failures*

Mediation: checking

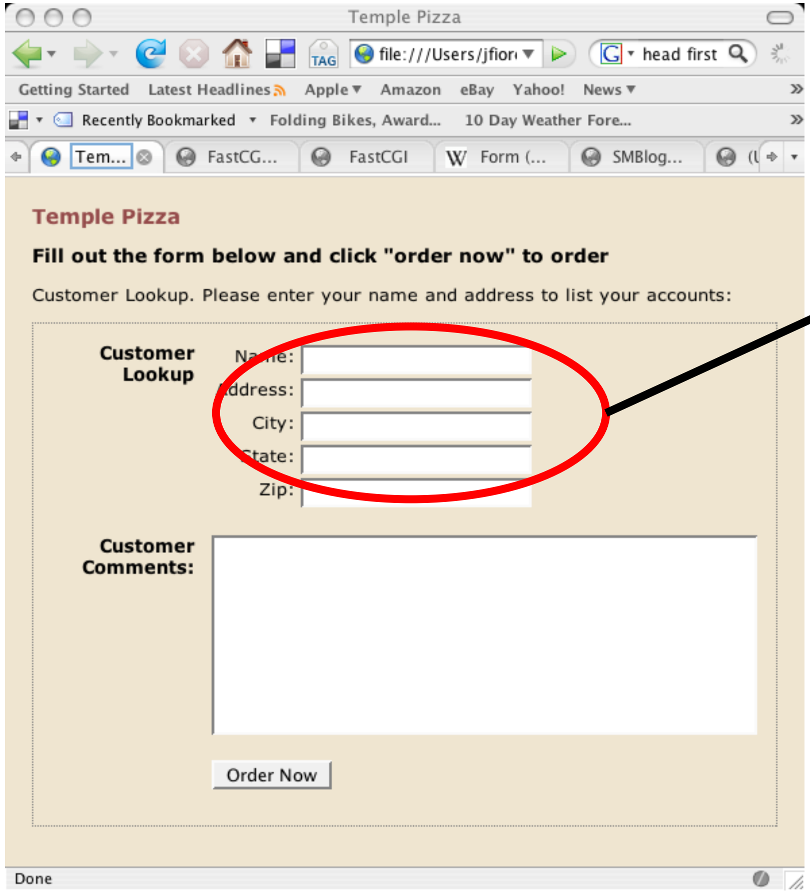
- ✓ Verifying that the subject is authorized to perform the operation on an object

Preventing incomplete mediation:

- ✓ Validate all input
- ✓ Limit users' access to sensitive data and functions
- ✓ If data can be changed, assume they have been

Complete mediation in OS is achieved using a reference monitor

Incomplete Mediation (ex. SQL Injection)



Temple Pizza

Fill out the form below and click "order now" to order

Customer Lookup. Please enter your name and address to list your accounts:

Customer Lookup

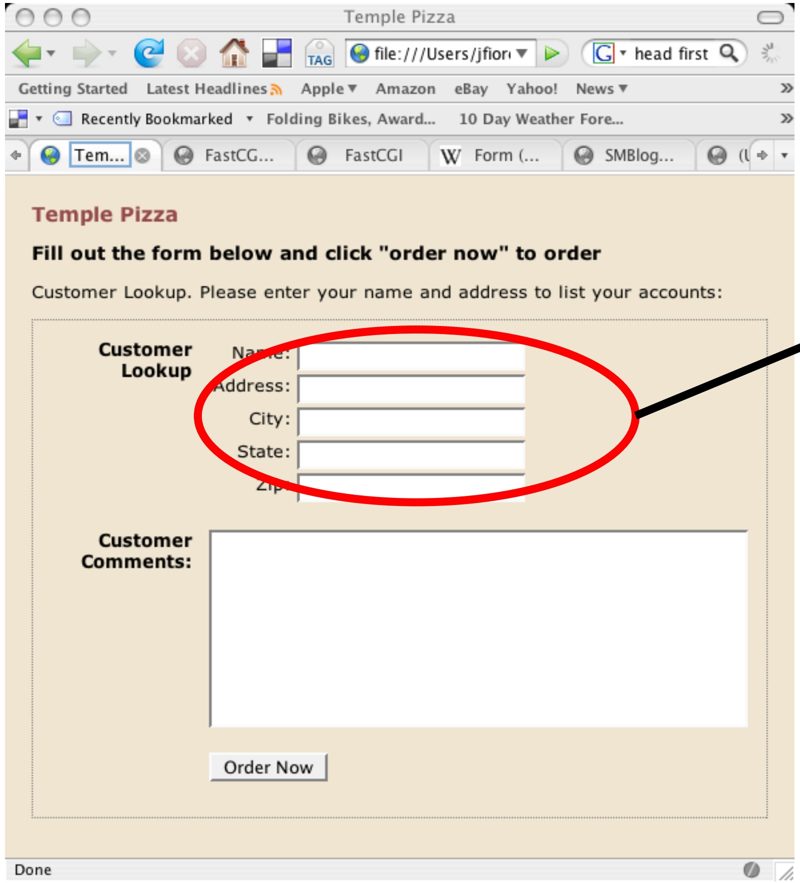
Name:
Address:
City:
State:
Zip:

Customer Comments:

John Fiore

SELECT * from CUSTOMERS
WHERE name = 'John Fiore'

Incomplete Mediation (ex. SQL Injection)



Temple Pizza

Fill out the form below and click "order now" to order

Customer Lookup. Please enter your name and address to list your accounts:

Customer Lookup
Name:
Address:
City:
State:
Zip

Customer Comments:

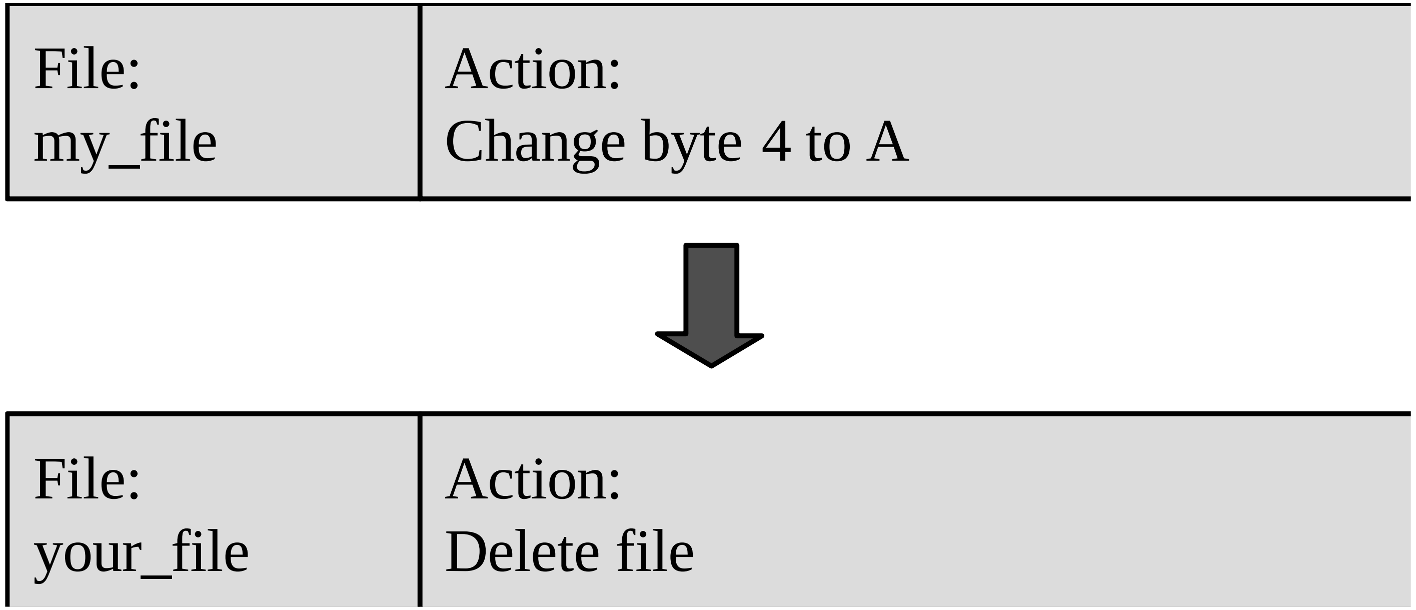
John Fiore' or '1'='1

SELECT * from CUSTOMERS
WHERE name = 'John Fiore'
OR '1'='1'

Time-of-Check to Time-of-Use

Time-of-Check to Time-of-Use

Mediation performed with a “bait and switch” in the middle

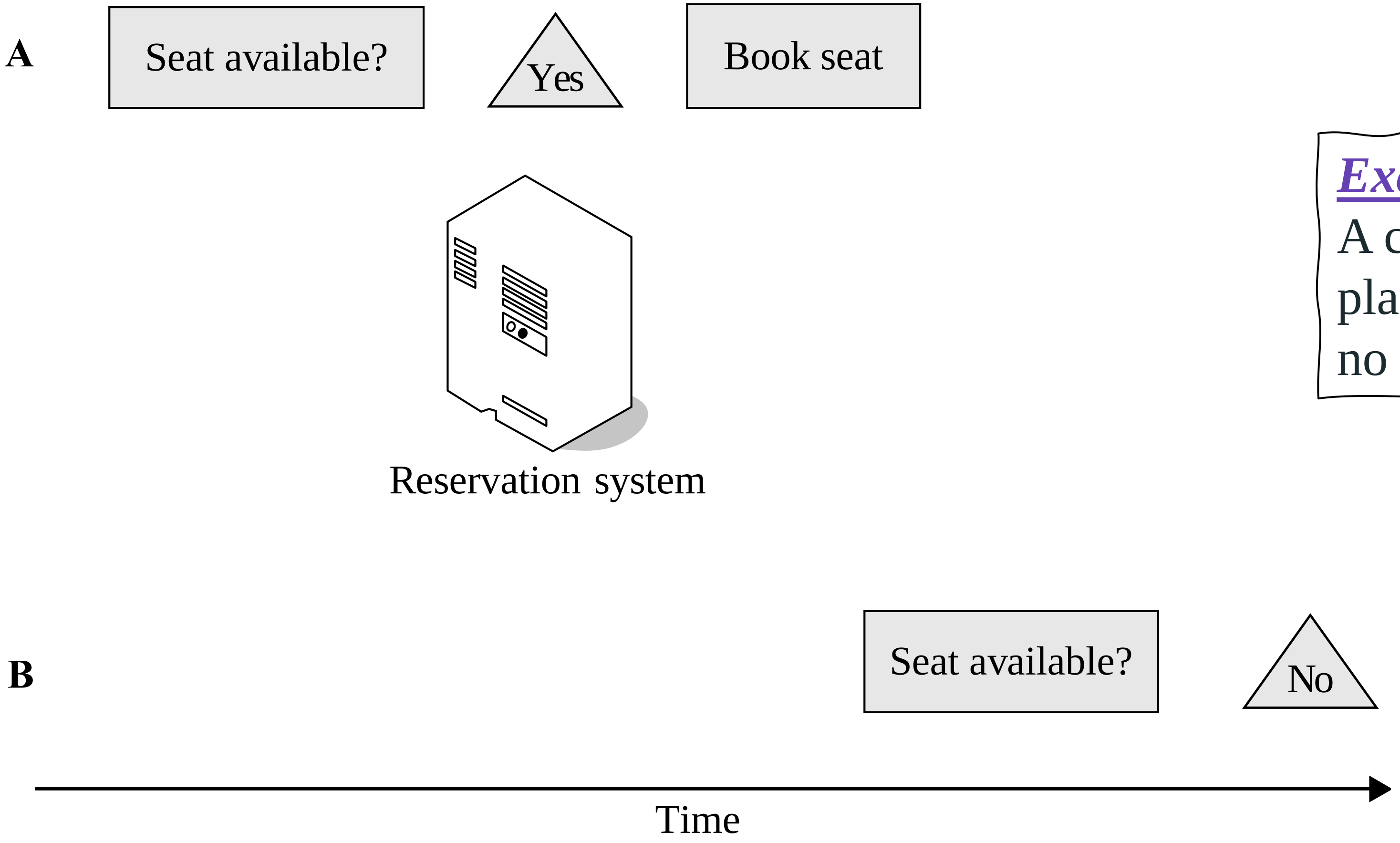


File:	Action:
my_file	Change byte 4 to A

File:	Action:
your_file	Delete file

Race Conditions

Race Conditions



A
Seat available?
Yes
Book seat

Reservation system

B
Seat available?
No

Time

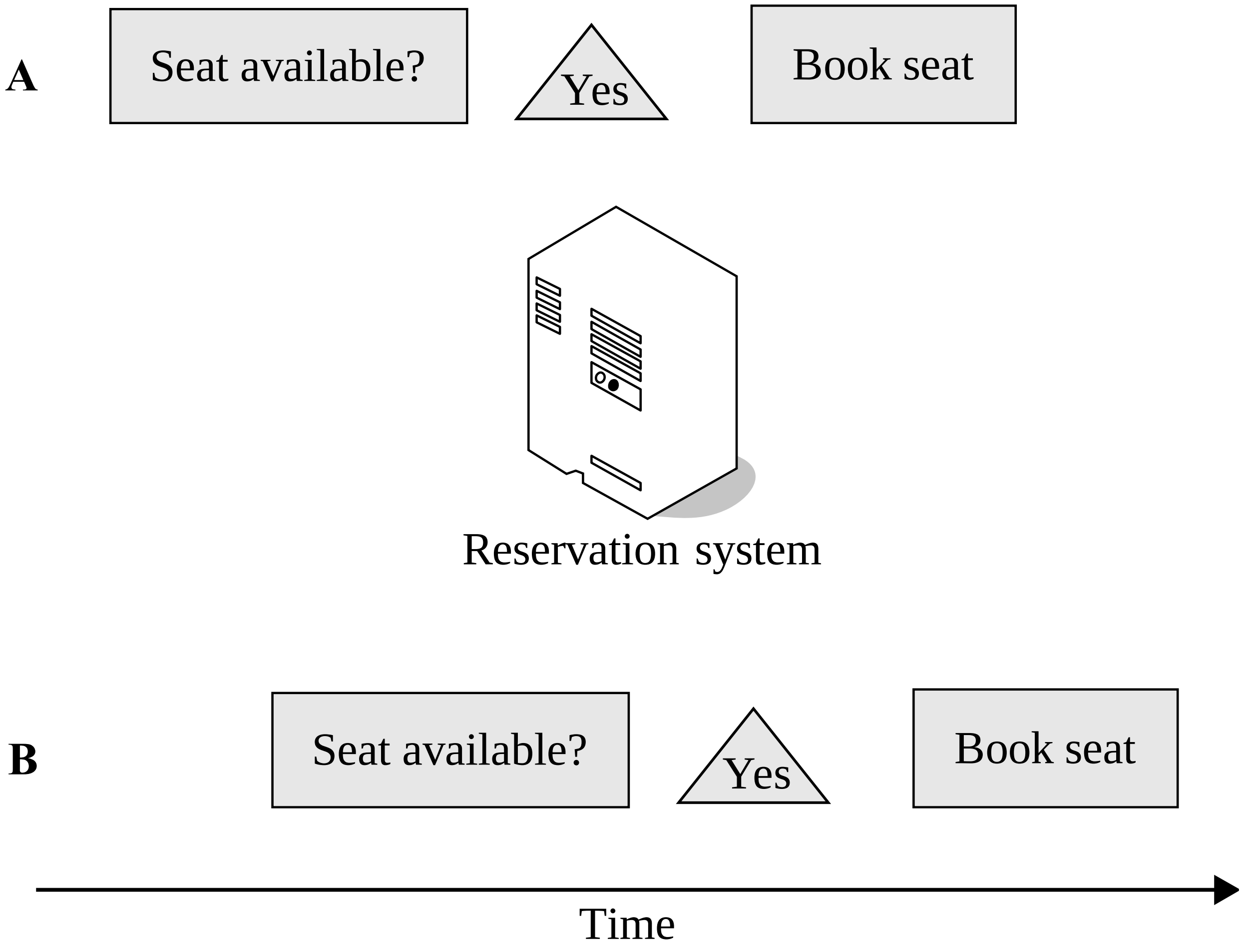
Example 1 (no race condition):
A customer books the last seat on the plane, and thereafter the system shows no seat available.

Race Conditions

Example 2 (race condition):

Before customer A can complete the booking for the last available seat, customer B looks for available seats.

This system has a *race condition*, where the overlap in timing of the requests causes errant behavior.



A

Seat available?

Yes

Book seat

Reservation system

B

Seat available?

Yes

Book seat

Time

Intentional Causes

Intentional Causes can be

Non-malicious

Causing failure to test the system

Malicious:

Causing failure to harm

Malware

Programs planted by an agent with malicious intent to cause unanticipated or undesired effects

Intentional Causes

Types of malware include:

Virus

A program that can replicate itself and pass on malicious code to other non-malicious programs by modifying them.

Two categories:

- transient virus, has a life span that depends on the life of its host, and
- resident virus, locates itself in memory

Worm

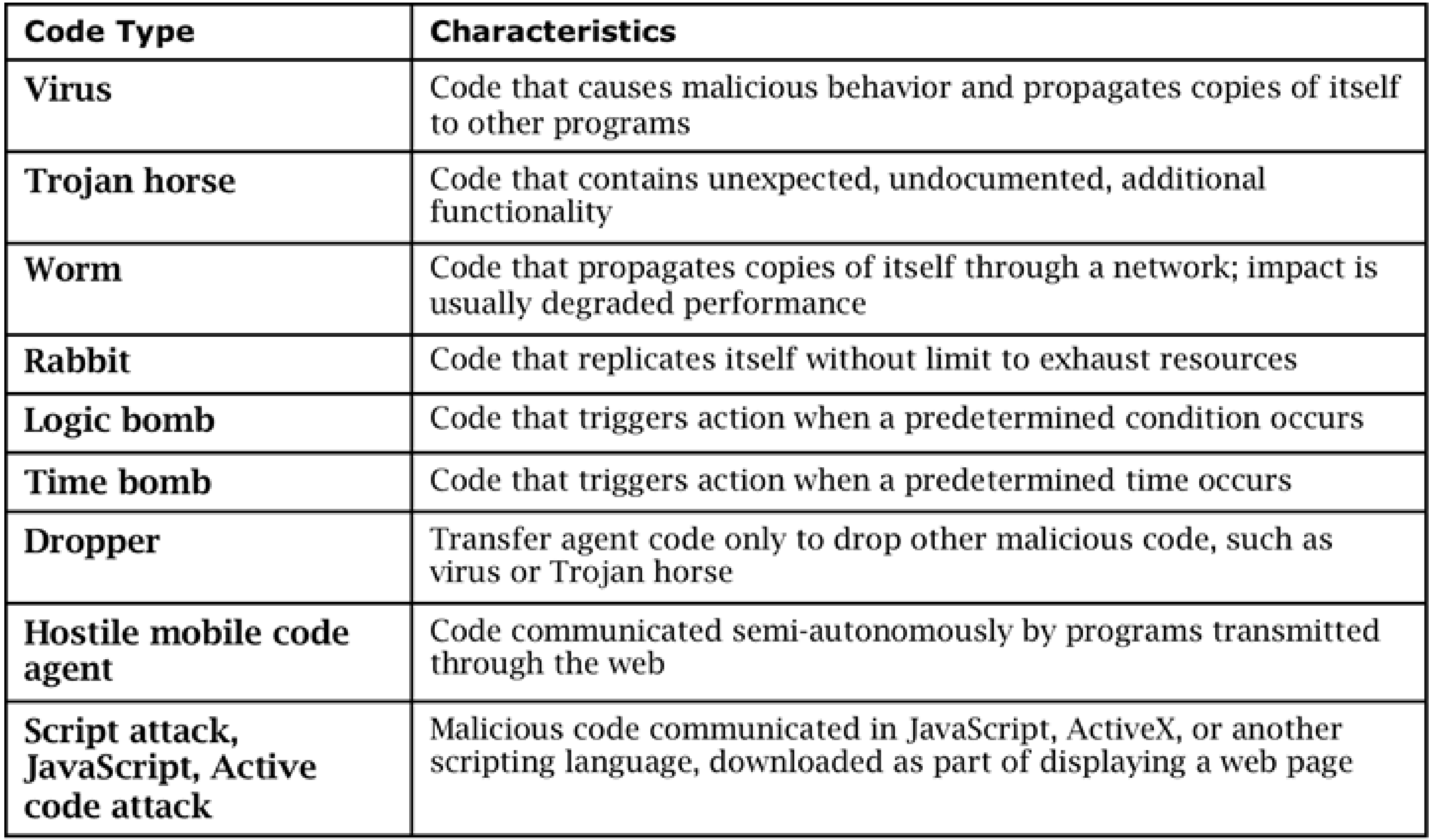
A program that spreads copies of itself through a network

Trojan horse

Code that, in addition to its stated effect, has a second, nonobvious, malicious effect

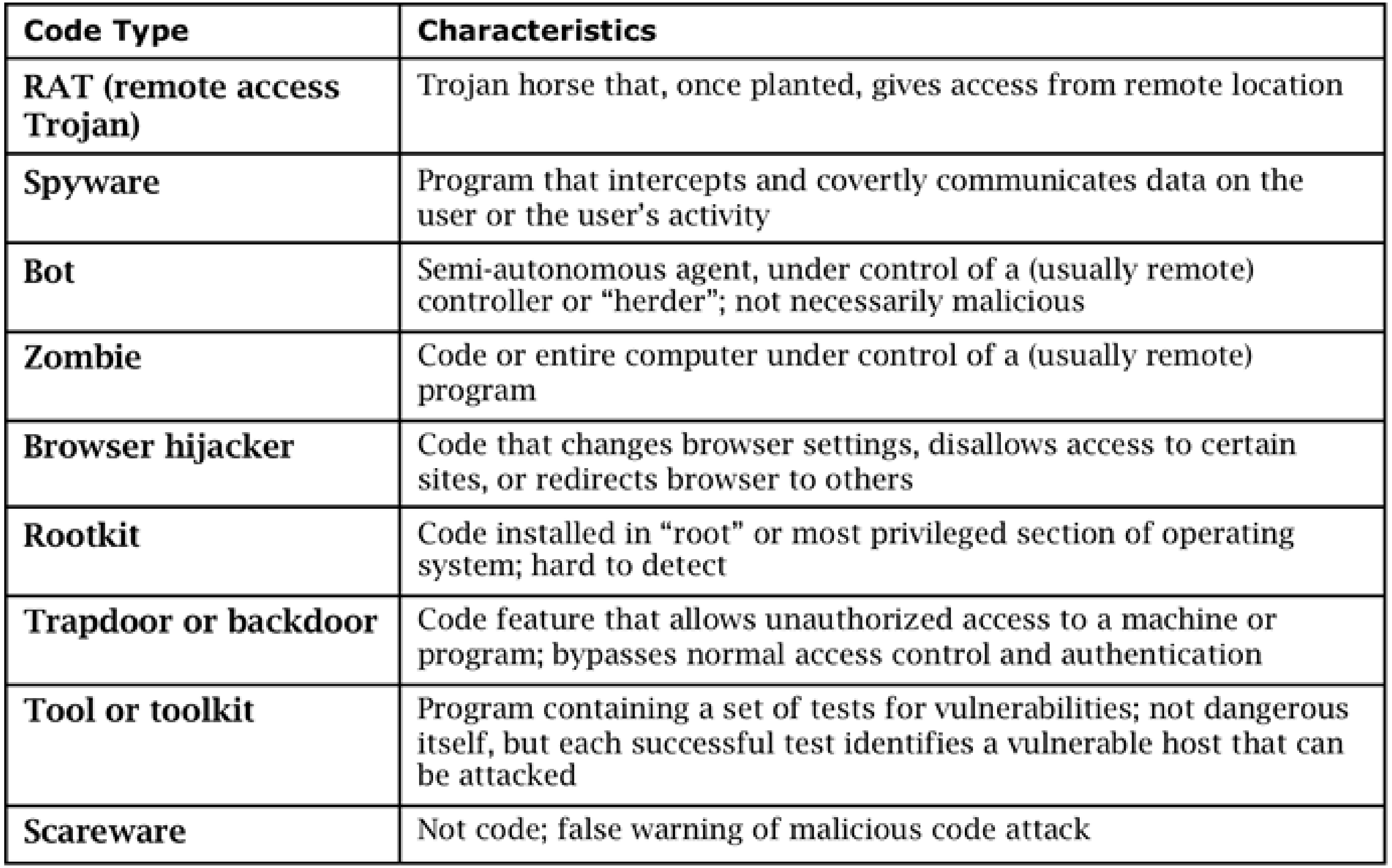
Others, such as: Bombs; Bots; RAT; and more

Types of Malware



Code Type	Characteristics
Virus	Code that causes malicious behavior and propagates copies of itself to other programs
Trojan horse	Code that contains unexpected, undocumented, additional functionality
Worm	Code that propagates copies of itself through a network; impact is usually degraded performance
Rabbit	Code that replicates itself without limit to exhaust resources
Logic bomb	Code that triggers action when a predetermined condition occurs
Time bomb	Code that triggers action when a predetermined time occurs
Dropper	Transfer agent code only to drop other malicious code, such as virus or Trojan horse
Hostile mobile code agent	Code communicated semi-autonomously by programs transmitted through the web
Script attack, JavaScript, Active code attack	Malicious code communicated in JavaScript, ActiveX, or another scripting language, downloaded as part of displaying a web page

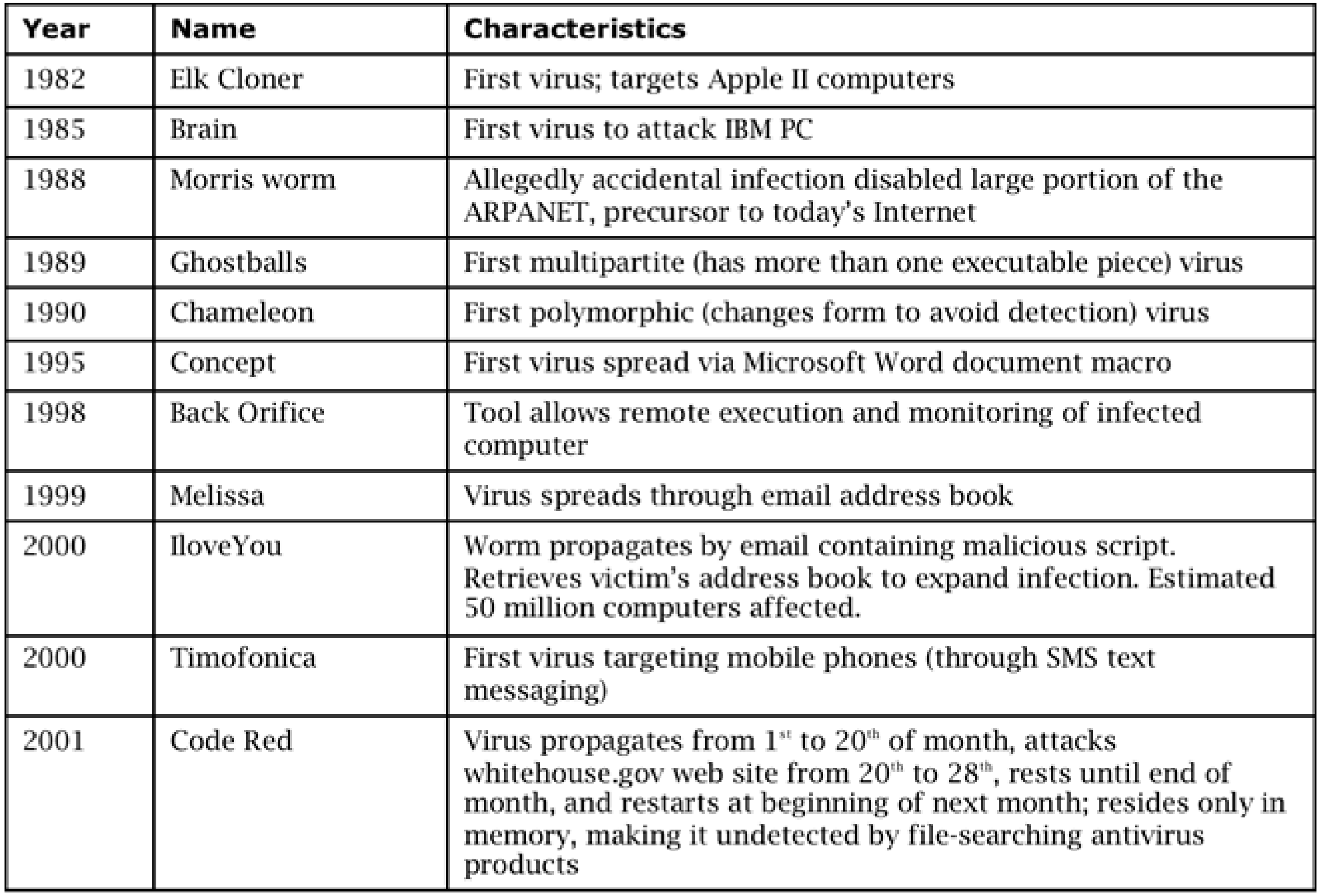
Types of Malware



Code Type	Characteristics
RAT (remote access Trojan)	Trojan horse that, once planted, gives access from remote location
Spyware	Program that intercepts and covertly communicates data on the user or the user's activity
Bot	Semi-autonomous agent, under control of a (usually remote) controller or "herder"; not necessarily malicious
Zombie	Code or entire computer under control of a (usually remote) program
Browser hijacker	Code that changes browser settings, disallows access to certain sites, or redirects browser to others
Rootkit	Code installed in "root" or most privileged section of operating system; hard to detect
Trapdoor or backdoor	Code feature that allows unauthorized access to a machine or program; bypasses normal access control and authentication
Tool or toolkit	Program containing a set of tests for vulnerabilities; not dangerous itself, but each successful test identifies a vulnerable host that can be attacked
Scareware	Not code; false warning of malicious code attack

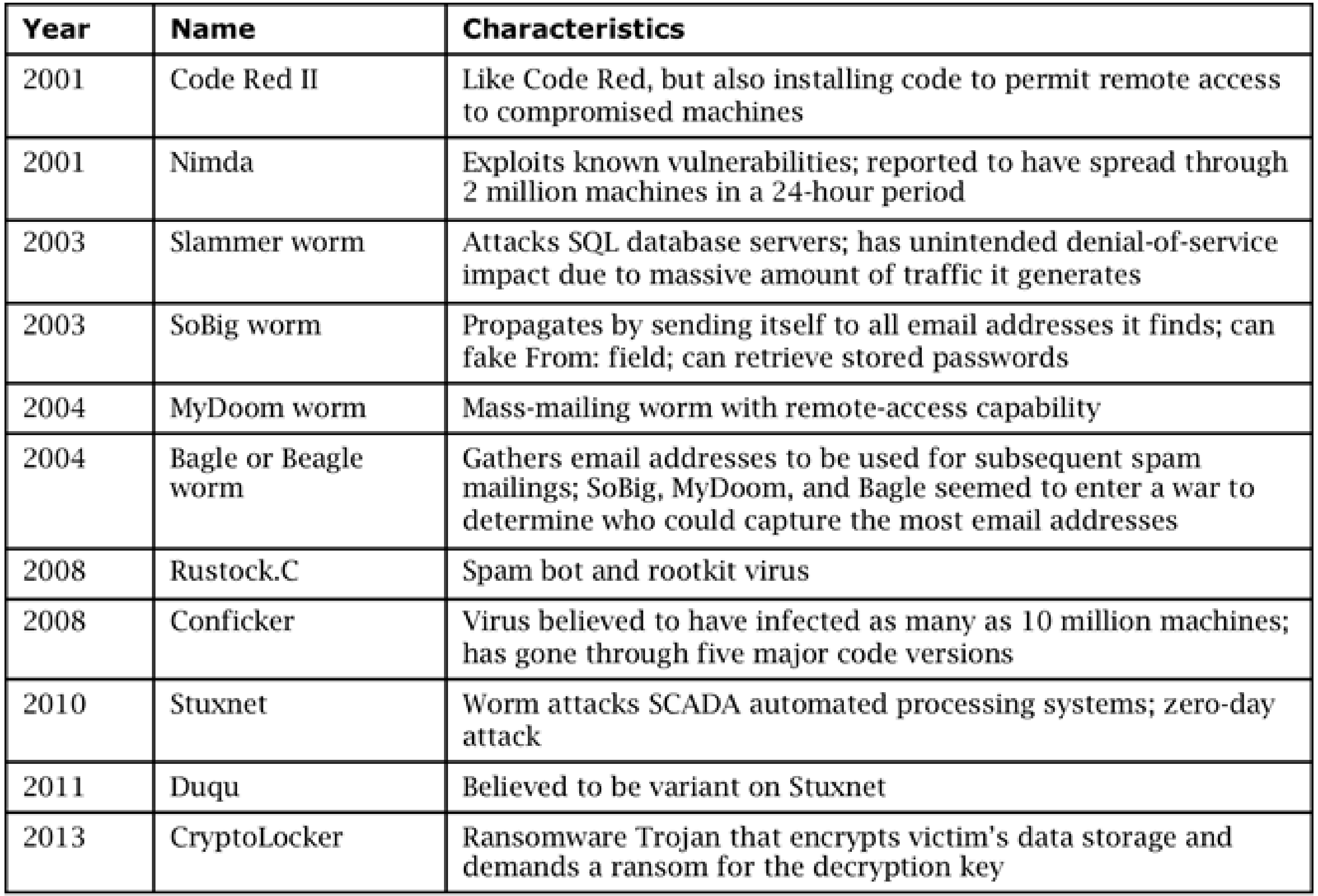
FROM *SECURITY IN COMPUTING, FIFTH EDITION*, BY CHARLES P. PFLEEGER, ET AL. (ISBN: 9780134085043). COPYRIGHT 2015 BY PEARSON EDUCATION, INC. ALL RIGHTS RESERVED.

History of Malware



Year	Name	Characteristics
1982	Elk Cloner	First virus; targets Apple II computers
1985	Brain	First virus to attack IBM PC
1988	Morris worm	Allegedly accidental infection disabled large portion of the ARPANET, precursor to today's Internet
1989	Ghostballs	First multipartite (has more than one executable piece) virus
1990	Chameleon	First polymorphic (changes form to avoid detection) virus
1995	Concept	First virus spread via Microsoft Word document macro
1998	Back Orifice	Tool allows remote execution and monitoring of infected computer
1999	Melissa	Virus spreads through email address book
2000	IloveYou	Worm propagates by email containing malicious script. Retrieves victim's address book to expand infection. Estimated 50 million computers affected.
2000	Timofonica	First virus targeting mobile phones (through SMS text messaging)
2001	Code Red	Virus propagates from 1st to 20th of month, attacks whitehouse.gov web site from 20th to 28th, rests until end of month, and restarts at beginning of next month; resides only in memory, making it undetected by file-searching antivirus products

History of Malware

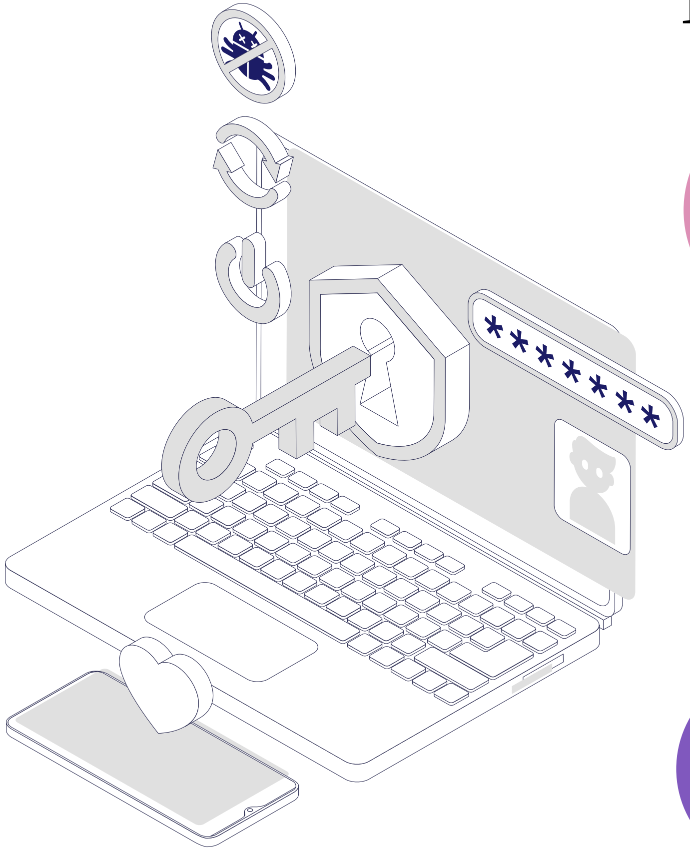


Year	Name	Characteristics
2001	Code Red II	Like Code Red, but also installing code to permit remote access to compromised machines
2001	Nimda	Exploits known vulnerabilities; reported to have spread through 2 million machines in a 24-hour period
2003	Slammer worm	Attacks SQL database servers; has unintended denial-of-service impact due to massive amount of traffic it generates
2003	SoBig worm	Propagates by sending itself to all email addresses it finds; can fake From: field; can retrieve stored passwords
2004	MyDoom worm	Mass-mailing worm with remote-access capability
2004	Bagle or Beagle worm	Gathers email addresses to be used for subsequent spam mailings; SoBig, MyDoom, and Bagle seemed to enter a war to determine who could capture the most email addresses
2008	Rustock.C	Spam bot and rootkit virus
2008	Conficker	Virus believed to have infected as many as 10 million machines; has gone through five major code versions
2010	Stuxnet	Worm attacks SCADA automated processing systems; zero-day attack
2011	Duqu	Believed to be variant on Stuxnet
2013	CryptoLocker	Ransomware Trojan that encrypts victim's data storage and demands a ransom for the decryption key

FROM *SECURITY IN COMPUTING, FIFTH EDITION*, BY CHARLES P. PFLEEGER, ET AL. (ISBN: 9780134085043). COPYRIGHT 2015 BY PEARSON EDUCATION, INC. ALL RIGHTS RESERVED.

Harm from Malicious Code

Malicious code can be directed at a specific user or class of users, or it can be for anyone



Three categories of harm:

1 ***Nondestructive***
aka virus hoax. E.g., sending a funny message or flashing an image on the screen to show the author’s capability.

2 ***Destructive***:
corrupts/deletes files, damages sw, or executes commands to cause hw stress.

3 ***Commercial/criminal intent:***
tries to take over the recipient’s computer. E.g., collecting personal data, such as login credentials to a banking system

Harm from Malicious Code

Targets:
- Users
- Systems
- Groups/ Countries/ Nations

Types of Harm to:

Users and systems:
- **Sending email** to user contacts
- **Deleting or encrypting files**
- **Stealing sensitive information**, such as passwords
- **Modifying system information**, such as the Windows registry
- **Attaching to critical system files**
- **Hide copies of malware** in multiple complementary locations

The world
- Some malware has been known to **infect millions of systems**, growing at a geometric rate
- **Infected systems** often become staging areas for **new infections**

FROM *SECURITY IN COMPUTING, FIFTH EDITION*, BY CHARLES P. PFLEEGER, ET AL. (ISBN: 9780134085043). COPYRIGHT 2015 BY PEARSON EDUCATION, INC. ALL RIGHTS RESERVED.

Transmission and Propagation

How malware are transmitted?

A virus is activated by being executed

Host is called

Many ways to ensure that programs will be executed

Setup and installer programs

The SETUP program call dozens or hundreds of other programs. If any one of these programs contains a virus, the virus code could be activated

Attached files.

e-mail attachment is a common way for viruses to get activated

Transmission and Propagation

Many ways to ensure that programs will be executed

Autorun.

Running an infected program obtained from distribution medium, such as a CD or a USB stick

Document viruses

Viruses implemented within a formatted document.

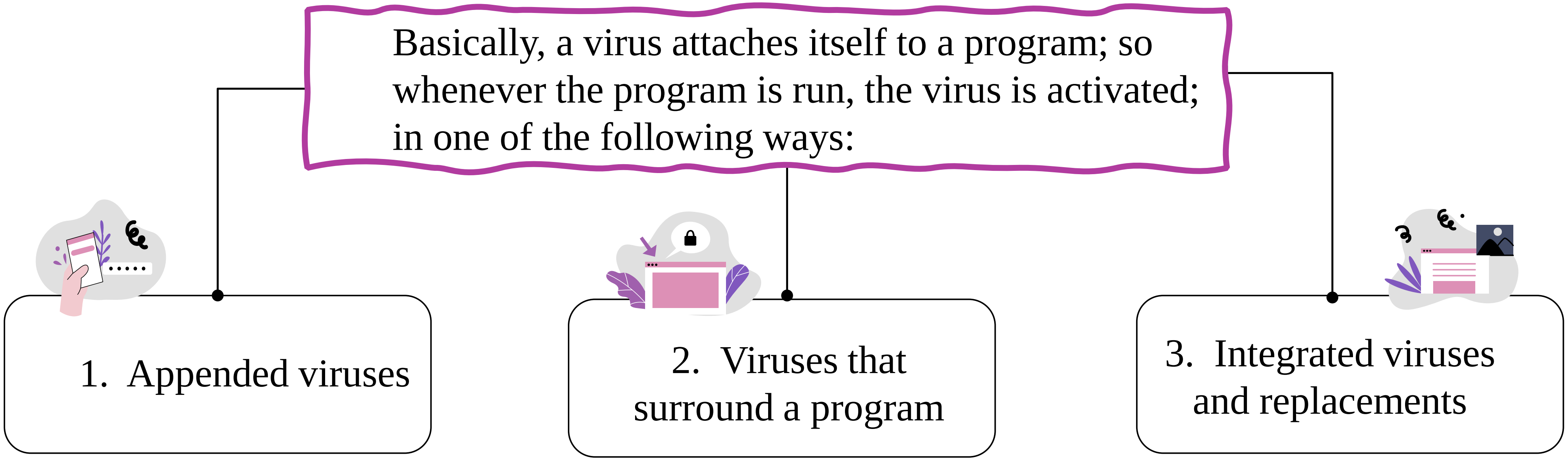
- Highly structured and contain data and commands.
- Objects such as graphics or photo images can contain code to be executed by an editor/viewer

Transmission and Propagation

How malware replicate?

Using non-malicious programs

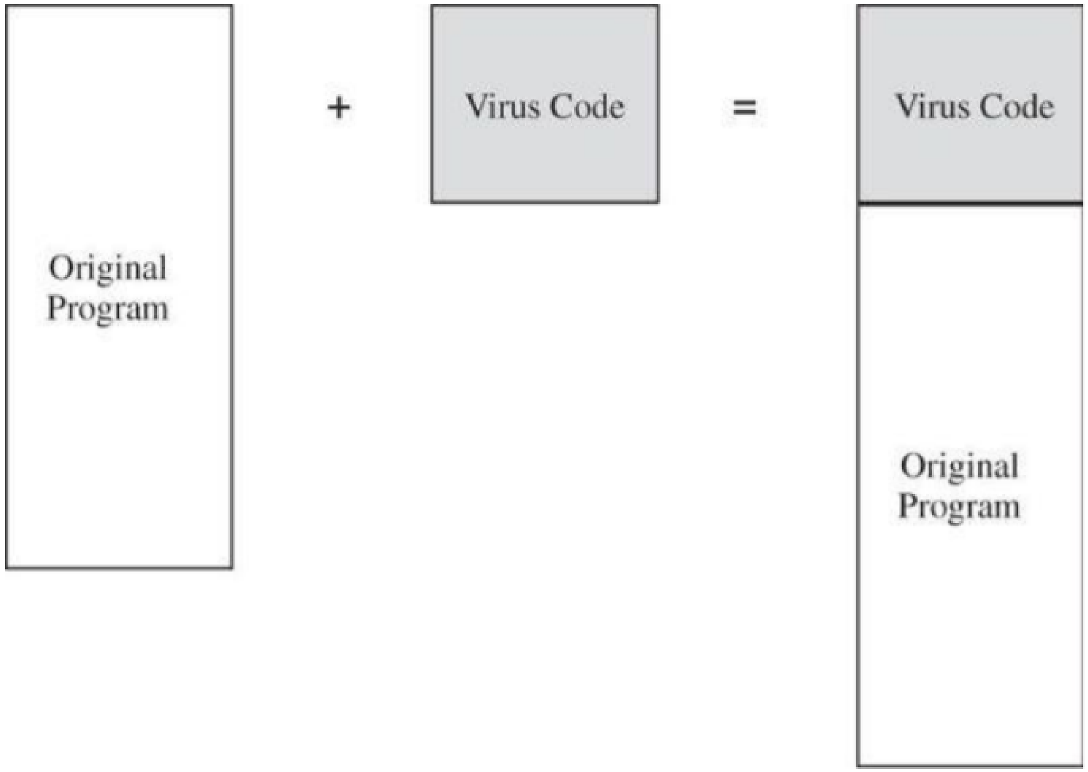
A virus is small, its code can be “hidden” inside other larger and more complicated programs



Basically, a virus attaches itself to a program; so whenever the program is run, the virus is activated; in one of the following ways:

1. Appended viruses
2. Viruses that surround a program
3. Integrated viruses and replacements

Transmission and Propagation – cont.

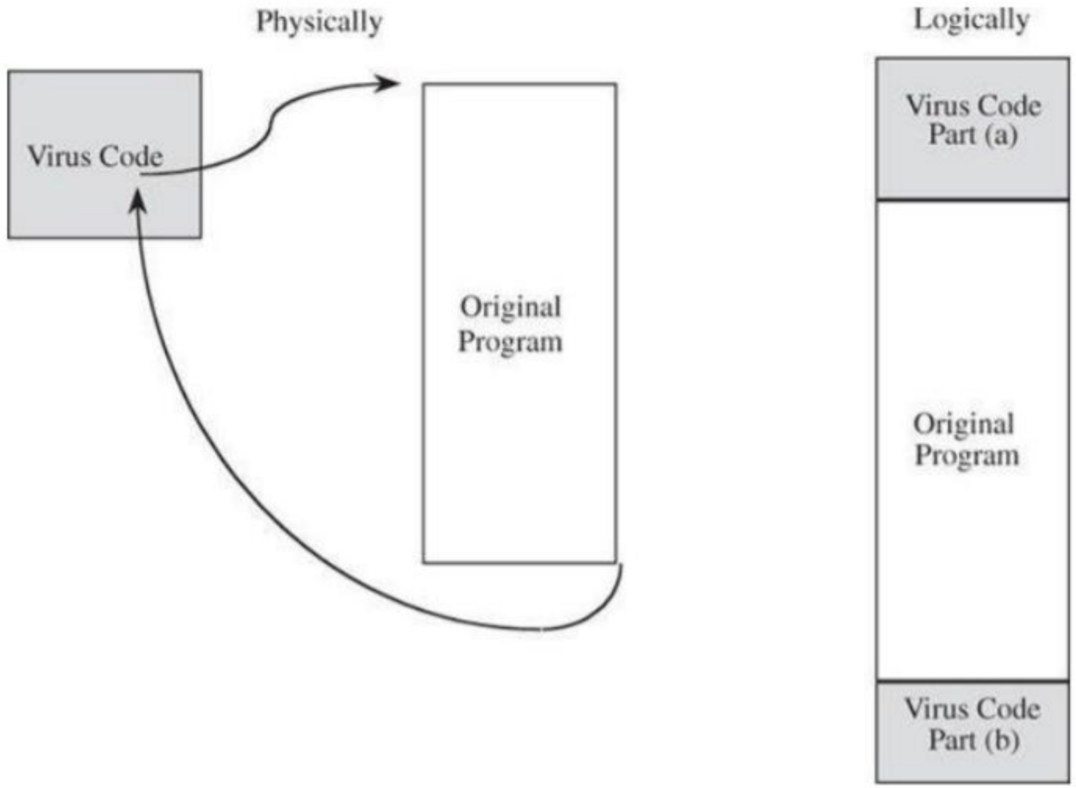


1

Appended viruses

The virus inserts a copy of itself into the executable program file before the first executable instruction

Transmission and Propagation – cont.

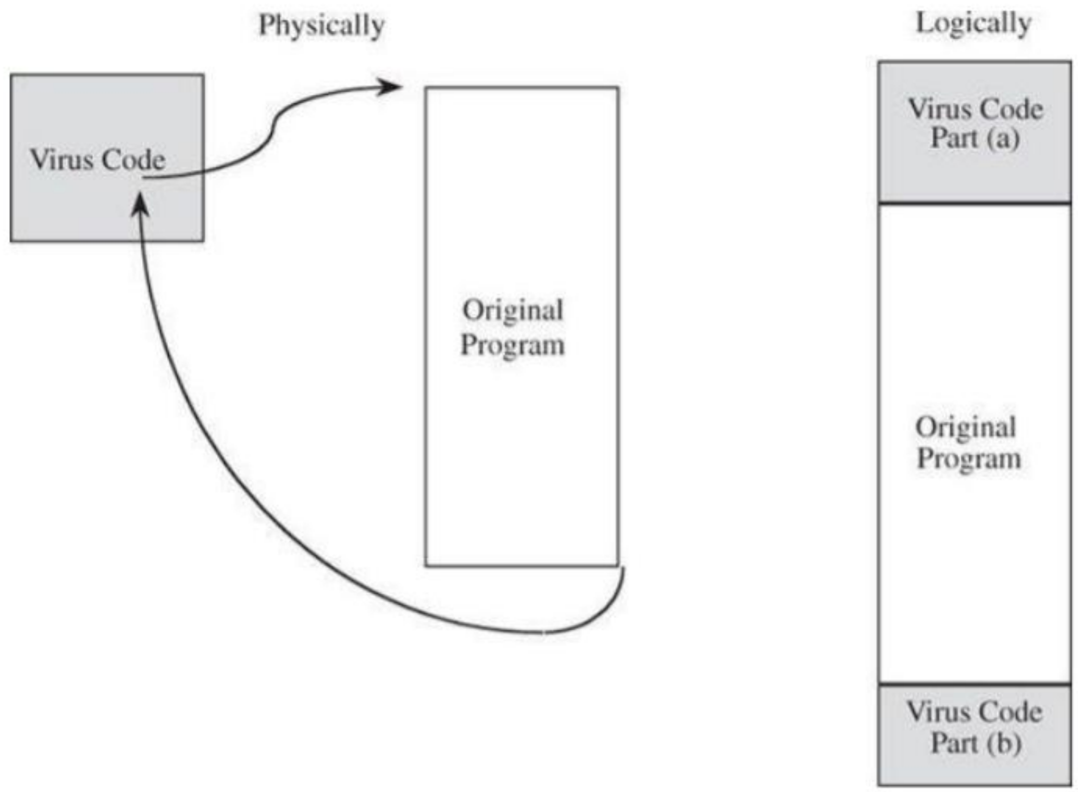


2

Viruses that surround a program

The virus runs the original program but has control before and after its execution.

Transmission and Propagation – cont.



3

Integrated viruses and replacements

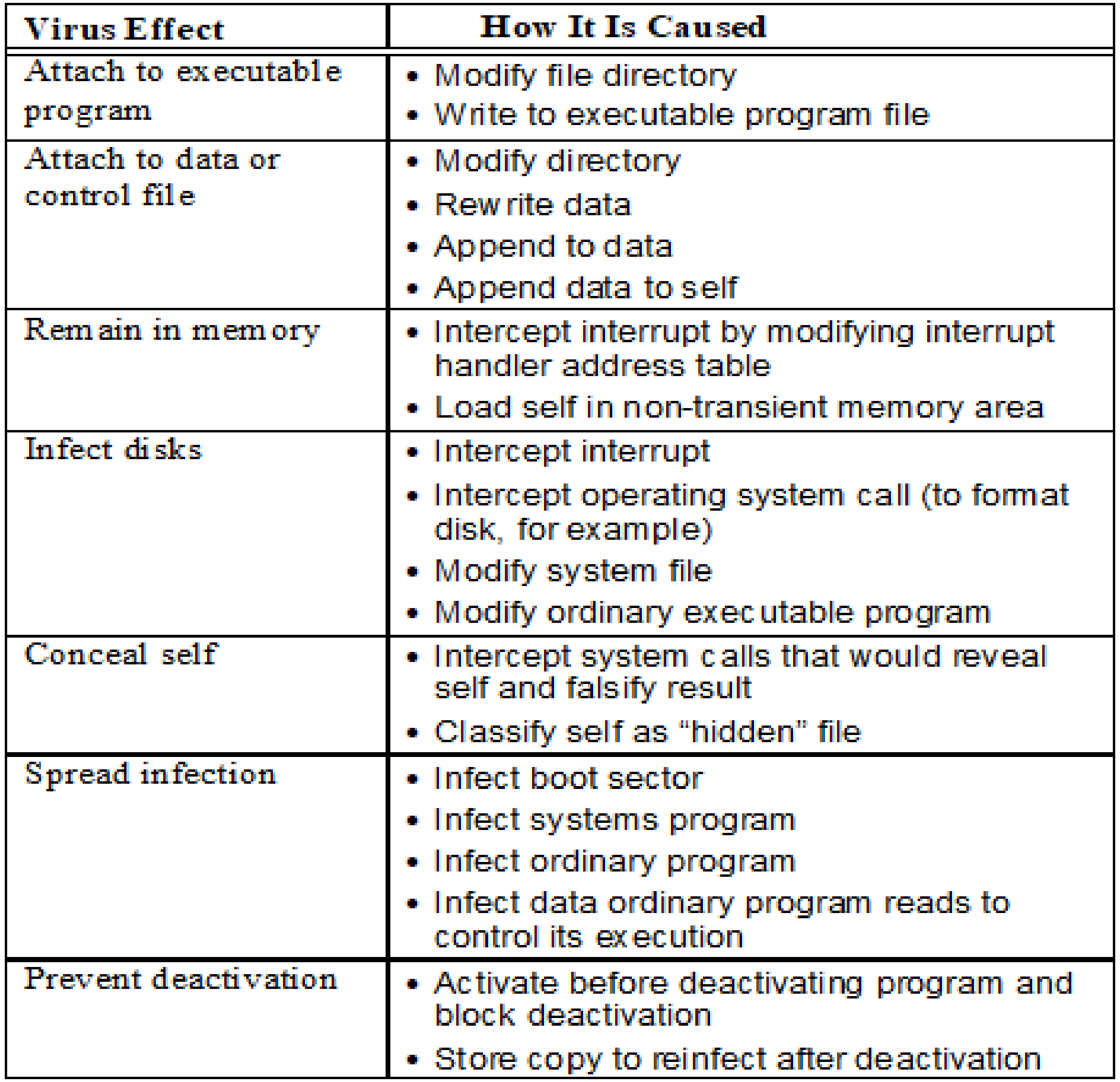
The virus replaces some of its target, integrating itself into the original code of the target

Malware Activation

How Malicious Code Gains Control?

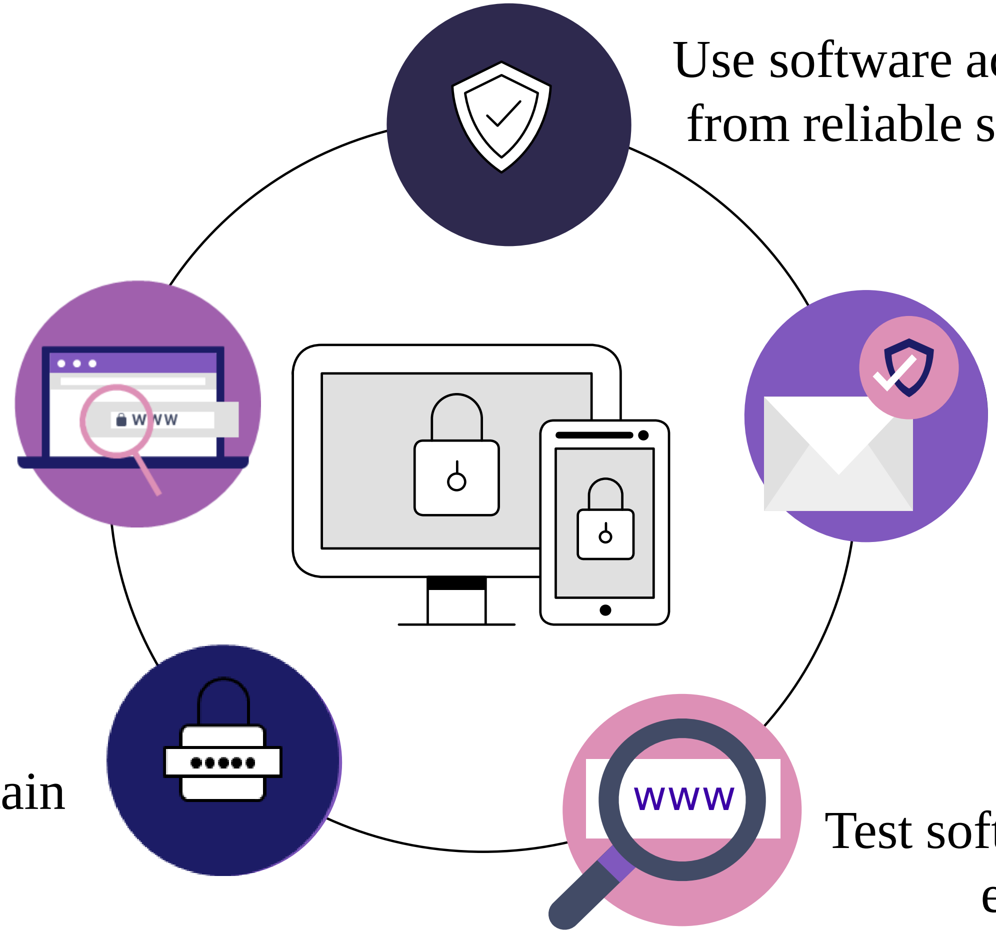
One-time execution (implanting)	Virus executes a one-time process to transmit or receive and install the infection
Boot sector viruses	Causes continuing/repeated harm, instead of just a one-time assault Malicious code can intrude in bootstrap sequence
Memory-resident viruses	Resident code is activated many times while the machine is running, therefore very appealing to virus writers
Application files	Many applications, have a “macro” feature; virus writer adds malware to a trusted, commonly used application
Code libraries	Libraries are used by many programs

Virus Effects



Virus Effect	How It Is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory	• Intercept interrupt by modifying interrupt handler address table • Load self in non-transient memory area
Infect disks	• Intercept interrupt • Intercept operating system call (to format disk, for example) • Modify system file • Modify ordinary executable program
Conceal self	• Intercept system calls that would reveal self and falsify result • Classify self as "hidden" file
Spread infection	• Infect boot sector • Infect systems program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation	• Activate before deactivating program and block deactivation • Store copy to reinfect after deactivation

Countermeasures for Users



Use software acquired from reliable sources

Treat every website as potentially harmful

Only open attachments when you know them to be safe

Create and maintain backups

Test software in an isolated environment

Virus Detection

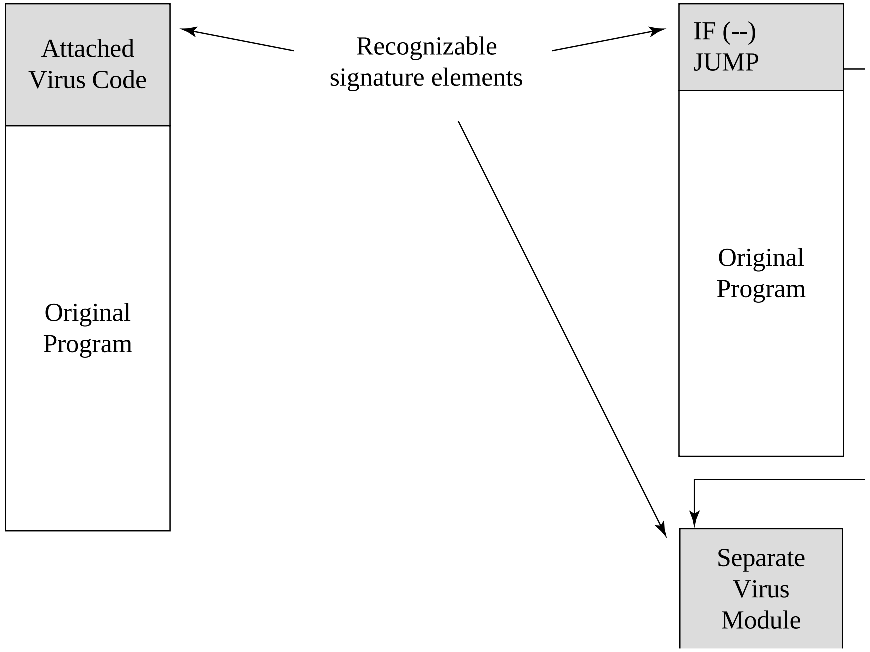
- The pattern which distinguishes a virus is called a **signature**.
    - Anti-viruses (or called virus scanners) look for signatures to identify a virus
    - The signature is part of the virus code

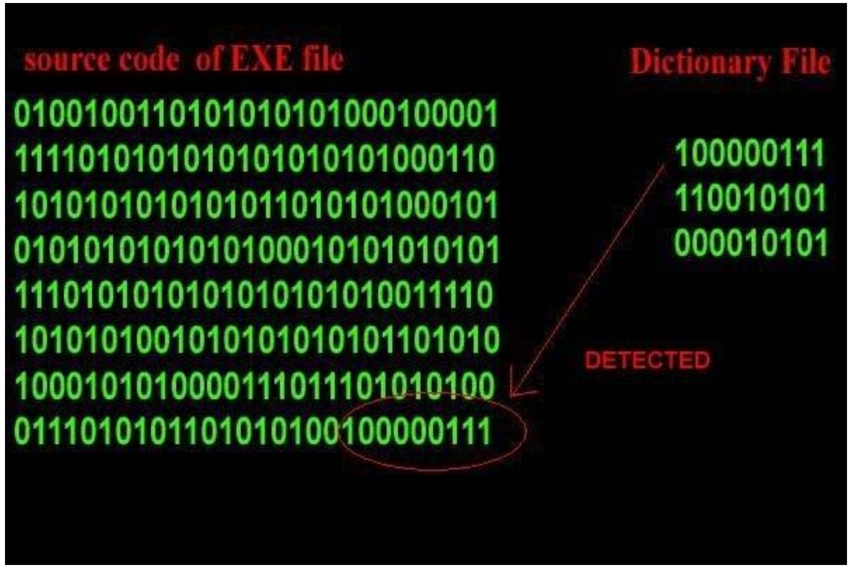
- Traditional virus scanners have **trouble keeping up with new malware**—detect about 45% of infections

- **Detection mechanisms:**
    - Known string patterns in files or memory
    - Execution patterns
    - Storage patterns

- Virus writers can write **viruses** that **can change their patterns** so they are **hard to detect**.
    - A virus that can change its appearance is called a **polymorphic virus**

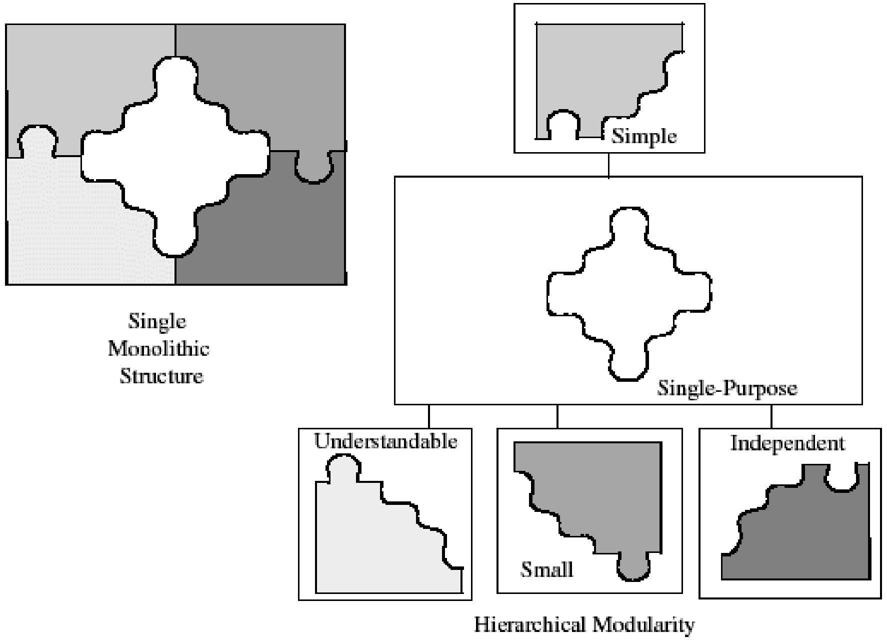
Virus Signatures





https://www.researchgate.net/profile/Vinh_Nguyen144/publication/329327300/figure/fig2/AS:698744064442370@1543604975230/Anti-virus-signatures-based-detection.jpg

Countermeasures for Developers



- ✓ Modular code: Each code module should be
  - Single-purpose
  - Small
  - Simple
  - Independent
- ✓ Encapsulation
- ✓ Information hiding
- ✓ Mutual Suspicion
- ✓ Confinement
- ✓ Genetic diversity

Code Testing

1 **Unit testing**

2 **Integration testing**

3 **Function testing**

4 **Performance testing**

5 **Acceptance testing**

6 **Installation testing**

7 **Regression testing**

8 **Penetration testing**

Design Principles for Security

- ✓ **Least privilege**

  Operate using the fewest privileges possible
- ✓ **Economy of mechanism**

  Protection system should be small, simple, and straightforward.
- ✓ **Open design**

  Should be public, depending on secrecy of relatively few key items, such as a password table. An open design is available for extensive public scrutiny, thus, providing independent confirmation of the design security
  - ✓ **Is it always the case?!**
- ✓ **Complete mediation**

  Every access attempt must be checked

FROM *SECURITY IN COMPUTING, FIFTH EDITION*, BY CHARLES P. PFLEEGER, ET AL. (ISBN: 9780134085043). COPYRIGHT 2015 BY PEARSON EDUCATION, INC. ALL RIGHTS RESERVED.

Design Principles for Security

- ✓ **Permission based**

  Default condition should be denial of access

- ✓ **Separation of privilege**

  Access to objects should depend on more than one condition; e.g., user authentication plus a cryptographic key

- ✓ **Least common mechanism**

  Mechanisms used to access resources should not be shared. LCM concerns the dangers of sharing state among different programs. If one program can corrupt the shared state, it can then corrupt other programs which depend on it

- ✓ **Ease of use**

  If a protection mechanism is hard to use, it is expected to be avoided

Other Countermeasures

Good

- ✓ Proofs of program correctness—where possible
  - Use of verification techniques and formal methods
- ✓ Defensive programming
  - Program designers must not only write correct code but must also anticipate what could go wrong
- ✓ Design by contract
  - Techniques involve formal program development approaches

Bad- still countermeasures but not good practices!!

- ✓ Penetrate-and-patch
- ✓ Security by obscurity

Summary

Buffer overflow attacks can take advantage of the fact that code and data are stored in the same memory in order to maliciously modify executing programs

Programs can have a number of other types of vulnerabilities, including off-by-one errors, incomplete mediation, and race conditions

Malware can have a variety of harmful effects depending on its characteristics, including resource usage, infection vector, and payload

Developers can use a variety of techniques for writing and testing code for security