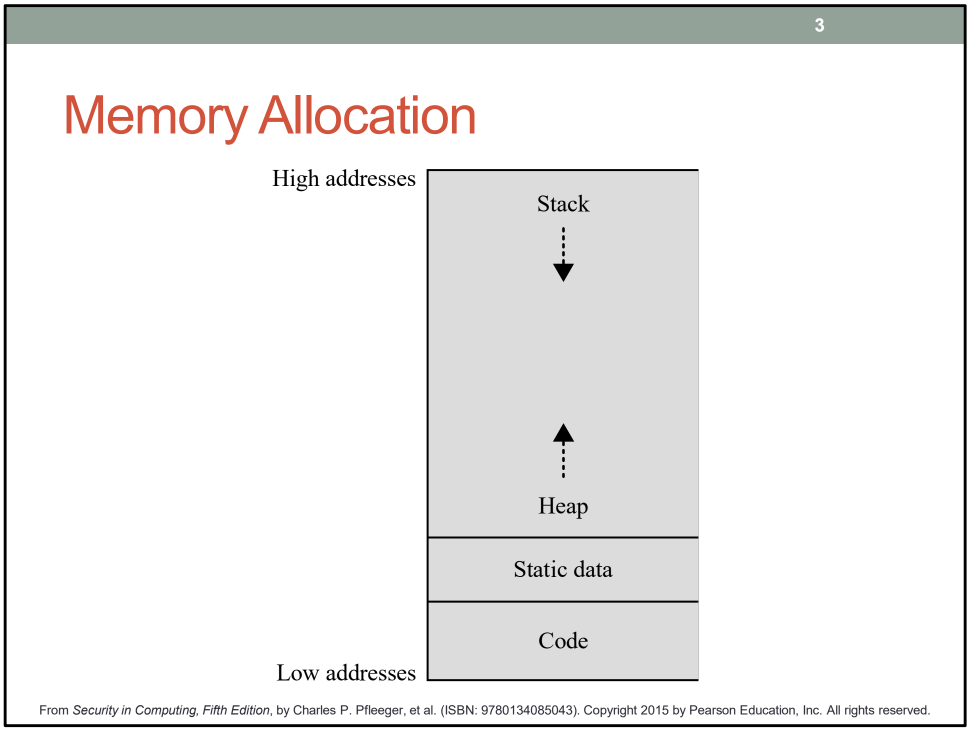
SECURITY IN COMPUTING, FIFTH EDITION

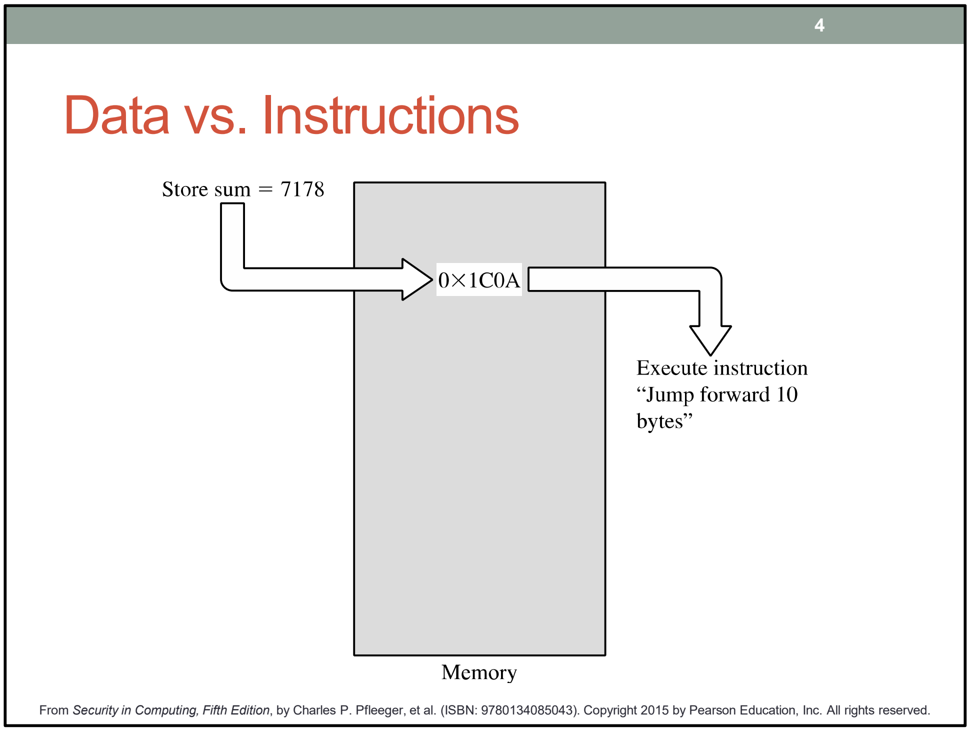
Chapter 3: Programs and Programming

Objectives for Chapter 3

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



Much of this chapter requires basic knowledge of how memory is organized, and this is a nice, simple diagram to refresh students on how it works. The key takeaways: code and data separated, with the heap growing up toward high addresses and the stack growing down from the high addresses.



The same hex value in the same spot in memory can either be a meaningful data value or a meaningful instruction depending on whether the computer treats it as code or data. This will be the basis of the attacks in the following slides.

Buffer Overflows

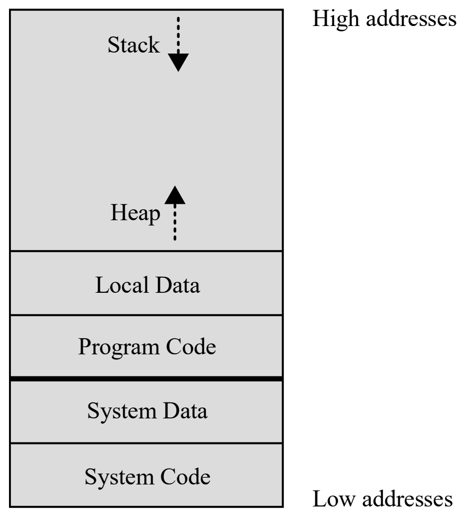
- Occur when data is written beyond the space allocated for it, such as a 10^{th} byte in a 9-byte array
- In a typical exploitable buffer overflow, an attacker's inputs are expected to go into regions of memory allocated for data, but those inputs are instead allowed to overwrite memory holding executable 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

How Buffer Overflows Happen

```
char sample[10];

int i;

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

sample[10] = 'B';
```

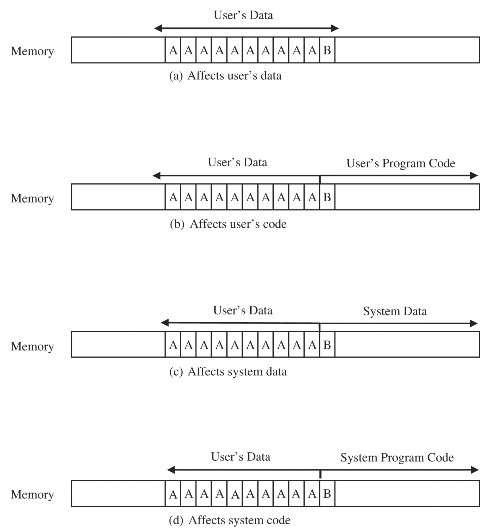

This is a very simple buffer overflow. Character B is placed in memory that wasn't allocated by or for this procedure.

Memory Organization



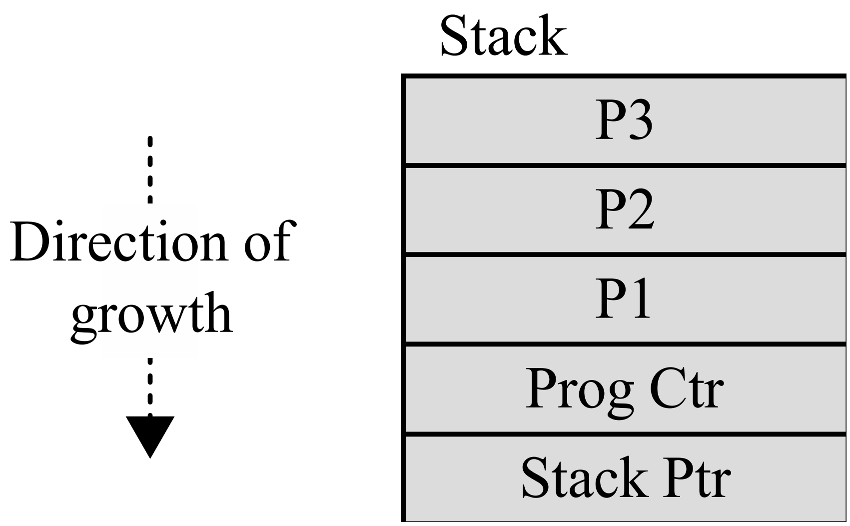
Similar to the earlier picture on memory organization, only this one shows where the system data/code reside vs. where the program code and its local data reside. This context is important for understanding how an attack that takes place inside a given program can affect that program vs. how it can affect the rest of the system.

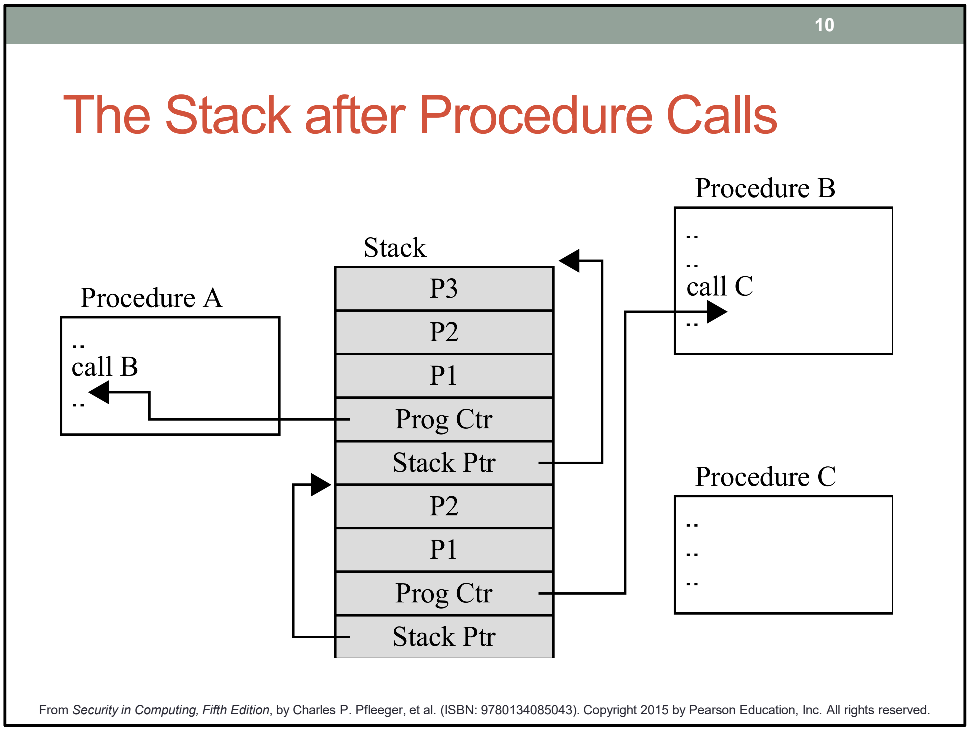
Where a Buffer Can Overflow



Examples of buffer overflow effects in the context of the earlier AAAAAAAAAAB example. The memory that's overwritten depends on where the buffer resides.

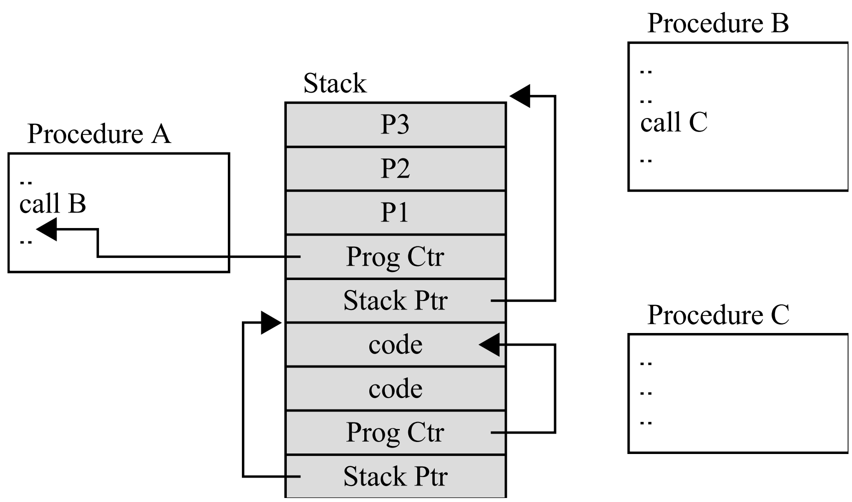
The Stack





When procedure A calls procedure B, procedure B gets added to the stack along with a pointer back to procedure A. In this way, when procedure B is finished running, it can get popped off the stack, and procedure A will just continue executing where it left off.

Compromised Stack



Instead of pointing at procedure B in this case, the program counter is pointing at code that's been placed on the stack as a result of an overflow.

Overwriting Memory for Execution

- Overwrite the program counter stored in the stack
- Overwrite part of the code in low memory, substituting new instructions
- Overwrite 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
- Many languages have overflow protections
- Code analyzers can identify many overflow vulnerabilities
- Canary values in stack to signal modification

Incomplete Mediation

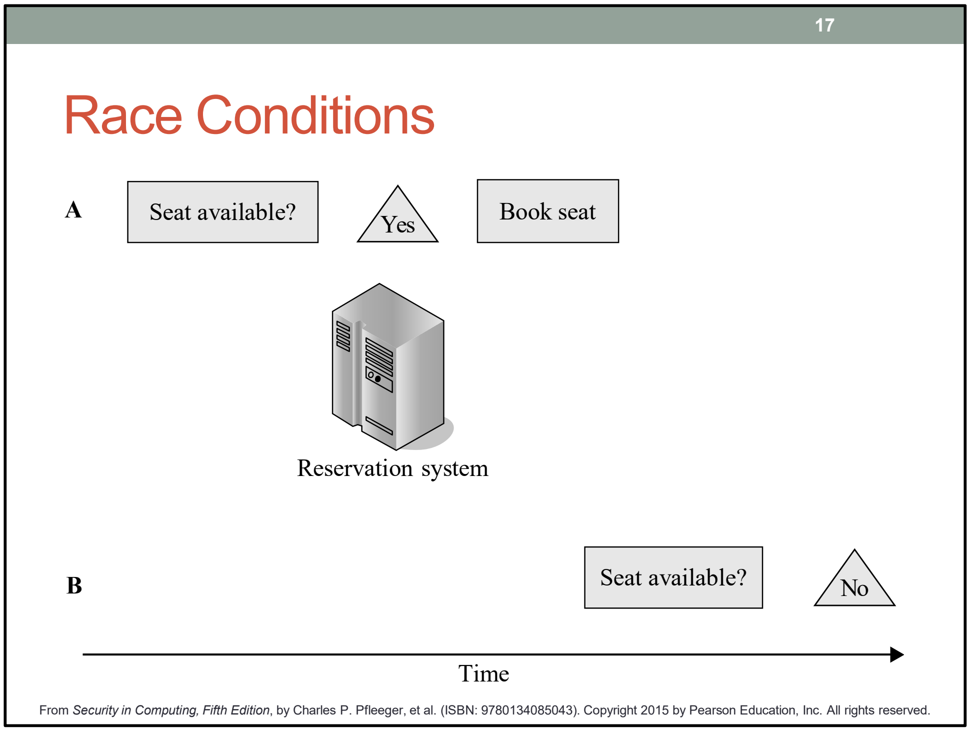
- Mediation: Verifying that the subject is authorized to perform the operation on an object
- Preventing incomplete mediation:
  - Validate all input
  - Limit users' access to sensitive data and functions
  - Complete mediation using a reference monitor

Time-of-Check to Time-of-Use

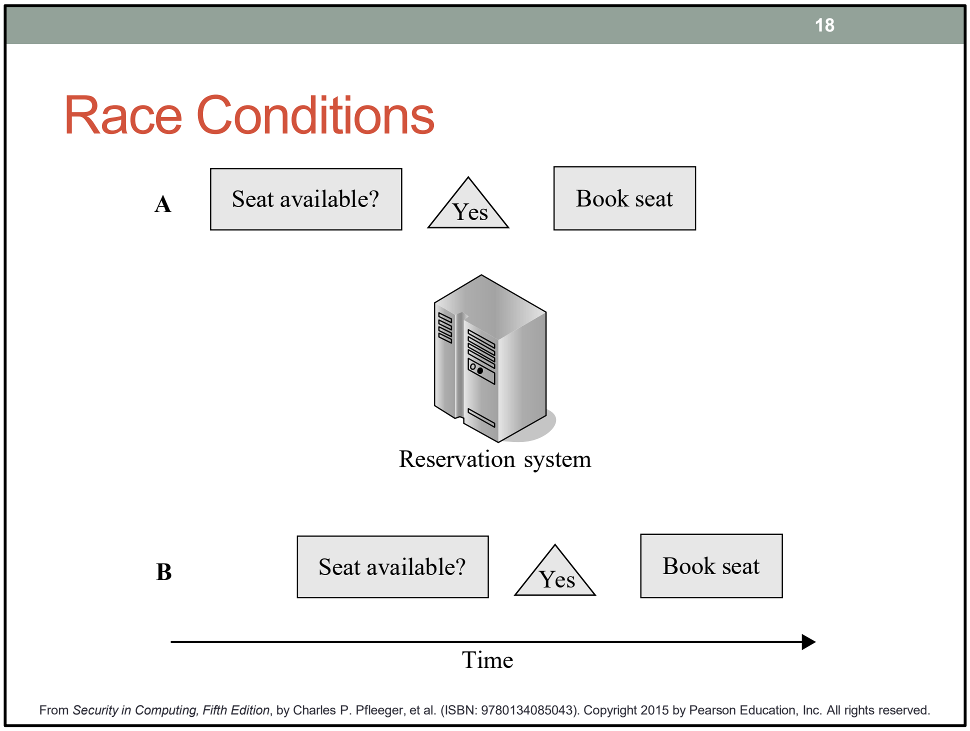
- Mediation performed with a “bait and switch” in the middle

File: my_file	Action: Change byte 4 to A

File: your_file	Action: Delete file



Example 1 (no race condition): A booker books the last seat on the plane, and thereafter the system shows no seat available. See next slide to continue.



Example 2 (race condition): Before the first booker can complete the booking for the last available seat, a second booker looks for available seats. This system has a race condition, where the overlap in timing of the requests causes errant behavior.

Other Programming Oversights

- Undocumented access points (backdoors)
- Off-by-one errors
- Integer overflows
- Unterminated null-terminated string
- Parameter length, type, or number errors
- Unsafe utility libraries

Malware

- Programs planted by an agent with malicious intent to cause unanticipated or undesired effects
- Virus
  - A program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them
- 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

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 (cont.)

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

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 (cont.)

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

Harm from Malicious Code

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

Transmission and Propagation

- Setup and installer program
- Attached file
- Document viruses
- Autorun
- Using nonmalicious programs:
  - Appended viruses
  - Viruses that surround a program
  - Integrated viruses and replacements

Malware Activation

- One-time execution (implanting)
- Boot sector viruses
- Memory-resident viruses
- Application files
- Code libraries

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

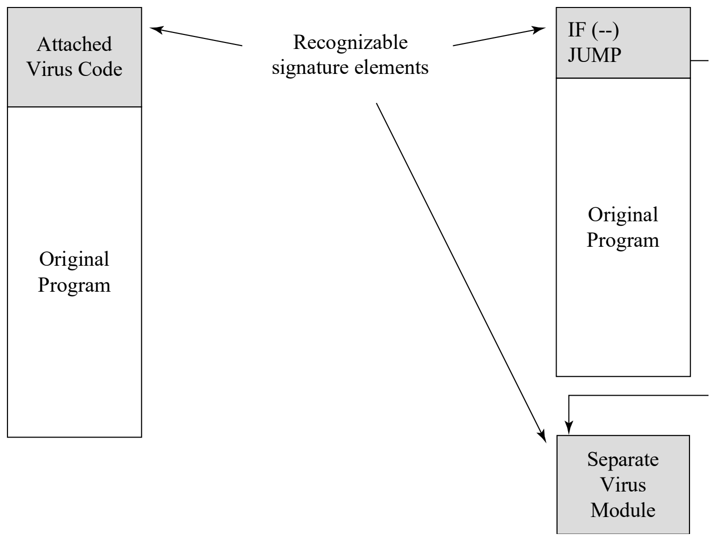
Countermeasures for Users

- Use software acquired from reliable sources
- Test software in an isolated environment
- Only open attachments when you know them to be safe
- Treat every website as potentially harmful
- Create and maintain backups

Virus Detection

- Virus scanners look for signs of malicious code infection using signatures in program files and memory
- 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 Signatures



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

- Unit testing
- Integration testing
- Function testing
- Performance testing
- Acceptance testing
- Installation testing
- Regression testing
- Penetration testing

Design Principles for Security

- Least privilege
- Economy of mechanism
- Open design
- Complete mediation
- Permission based
- Separation of privilege
- Least common mechanism
- Ease of use

Other Countermeasures

- Good
  - Proofs of program correctness—where possible
  - Defensive programming
  - Design by contract
- Bad
  - 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