CS306: Introduction to IT Security

Fall 2018

Lecture 13: Topics

Instructor: **Nikos Triandopoulos**

December 4, 2018

STEVENS
INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

Buffer overflow

Memory basics

Stack

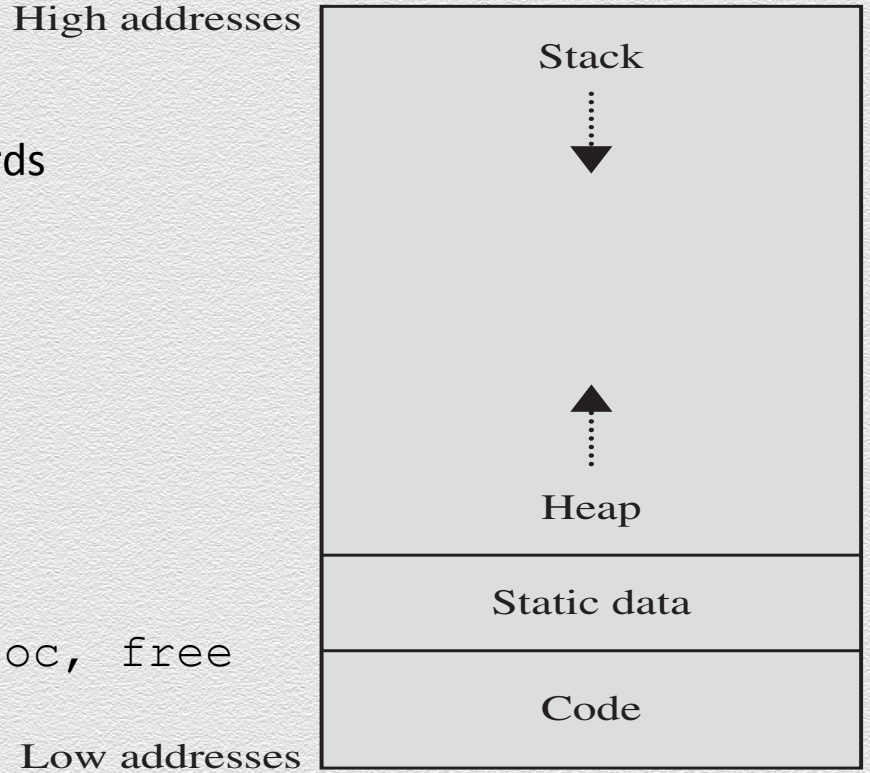
- used whenever a function call is made
- typically higher addresses growing downwards

Static data area

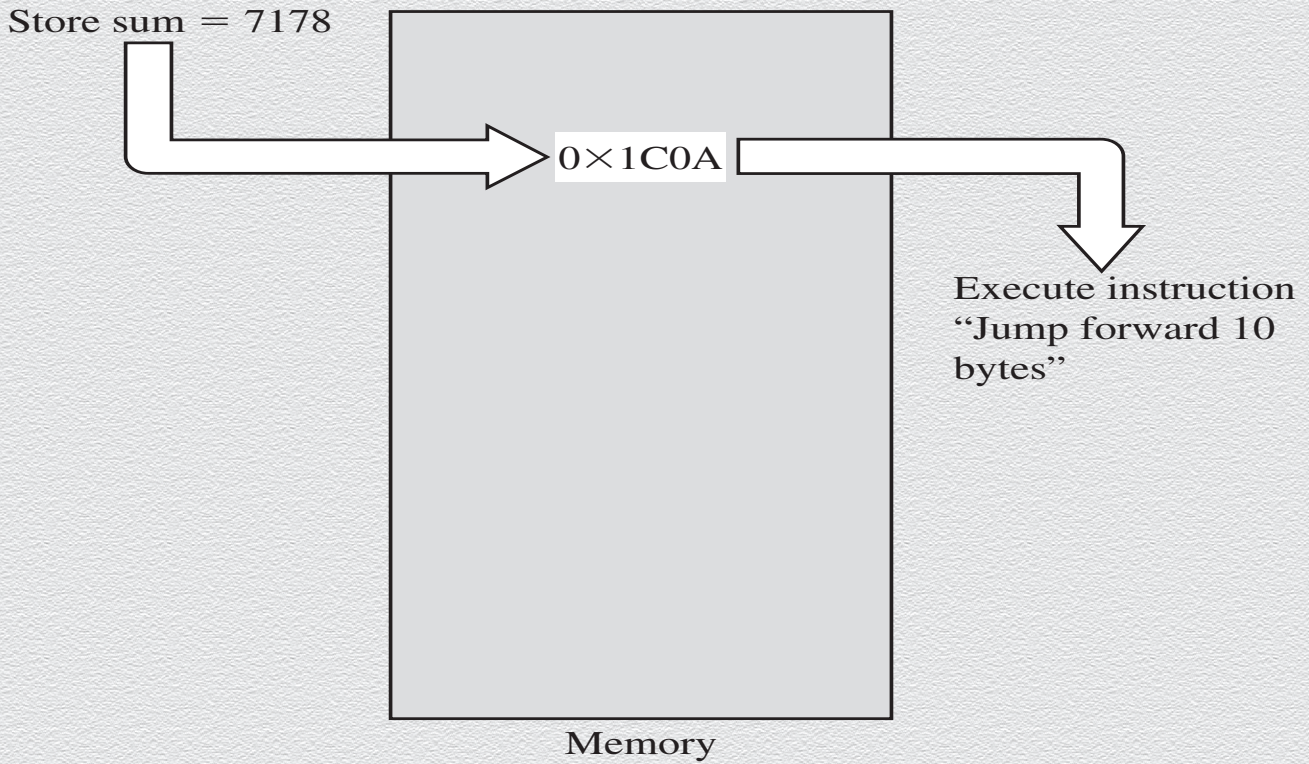
- global variables used by programs (not initialized with zero)
- e.g., `char s[] = "hello world"`

Heap

- begins after data area, growing upwards
- dynamically managed by `malloc, realloc, free`



Data vs. Instructions



Buffer overflows

Based on programmers' oversights (or programming languages **vulnerabilities**)

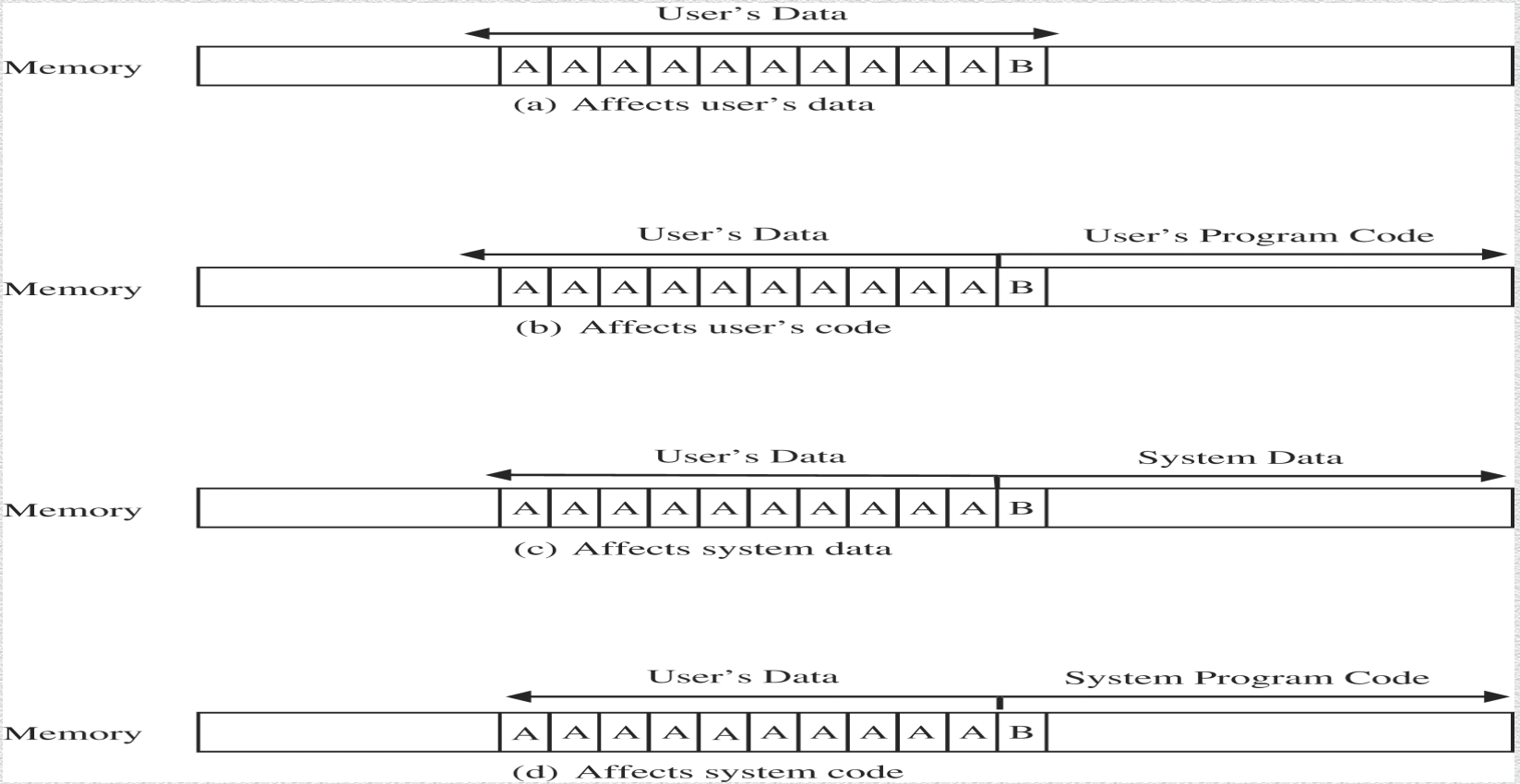
- exploited by attackers by **inputting more data than expected**
  - attacker's data that is written beyond the space allocated for it
  - e.g., a 10^{th} byte in a 9-byte array
- typical exploitable buffer overflow
  - users' inputs are expected to go into regions of memory allocated for data; instead
  - attacker's inputs **are allowed to overwrite** memory holding executable code
- attacker's challenge is to discover buffer-overflow vulnerabilities
  - find opportunities **leading to overwritten memory being executed**
  - **find the right code to input** (that inflicts some specific harm)

Example: How buffer overflows happen

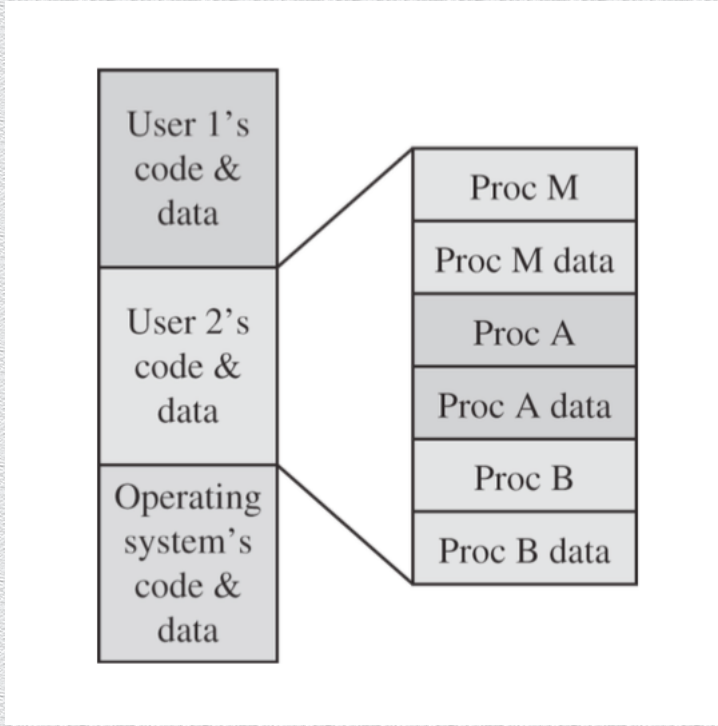
```
char sample[10];

int i;

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

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

(or `sample[i] = 'B';`)

Overflows can affect data or code, or even the OS



Overflows can affect other users

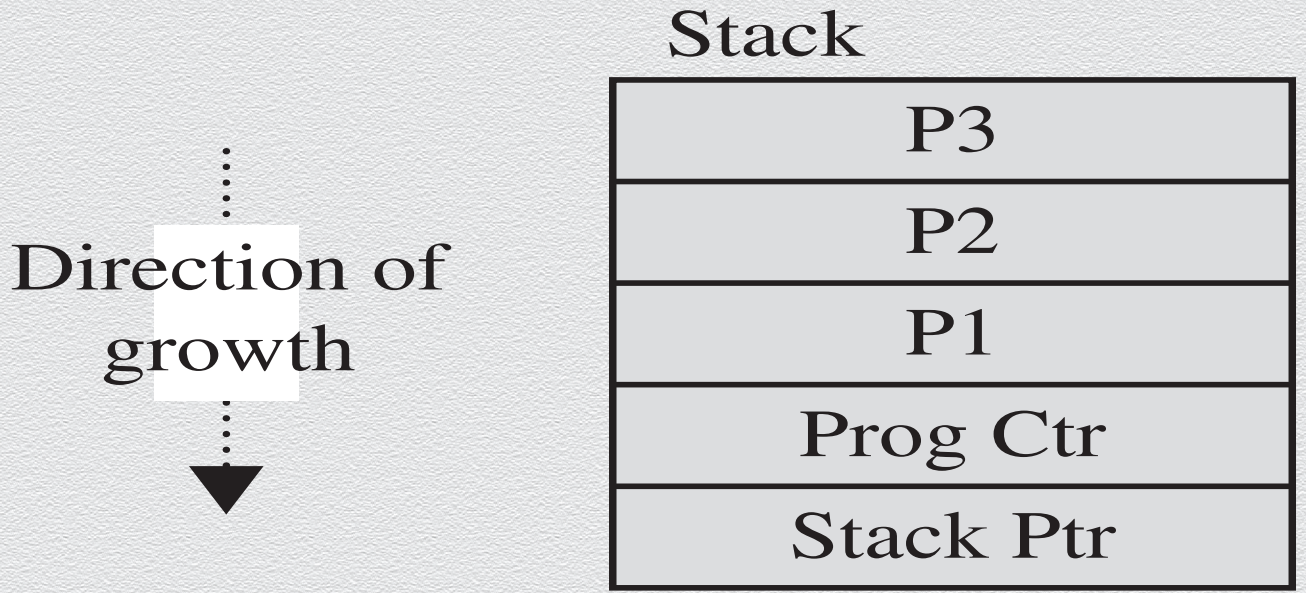


Harm from buffer overflows

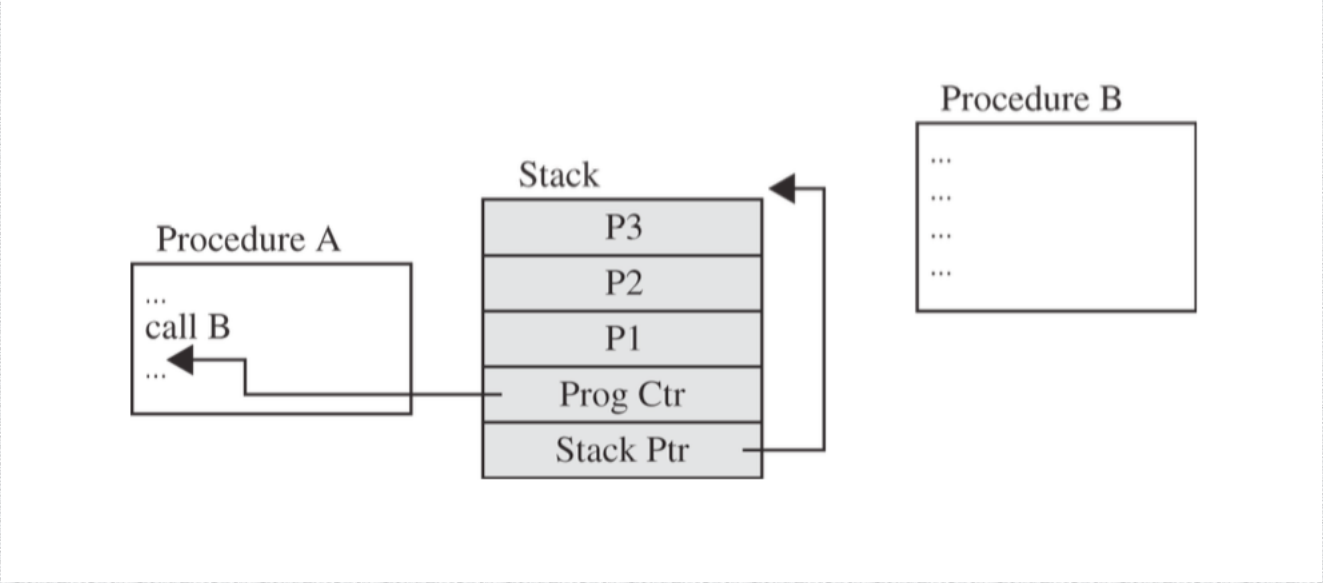
- overwrite
  - an instruction or data item of same program's data
    - e.g., PC and data in the stack so that PC points to the stack
  - data or code belonging to another program or the OS
    - e.g., part of the code in low memory, substituting new instructions
    - gives to attacker that program's execution privileges or root privileges
- results in
  - **unauthorized access**
  - **privilege escalation**

When successfully completed, attacker runs **maliciously written code** at **higher privilege levels**!

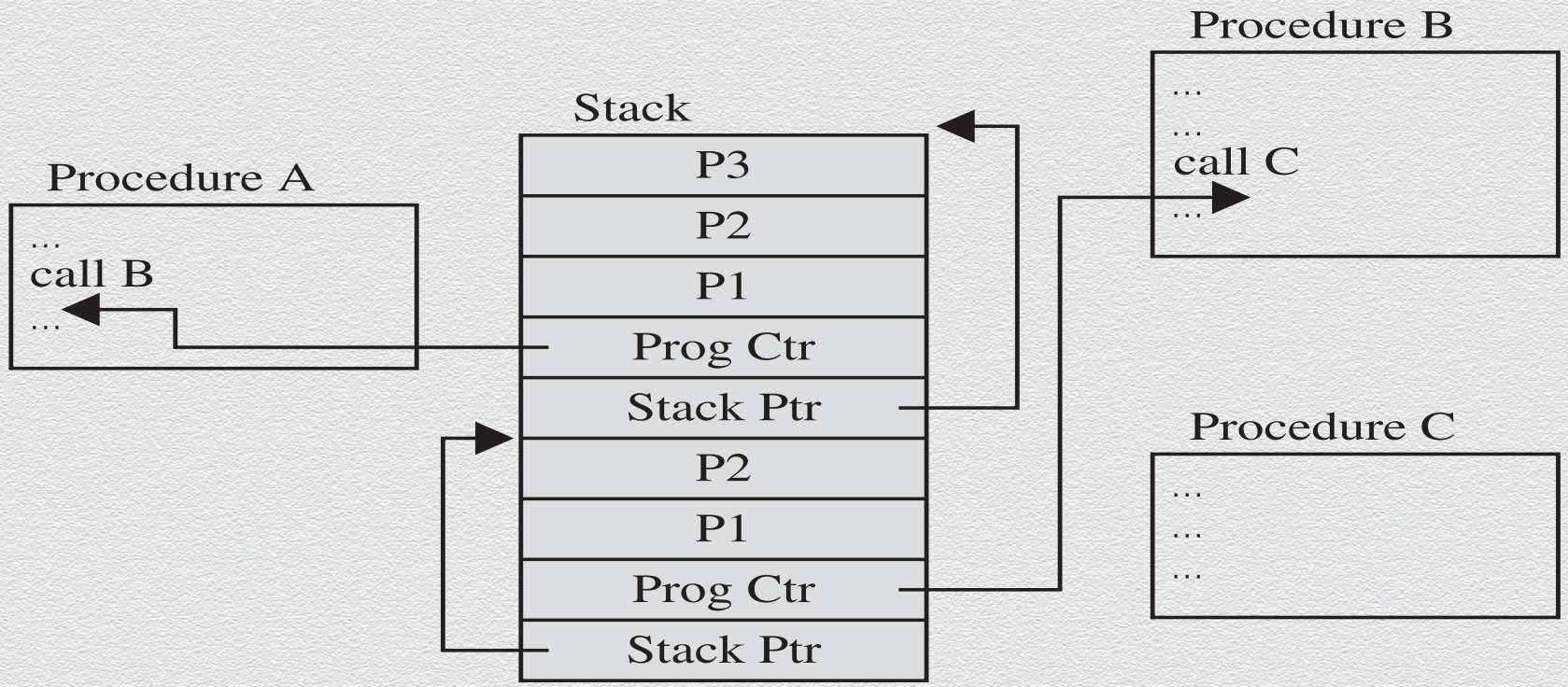
The stack



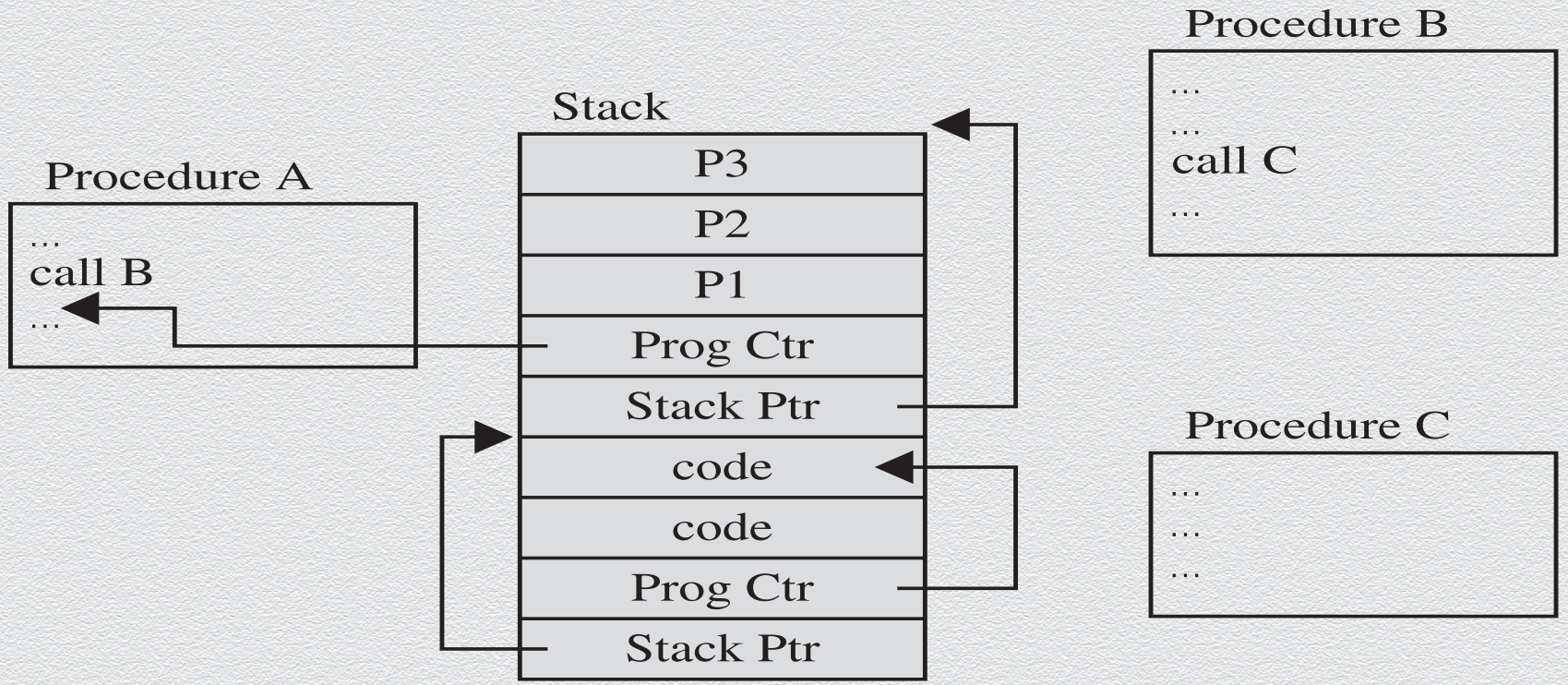
The stack after procedure calls: A calls B



The stack after nested procedure calls: A calls B, B calls C



Compromised stack



Attack structure

To exploit a buffer overflow vulnerability the attacker must address some challenges

[1] write malicious code (that does some harm)

- not trivial task (depends on next steps/challenges)
- e.g., a special type of malicious code called **shellcode** can be written

[2] inject the malicious code into the memory of the target program (TP)

- control the contents of the buffer in TP
- e.g., in following example, **by storing the malicious code in the input file**

[3] jump to (and execute) the malicious code

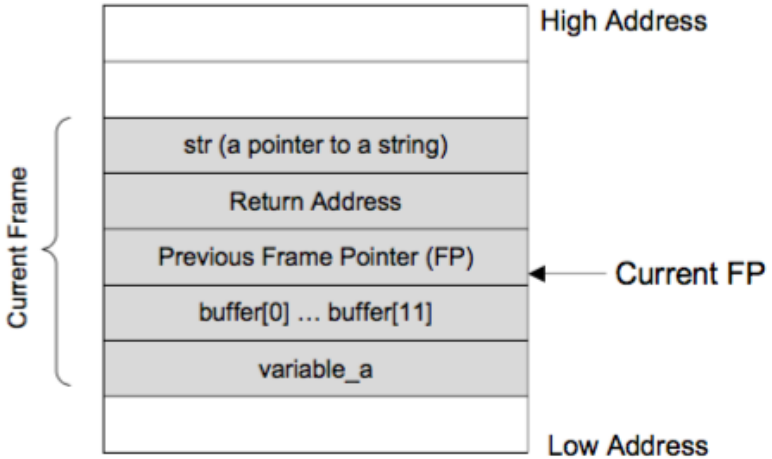
- control the execution of TP and execute injected malicious code
- e.g., in following example, **by pointing the program counter to the right position in the stack**

Buffer-overflow vulnerability: A specific example

- layout of stack after the program execution has entered function func()
- grows from-high-to-low addresses (but `buffer` grows from-low-to-high)

```
void func (char *str) {
  char buffer[12];
  int  variable_a;
  strcpy (buffer, str);
}

Int main() {
  char *str = "I am greater than 12 bytes";
  func (str);
}
```

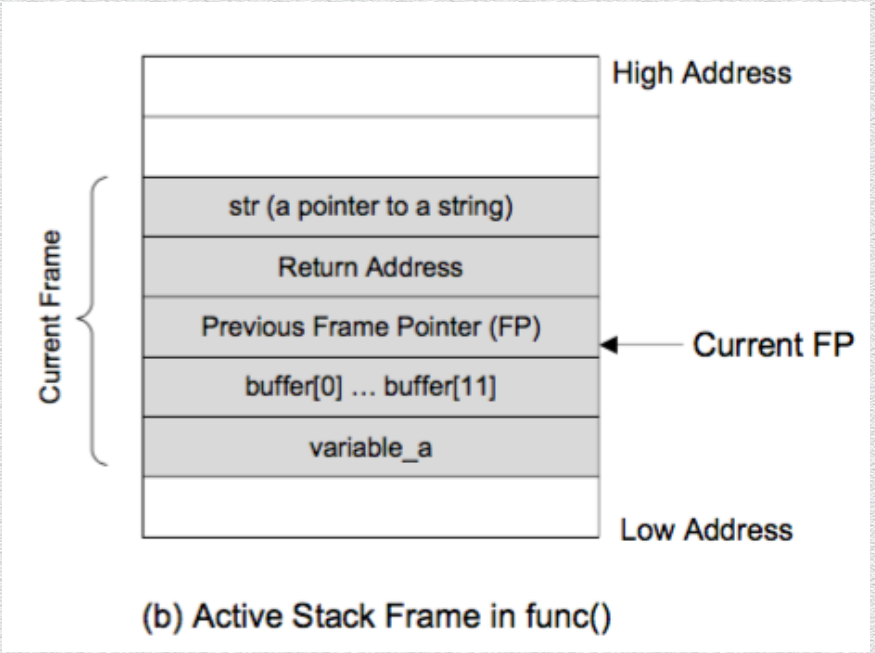
(a) A code example



(b) Active Stack Frame in func()

Data stored in current frame

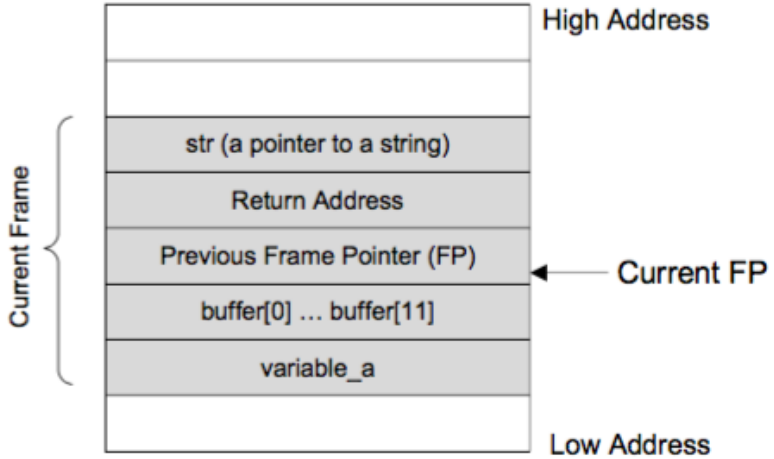
- local data: `buffer, variable_a`
- function parameter: `str`
- **return address**
  - what to execute after function ends
  - command after function call
- frame pointer (FP)
  - pointer on current frame that is used to reference local data & function parameters
  - e.g., `variable_a` is referred to as FP-16, `buffer` as FP-12, `str` as FP+8
- previous frame pointer
  - pointer to previous frame (corresponding to function that called `func()`)



Buffer overflow: [2] Inject malicious code

- `strcpy(buffer, str)` copies the contents from `str` to `buffer[]`
- the string pointed by `str` has more than 12 chars, while the size of `buffer[]` is only 12
- `strcpy()` does not check whether the boundary of `buffer[]` has reached
  - it only stops when seeing the end-of-string character '\0'
- contents in the memory above `buffer[]` will be overwritten by the characters at the end of `str`

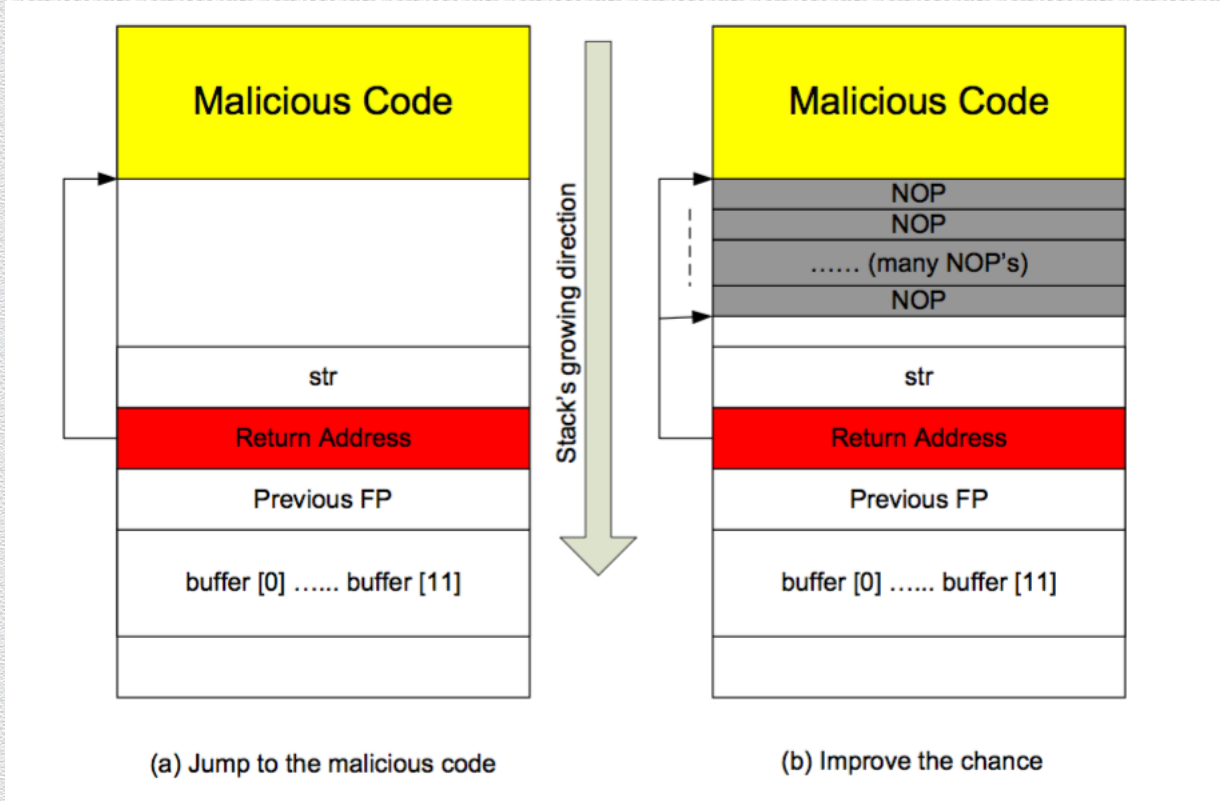
```
void func (char *str) {
  char buffer[12];
  int  variable_a;
  strcpy (buffer, str);
}

Int main() {
  char *str = "I am greater than 12 bytes";
  func (str);
}
```



[2] Inject malicious code: A more interesting example

```
/* stack.c */ /* This program has a buffer overflow vulnerability. */

#include <stdlib.h> #include <stdio.h> #include <string.h>
int func (char *str) {
char buffer[12];
strcpy(buffer, str); /* This statement has a buffer overflow problem */
return 1; }

int main(int argc, char **argv) {
char str[517];
FILE *badfile;
badfile = fopen("badfile", "r");
fread(str, sizeof(char), 517, badfile);
func (str);
printf("Returned Properly\n");
return 1; }
```

[3] Jump to the malicious code



[3] Jump to the malicious code

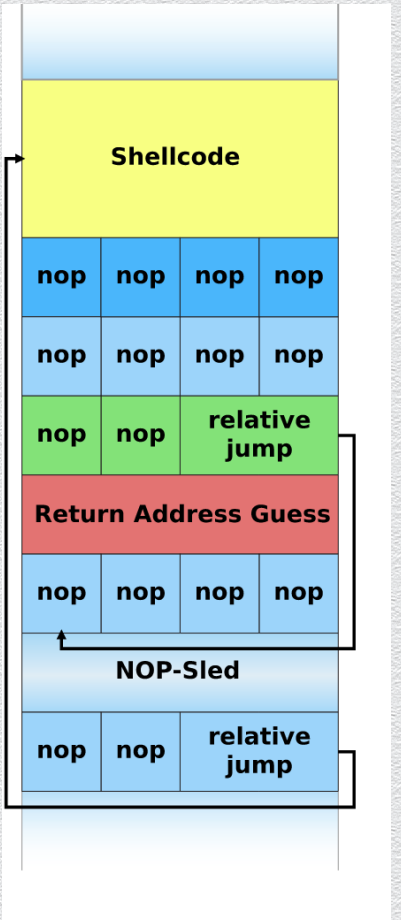
To run the malicious code (already injected in TP's stack)

- need to **know the absolute address of the malicious code**
  - overflow the buffer so that this address overwrites the return address
  - when function returns, the malicious code will run
- strategies to find where the malicious code starts
  - make a copy of the TP and find (approximate) the start of malicious code by debugging
  - set-UID TP: allows to run an executable with the privileges of the executable's owner
- if the TP runs remotely, you can always guess
  - stack usually starts at the same address and is not very deep
  - range of addresses to guess is actually quite small

[3] Jump to the malicious code: Nopsled

To improve the chance of success

- add many NOP operations to the beginning of the malicious code
- NOP (no operation) is a special instruction
  - does nothing other than advancing to the next instruction
  - therefore, as long as the guessed address points to one of the NOPs, the attack will be successful!
  - with NOPs, the chance of guessing the correct entry point to the malicious code is significantly improved!



[1] Write malicious code: Shellcode

Powerful code that invokes a shell

- attacker can run any command in that shell!
- if TP has root privileges, then any command runs also at root level!
- e.g., C program that simply launches a shell:

```
#include <stdio.h>
int main( ) {
char *name[2];
name[0] = ''/bin/sh'';
name[1] = NULL;
execve(name[0], name, NULL); }
```

[1] Write malicious code: Further challenges

Directly compiling the previous program into binary code is not enough

- (1) to invoke system call `execve()`, need to know the address of the string "`/bin/sh`"
  - storing and deriving the address of this argument is not easy
- (2) function `strcpy()` will stop in the first occurrence of a `NULL` (i.e., 0) value

- e.g., C program that simply launches a shell:

```
#include <stdio.h>
int main( ) {
char *name[2];
name[0] = ``/bin/sh'';
name[1] = NULL;
execve(name[0], name, NULL); }
```

[1] Write malicious code: Solutions

Directly compiling the previous program into binary code is not enough

- (1) to invoke system call `execve()`, need to know the address of the string "`/bin/sh`"
  - **push string "`/bin/sh`" onto stack and use the stack pointer `esp` to get its location**
- (2) function `strcpy()` will stop in the first occurrence of a `NULL` (i.e., 0) value
  - **convert instructions containing 0 into equivalent instructions not containing 0**
  - **e.g., to store 0 to a register, use `XOR` operation, instead of directly assigning 0**
- e.g., C program that simply launches a shell:

```
#include <stdio.h>
int main( ) {
char *name[2];
name[0] = ''/bin/sh'';
name[1] = NULL;
execve(name[0], name, NULL); }
```

[1] Write malicious code: Final malicious code

```
xorl %eax,%eax
pushl %eax              # push 0 into stack (end of string)
pushl $0x68732f2f       # push "//sh" into stack
pushl $0x6e69622f       # push "/bin" into stack
movl %esp,%ebx          # %ebx = name[0]
pushl %eax              # name[1]
pushl %ebx              # name[0]
movl %esp,%ecx          # %ecx = name
cdq                     # %edx=0
movb $0x0b,%al
int $0x80               # invoke execve(name[0], name, 0)
```

Other software security topics

Incomplete mediation

- Access control
  - what subject can perform what operation on what object
- Mediation (means 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
  - complete mediation using a reference monitor
    - access control that is always invoked, tamperproof and verifiable

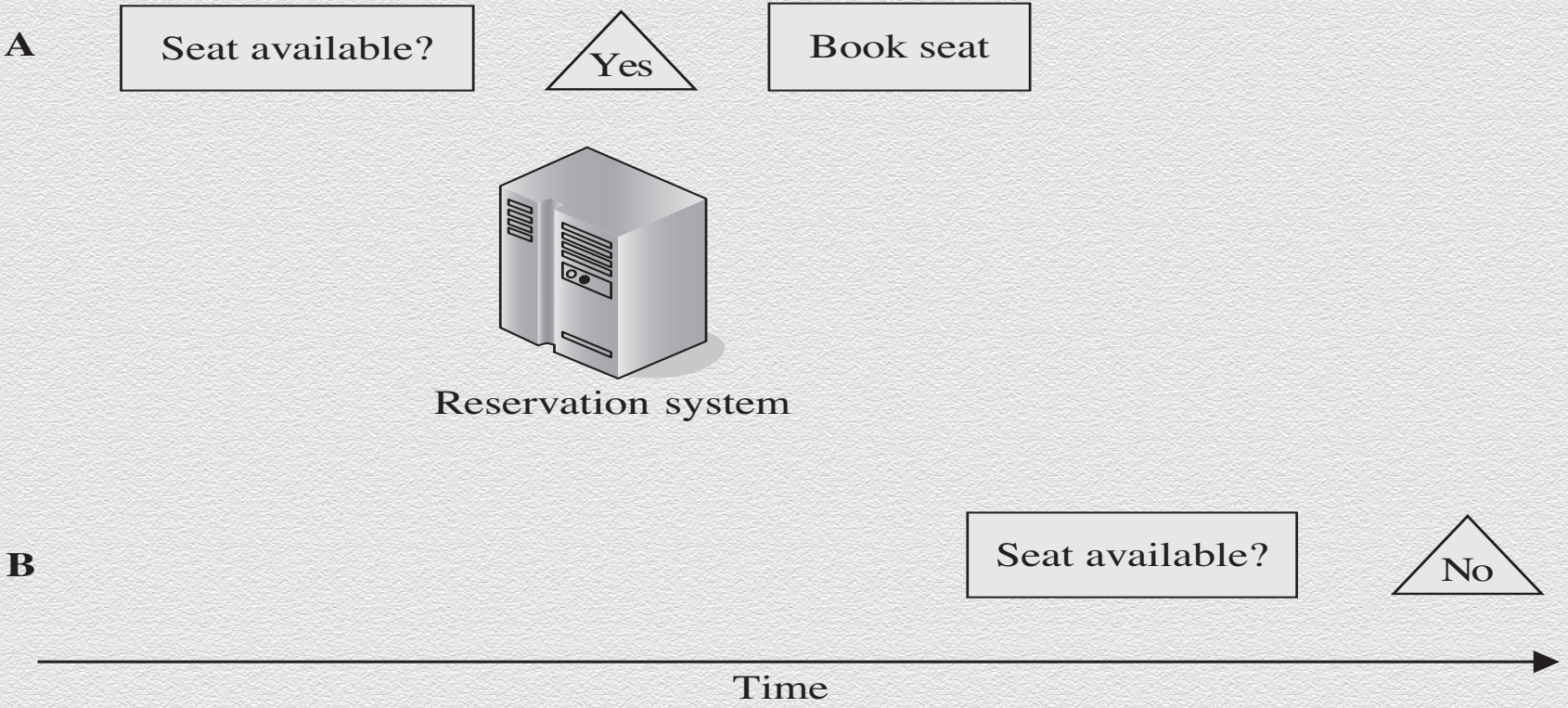
Time-of-Check to Time-of-Use

- mediation performed with a “bait and switch” in the middle
- between access check and resource use, data should remain unchanged
- exploits the details in the two processes

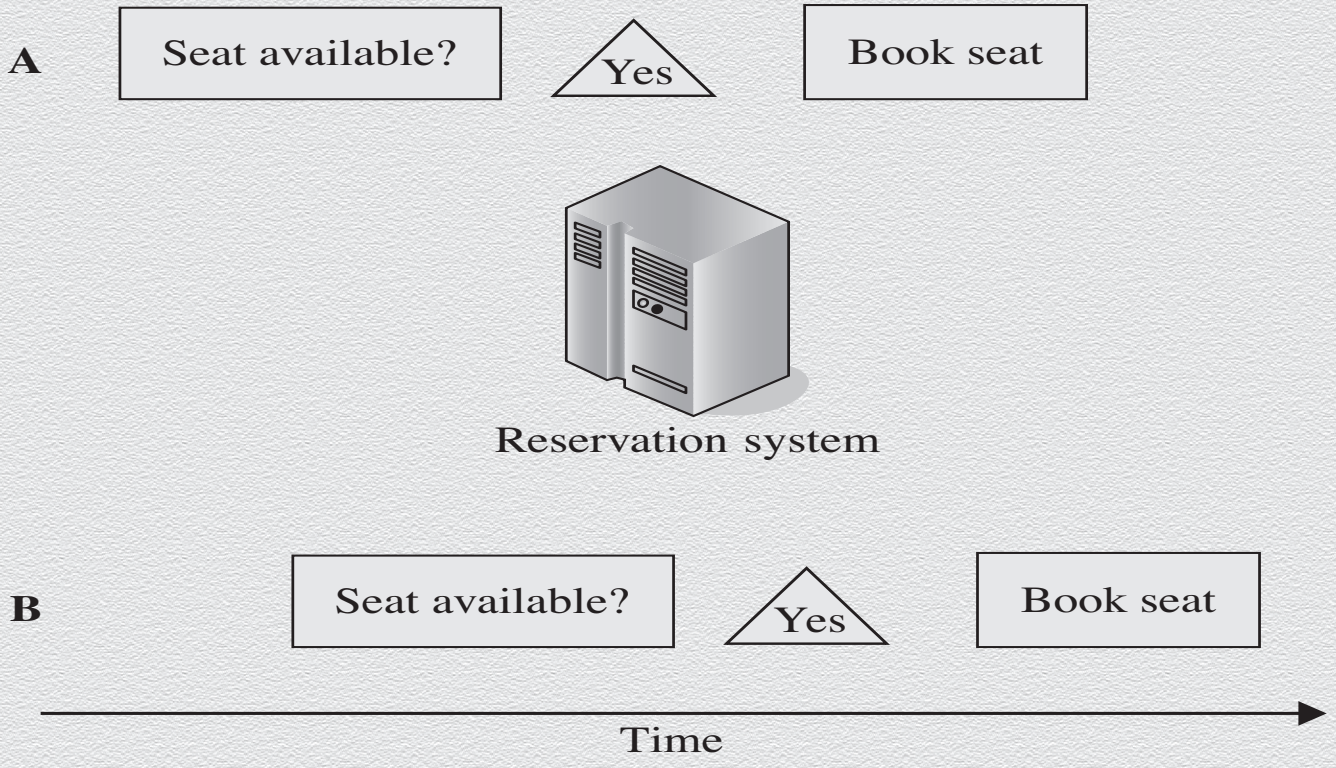
File: my_file	Action: Change byte 4 to A

File: your_file	Action: Delete file

Race conditions



Race conditions



Other programming oversights

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

Malware

Malware

- Programs planted by an agent with malicious intent
  - to cause unanticipated or undesired effects
- Virus
  - a program that can replicate itself
    - pass on malicious code to other non-malicious 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 non-malicious 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

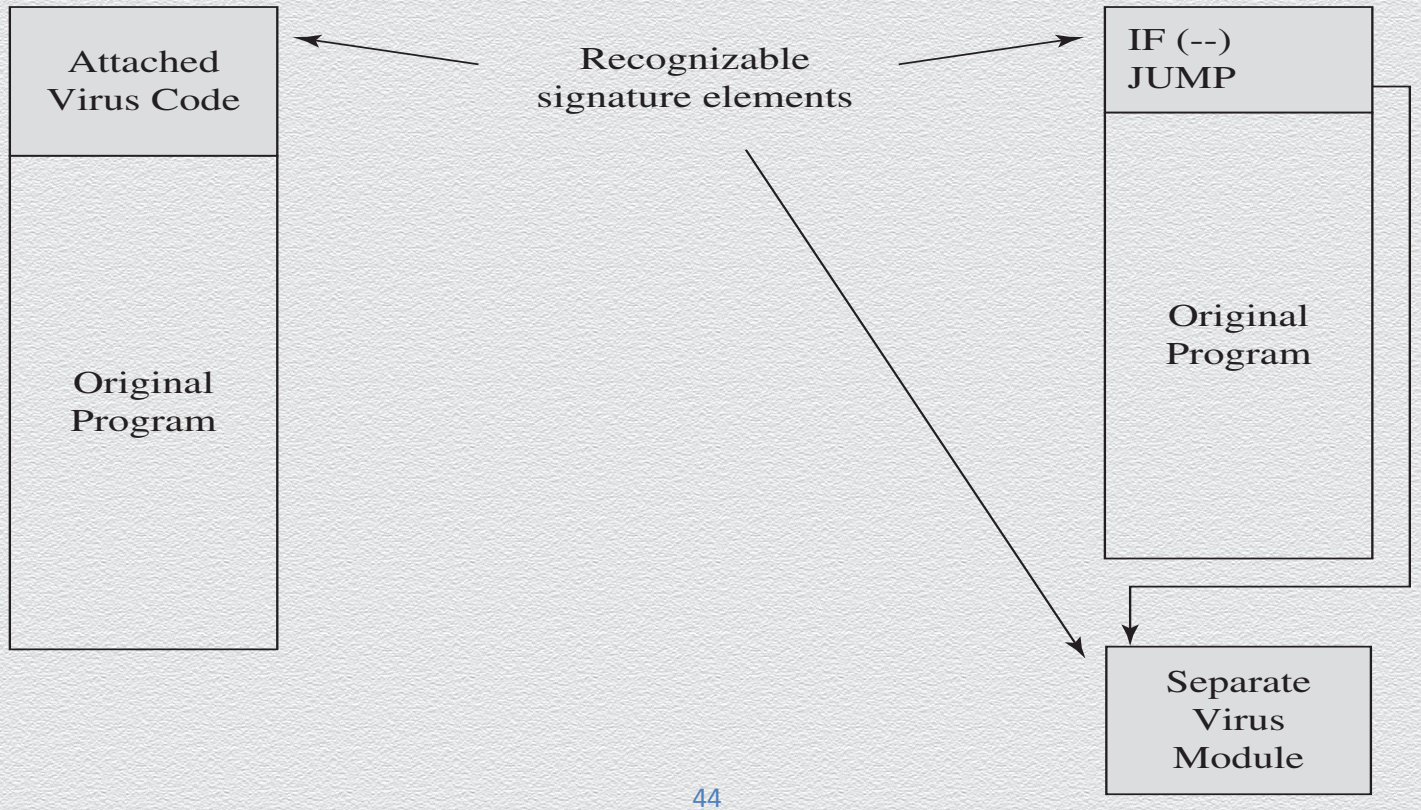
Countermeasures for users

- 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

Legal issues & Ethics

Protecting programs & data

- Copyrights
  - Designed to protect the expression of ideas
  - Gives the author the exclusive right to make copies of the expression and sell them to the public
- Patents
  - Designed to protect inventions, tangible objects, or ways to make them
  - Patents were intended to apply to the results of science, technology, and engineering as opposed to arts and writing
- Trade secrets
  - Information that gives one company a competitive advantage over others
  - Must be closely guarded as a secret, or legal protections are lost

Copyrights

- What
  - In US, copyright can be registered for original works of expression but not for ideas
- “Fair use”
  - allows copyrighted material to be used in ways that do not interfere with author’s rights
  - criticism, comment, new reporting, teaching, scholarship, or research
- Software can be copyrighted
  - the code is protected but the algorithms are not
  - if source code is not published (i.e., only compiled code is published), copyright may not apply

Patents

- Novelty requirement
  - cannot be obvious to a person ordinarily skilled in the relevant field
- Must convince the patent office that the invention deserves a patent
  - i.e., that it is novel
- A patent holder must oppose all infringement or risk losing the patent rights
- Since 1981, patent law has extended to include computer software
  - recognizing that algorithms are inventions

Trade secrets

- If someone obtains a trade secret improperly and profits from it, the owner can recover profits, damages, lost revenues, and legal costs
- If someone else happens to discover the secret independently, there is no infringement
- Reverse engineering a trade secret is not infringement
- Trade secrets can protect secret computer algorithms from being used in other products
  - cannot provide legal protection against software piracy
  - the challenge of using trade secrets to protect software is that software can be effectively reverse engineered

Comparing copyrights, patents, and trade secrets

	Copyright	Patent	Trade Secret
Protects	Expression of idea, not idea itself	Invention—the way something works	A secret, competitive advantage
Protected object made public	Yes; intention is to promote publication	Design filed at Patent Office	No
Requirement to distribute	Yes	No	No
Ease of filing	Very easy, do-it-yourself	Very complicated; specialist lawyer suggested	No filing
Duration	Varies by country; approximately 75–100 years is typical	19 years	Indefinite
Legal protection	Sue if unauthorized copy sold	Sue if invention copied	Sue if secret improperly obtained

Special characteristics of information

- Like material goods, information is valuable
- Unlike material goods, information
  - Is not depletable
  - Can be replicated
  - Has a minimal marginal cost
  - Often has a time-dependent value
  - Often transferred intangibly
- All of these factors impact how information is treated under the law

Rights of Employees and Employers

- Ownership of a patent
  - An employer has the right to patent if the employee's job functions included inventing the product. Even if an employee patents something, the employer can argue for a right to use the invention if the employer contributed some resources
- Ownership of a copyright
  - Similar to patent
- Licenses
  - In return for a fee, a programmer grants a company a license to use her program. The license can include many factors, such as time period, number of users, number of systems, and so on
- Trade secret protection
  - A company owns the trade secrets of its business-confidential data. As with copyrights and patents, an employer can argue about having contributed to the development of trade secrets

Reporting Software Flaws

- A proposed model for responsible vulnerability reporting:
  - The vendor must acknowledge a vulnerability report confidentially to the reporter.
  - The vendor must agree that the vulnerability exists (or argue otherwise) confidentially to the reporter.
  - The vendor must inform users of the vulnerability and any available countermeasures within 30 days or request additional time from the reporter as needed.
  - After informing users, the vendor may request from the reporter a 30-day quiet period to allow users time to install patches.
  - At the end of the quiet period, the vendor and reporter should agree on a date at which time the vulnerability information may be released to the general public.
  - The vendor should credit the reporter with having located the vulnerability.
  - If the vendor does not follow these steps, the reporter should work with a coordinator to determine a responsible way to publicize the vulnerability.

Computer crime

- Rules of property
  - Most laws have evolved to recognize data and computer services as property.
- Rules of evidence
  - Demonstrating authenticity of computer-based evidence is a challenge.
  - Chain of custody: Law enforcement track clearly and completely the order and identity of people who had access to evidence in an effort to demonstrate that no one had the opportunity to tamper.
- Threats to integrity and confidentiality
  - Laws have evolved to recognize breaches of privacy and damage to data as crimes.
- Value of data
  - Digital data, from a legal perspective, is now considered to be worth what a buyer would be willing to pay for it.

Computer crime is hard to prosecute

- Lack of domain understanding by courts, lawyers, law enforcement, and jurors
- Lack of physical evidence
- Lack of political impact because direct harm to people is harder to identify
- Complexity of cases
- Ages of defendants, who are more likely than many other serious criminals to be juvenile
- Even when there is clear evidence of a crime, the victim (e.g., banks) may not wish to prosecute because they may lose the trust of their customers

Example computer statutes

- US Computer Fraud and Abuse Act
  - Prohibits computer fraud, trafficking in passwords, transmitting code that damages a system, unauthorized access to systems
- US Economic Espionage Act
  - Outlaws use of a computer for foreign espionage
- US Freedom of Information Act
  - Provides public access to information collective by the executive branch of the US government
- US Privacy Act
  - Protects privacy of personal data collected by the government

Example computer statutes (cont.)

- US Electronic Communications Privacy Act
  - Protects against electronic wiretapping
- Gramm-Leach-Bliley Act
  - Requires financial institutions to undergo security risk assessments, adopt a program to protect customers' nonpublic personal information, and provide customers with privacy policies
- HIPAA
  - Requires protection of the privacy of individuals' medical records
- USA Patriot Act
  - Gave law enforcement an easier path to obtaining wiretaps on potential foreign agents and made damaging computer systems a felony

Example computer statutes (cont.)

- The CAN SPAM Act
  - Bans deceptive email advertising, requires opt-out options
- California Breach Notification
  - Requires any company doing business in California to notify individuals of any breach that is reasonably believed to have compromised personal information of a California resident
- Council of Europe Agreement on Cybercrime
  - Requires signing countries to define cybercrime activities and support their investigation and prosecution across national boundaries
- EU Data Protection Act
  - Established privacy rights and protection responsibilities for all citizens of member countries

Comparing law and ethics

Law	Ethics
Described by formal, written documents	Described by unwritten principles
Interpreted by courts	Interpreted by each individual
Established by legislatures representing all people	Presented by philosophers, religions, professional groups
Applied to everyone	Chosen personally
Priority determined by courts if two laws conflict	Priority determined by an individual if two principles conflict
”Right” arbitrated finally by court	Not arbitrated externally
Enforced by police and courts	Enforced by intangibles such as principles and beliefs

Examining a situation for ethical issues

- Understand the situation
- Know several theories of ethical reasoning
- List the ethical principles involved
- Determine which principles outweigh others
- Make and defend an ethical choice

Bases of ethical theories

	Consequence-Based	**Rule-Based**
Individual	based on consequences to individual	based on rules acquired by the individual—from religion, experience, analysis
Universal	based on consequences to all of society	based on universal rules, evident to everyone

Situation I: Use of computer services

- Dave works as a programmer for a large software company. He writes and tests utility programs such as compilers. His company operates two computing shifts: During the day, program development and online applications are run; at night, batch production jobs are completed. Dave has access to workload data and learns that the evening batch runs are complementary to daytime programming tasks; that is, adding programming work during the night shift would not adversely affect performance of the computer to other users.
- Dave comes back after normal hours to develop a program to manage his own stock portfolio. His drain on the system is minimal, and he uses very few expendable supplies, such as printer paper. Is Dave's behavior ethical?

Situation II: Privacy rights

- Donald works for the county records department as a computer records clerk, where he has access to files of property tax records. For a scientific study, a researcher, Ethel, has been granted access to the numerical portion—but not the corresponding names—of some records.
- Ethel finds some information that she would like to use, but she needs the names and addresses corresponding with certain properties. Ethel asks Donald to retrieve the names and addresses so she can contact these people for more information and for permission to do further study.
- Should Donald release the names and addresses?

Situation III: Denial of service

- Charlie and Carol are students at a university in a computer science program.
- Each writes a program for a class assignment.
- Charlie's program happens to uncover a flaw in a compiler that ultimately causes the entire computing system to fail; all users lose the results of their current computation.
- Charlie's program uses acceptable features of the language; the compiler is at fault.
- Charlie did not suspect his program would cause a system failure. He reports the program to the computing center and tries to find ways to achieve his intended result without exercising the system flaw.

Situation III: Denial of service (part II)

- The system continues to fail periodically, for a total of 10 times (beyond the first failure).
- When the system fails, sometimes Charlie is running a program, but sometimes Charlie is not.
- The director contacts Charlie, who shows all his program versions to the computing center staff.
- The staff concludes that Charlie may have been inadvertently responsible for some, but not all, of the system failures, but that his latest approach to solving the assigned problem is unlikely to lead to additional system failures.

Situation III: Denial of service (part III)

- On further analysis, the computing center director notes that Carol has had programs running each of the first eight (of 10) times the system failed.
- The director uses administrative privilege to inspect Carol's files and finds a file that exploits the same vulnerability as did Charlie's program.
- The director immediately suspends Carol's account, denying Carol access to the computing system.
- Because of this, Carol is unable to complete her assignment on time; she receives a D in the course, and she drops out of school.

Situation IV: Ownership of Programs

- Greg is a programmer working for a large aerospace firm, Star Computers, which works on many government contracts; Cathy is Greg's supervisor.
- Greg is assigned to program various kinds of simulations.
- To improve his programming abilities, Greg writes some programming tools, such as a cross-reference facility and a program that automatically extracts documentation from source code.
- These are not assigned tasks for Greg; he writes them independently and uses them at work, but he does not tell anyone about them.
- Greg has written them in the evenings, at home, on his personal computer.

Situation IV: Ownership of Programs (part II)

- Greg decides to market these programming aids by himself. When Star's management hears of this, Cathy is instructed to tell Greg that he has no right to market these products since, when he was employed, he signed a form stating that all inventions become the property of the company.
- Cathy does not agree with this position because she knows that Greg has done this work on his own.
- She reluctantly tells Greg that he cannot market these products.
- She also asks Greg for a copy of the products.

Situation IV: Ownership of Programs (part III)

- Cathy quits working for Star and takes a supervisory position with Purple Computers, a competitor of Star.
- She takes with her a copy of Greg's products and distributes it to the people who work with her.
- These products are so successful that they substantially improve the effectiveness of her employees, and Cathy is praised by her management and receives a healthy bonus.
- Greg hears of this, and contacts Cathy, who contends that because the product was determined to belong to Star and because Star worked largely on government funding, the products were really in the public domain and therefore they belonged to no one in particular.

Situation V: Proprietary Resources

- Suzie owns a copy of G-Whiz, a proprietary software package she purchased legitimately. The software is copyrighted, and the documentation contains a license agreement that says that the software is for use by the purchaser only. Suzie invites Luis to look at the software to see if it will fit his needs. Luis goes to Suzie's computer and she demonstrates the software to him. He says he likes what he sees, but he would like to try it in a longer test.

Situation VI: Fraud

- Alicia works as a programmer in a corporation. Ed, her supervisor, tells her to write a program to allow people to post entries directly to the company's accounting files ("the books"). Alicia knows that ordinarily programs that affect the books involve several steps, all of which have to balance. Alicia realizes that with the new program, it will be possible for one person to make changes to crucial amounts, and there will be no way to trace who made these changes, with what justification, or when.
- Alicia raises these concerns to Ed, who tells her not to be concerned, that her job is simply to write the programs as he specifies. He says that he is aware of the potential misuse of these programs, but he justifies his request by noting that periodically a figure is mistakenly entered in the books and the company needs a way to correct the inaccurate figure.

Situation VII: Accuracy of information

- Emma is a researcher at an institute where Paul is a statistical programmer. Emma wrote a grant request to a cereal manufacturer to show the nutritional value of a new cereal, Raw Bits. The manufacturer funded Emma's study. Emma is not a statistician. She has brought all of her data to Paul to ask him to perform appropriate analyses and to print reports for her to send to the manufacturer. Unfortunately, the data Emma has collected seem to refute the claim that Raw Bits is nutritious, and, in fact, they may indicate that Raw Bits is harmful.
- Paul presents his analyses to Emma but also indicates that some other correlations could be performed that would cast Raw Bits in a more favorable light. Paul makes a facetious remark about his being able to use statistics to support either side of any issue.

Situation VIII: Ethics of hacking or cracking

- Goli is a computer security consultant; she enjoys the challenge of finding and fixing security vulnerabilities. Independently wealthy, she does not need to work, so she has ample spare time in which to test the security of systems.
- In her spare time, Goli does three things: First, she aggressively attacks commercial products for vulnerabilities. She is quite proud of the tools and approach she has developed, and she is quite successful at finding flaws. Second, she probes accessible systems on the Internet, and when she finds vulnerable sites, she contacts the owners to offer her services repairing the problems. Finally, she is a strong believer in high-quality pastry, and she plants small programs to slow performance in the websites of pastry shops that do not use enough butter in their pastries. Let us examine these three actions in order.

Summary

- Copyrights, patents, and trade secrets all have roles to play in providing legal protection for software
- Important legal intricacies determine relationships among employees, employers, software vendors, and customers
- Statutes in a variety of overlapping jurisdictions may determine what computer crimes are, how they are investigated, and how they may be enforced
- Unlike legal issues, ethical issues have both personal and philosophical elements and therefore often lack clear answers