LECTURE 5: Application Security

Dr. Esam A. Alwagait
alwagait@ksu.edu.sa

Module 3 Outline

- Programming errors with security implications
  - Buffer overflows
  - Incomplete access control
  - TOCTTOU errors
- Malicious code
  - Viruses
  - Worms
  - Trojan horses
- Program development controls against malicious code and vulnerabilities
  - Software engineering principles and practices
- Controls to protect against program flaws in execution
  - Developmental controls
  - Operational controls
  - Administrative controls

Why program security?

- Program security is a central piece of computer security?
- Programs constitute so much of a computing system
  - Operating systems
  - device drivers
  - networks
  - database management systems
  - Applications
  - even executable commands on web pages
- Key questions:
  - How do we keep programs free from flaws?
  - How do we protect computing resources against programs that contain flaws?

How to judge security of programs?

- Qualitative approaches
  - Quality of code
- Quantitative approaches
  - Quantity and types of errors, faults, and failures

Errors, faults, and failures (IEEE standard)

- Error: a mistake usually caused by a human such as mistake in performing an activity
  - Example: a designer may misunderstand a requirement
- Errors may lead to faults, an incorrect step that represents a problem behind the scenes (a potential problem!)
  - Example: when the designer misunderstanding lead to error in coding, requirements definition, etc.
- A failure is a departure from the system's required behavior
  - A deviation from intended behavior, not necessary the specification!
  - You log in Edugate and find yourself logging into other student’s account

Errors, faults, and failures (IEEE standard)

- Faults and failures are forms of program flaws
- A security flaw is a problem that affects security in:
  - Confidentiality, integrity, availability
- A fault is an inside view of the system, as seen by the eyes of the developers
  - i.e., programmer, designer, specifier
- A failure is an outside view
  - a problem that the user sees
- Not every fault corresponds to a failure
  - Example, if faulty code is never executed or a particular state is never entered, then the fault will never cause the code to fail
- Observe that there is some inconsistency with this terminology in the security community and IEEE standard

How to find and fix faults?

- How?
  - Analysis and testing
  - Working backward from user experience of a fault
  - What about faults that haven't yet led to failures?
- Fixing faults?
  - Early work relied on "penetrate and patch" paradigm (tiger team)
  - Sometimes led to rapid effort to patch the system!
  - Leading to produce new faults:
    - A narrow focus on the fault itself and not on its context (underlying design or requirements faults)
    - Nonobvious side effects in places other than the immediate area of the fault
    - Fixing one problem often caused a failure somewhere else
    - Improper fixes due to performance issues
- Other approaches?
  - Cause-effect analysis
    - Thus addressing both inside and outside views
  - Comparing requirement with the behavior

Is it possible to eliminate all program flaws?

- Security is fundamentally hard!
- Two issues
  - Program controls apply at the level of the individual program and programmer
    - We should consider both lists: Should do & Shouldn't do
      - How would you test "Shouldn't do" list?
    - It is almost impossible to ensure that a program does precisely what its designer or user intended, and nothing more, especially for large systems!
  - Programming and software engineering techniques change and evolve faster than computer security techniques
      - We often find ourselves trying to secure last year's technology while software developers are rapidly adopting today's and next year's technology
      - Security is in the catch-up mode!

Types of flaws

- Intentional flaws
  - Malicious flaws which are intentionally inserted for purpose of attacking/breaching a system
  - Nonmalicious flaws are usually features that part of the system, but can cause failures when exploited by an attacker
- Unintentional program errors, which are the source of most of security flaws

Unintentional program errors

- Common mistakes made by programmers and developers
- With the exception of a few classical types, many can cause malfunction, but do not lead to serious vulnerabilities
- Three classical types remain common among all development environments for decades:
  - Buffer overflow
  - Incomplete mediation
  - TOCTTOU errors

Buffer overflow

- The most commonly exploited flaw
- Happens when a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory
- Occurs when a process attempts to store data in random access memory (RAM) **beyond the boundaries** of a fixed-length storage buffer
- Extra data overflows into the adjacent memory locations and under certain conditions may cause the computer to stop functioning
- Attackers also use a buffer overflow in order to compromise a computer

Buffer Overflow

- Example: using C language

  char sample[10], k[2];

  sample[10] = 'B';

  strcpy(sample[0], k)

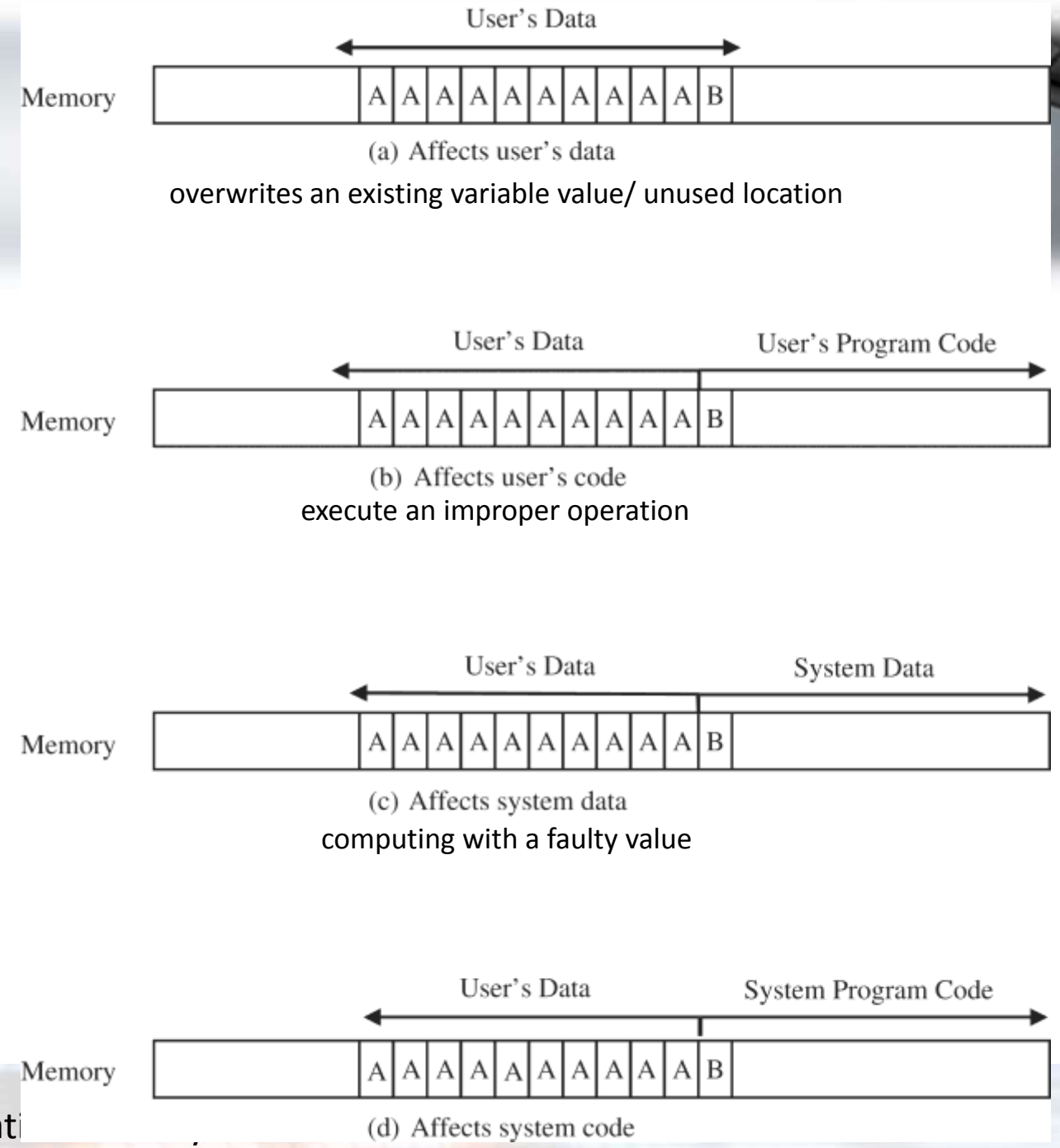
  - The subscript is out of bounds (that is, it does not fall between 0 and 9)
  - How about: sample[i] = 'B';
    - Is there a problem?

Buffer overflow

- It is important to recognize that the potential overflow causes a serious problem only in some instances
- The problem's occurrence depends on what is adjacent to the array

- Examine the following

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



Security Implication

- The attacker may replace code in the system space
  - Observe that operating system runs with higher privileges than those of a regular program
  - the attacker can execute many commands in a powerful role
- The attacker may use of the stack pointer or the return register
  - the attacker can change either the old stack pointer (changing the context for the calling procedure) or the return address (causing control to transfer where the attacker wants when the subprocedure returns)
- Or, when passing parameter values into a routine
  - when the parameters are passed to a web server on the Internet
  - http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212 &parm2=2009Jan17

Buffer Overflow Protection

- Basic defenses
  - Write “defensive” program code that will protect against these attacks
  - Use a programming language that makes these attacks more difficult
- For Windows-based systems, there are two defenses against buffer overflows
  - Data execution prevention (DEP)
  - Address space layout randomization (ASLR)

Buffer Overflow Protection (continued)

- **Data Execution Prevention (DEP)**
  - Most modern CPUs support an **NX (No eXecute)** bit to designate a part of memory for containing only data
  - DEP will not allow code in the memory area to be executed
  - Windows Vista allows software developers to enable NX hardware protection specifically for the application software that they develop

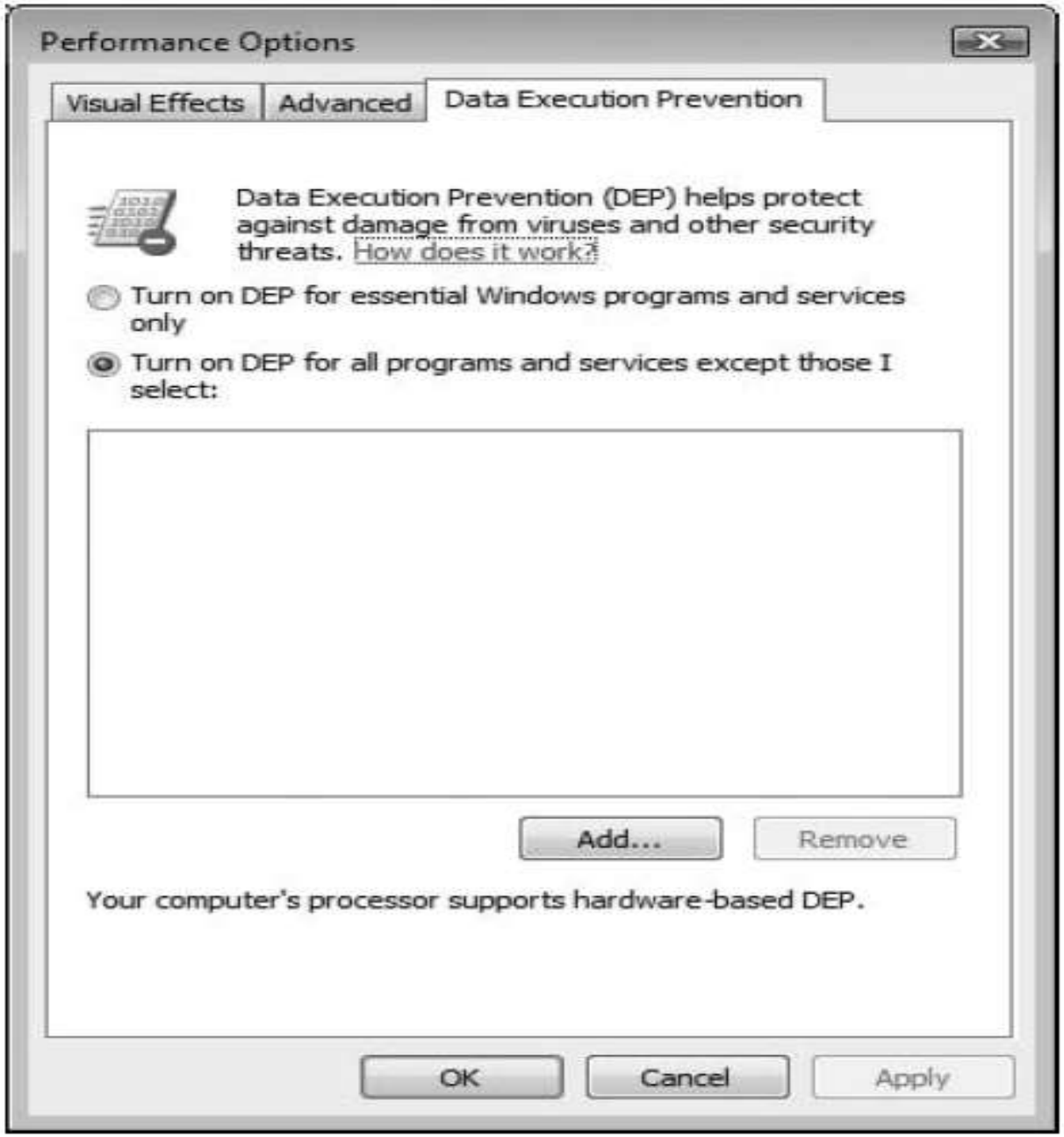


Figure 3-4 DEP options

Buffer Overflow Protection (continued)

- **Address Space Layout Randomization (ASLR)**
  - Randomly assigns executable operating system code to one of 256 possible locations in memory (ASRL moves the function entry point. The chance is 1/256)
  - This makes it harder for an attacker to locate and take advantage of any functionality inside these executables
  - ASLR is most effective when it is used in conjunction with DEP

Incomplete mediation

- Common in web-based applications (thru web browsers)
- Occurs when application accepts incorrect data input
- Example: passing parameters to a routine through web browser

http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212 &parm2=2009Jan17

- What happens if param2 was submitted with
  - 1800Jan01
  - 2000Feb32
  - 3048Min30
  - 1Aardvark2Many
  - 98776754543434232
  - DROP DATABASE clients;
- The web application (the mediation) needs to ensure correct inputs
  - Common example: Javascript code to do validation checks
    - Is this a good practice?

Security Implication

- A real example:

http://www.things.com/order.asp?custID=101&part=555A&qy=20&price=10&ship=boat&shipcost=5&total=205

- A malicious attacker supplied instead the following URL
- http://www.things.com/order.asp?custID=101&part=555A&qy=20&price=1&ship=boat&shipcost=5&total=25
- Thus exploited this flaw!
- Issues:
  - Besides lost profit
  - The length of time such a flaw remained undetected!
  - Other issues?

Security Implication

Client-side mediation

- Common scenario
  - Website performs client-side input validation check using Javascipt code
    - If incorrect data entered, a popup window will appear and prevent form submission
- What if the attacker:
  - Turns off Javascript?
  - Edits the form before submission?
  - Writes a script that interacts with the web server instead of using a web browser at all?
  - Connects to the server
    - telnet server.com:80
- How to fix this?
  - Client-side alone leaves a security flaw
  - Implement server-side mediation

Time-of-Check to Time-of-Use Errors (TOCTTOU)

- A programming flaw that involves synchronization
- Sometimes called serialization or synchronization flaw
- Common in access controls
- Typical scenario:
  - The user requests the system to perform an action
  - The systems verifies the system is allowed to perform the action
  - The user changes object
  - The system performs the operation
  - So, what was checked is no longer valid when the object is accessed

Security Implication

TOCTTOU errors

- Example1: change contents of a file
- Example2: Unix terminal program setuid
  - Uses superuser privileges
  - Supports a command to dump terminal contents into a file
  - Checking: logfile -> my-file
  - Execution: logfile -> /etc/passwd
- The problem
  - The state of the system changed between time of checking and time of executing the operation
- How to prevent this?
  - Copy data from the user's space to the routine's area and perform validation checks on the copy
  - Seal the request data with a checksum to detect modification
  - Ensure that critical parameters are not exposed during any loss of control

Legit request

my_file	change byte 4 to "A"

Modified file descriptor after access approval

your_file	delete file

Malware

- First, much of the work done by a program (any program) is invisible to users who are not likely to be aware of any malicious activity
- Malware exists in various software forms with malicious intent
- Malicious code behaves in unexpected, harmful ways
- All malware types need to be executed in order to cause harm
- How are these malware types get executed?
  - User action
    - Downloading and running malicious software
    - Viewing a web page containing malicious ActiveX control or Java applet code
    - Opening an executable email attachment (sometimes hidden!)
    - Inserting a CD, flash disk, etc.
    - Infected SETUP program, etc.
  - Exploiting an existing flaw in a system
    - Buffer overflow
      - Hosts platforms
      - network agent, daemons, etc.
      - Applications, such as email clients, web browsers, etc.
      - Device drivers

What can malicious code do?

- Malicious code can do anything any other program can do, but without the user's permission or even knowledge:
  - Displaying a message on a computer screen
  - Stopping/halting a running program, or running a program
  - Making a sound
  - Deleting/copying files
  - Or can be planted to remain undetected, until some event triggers the code to launch
    - The trigger can be a time or date
    - an interval, for example, after 30 minutes
    - an event, for example, when a particular program is executed
    - a condition, for example, when communication occurs on a network interface
    - a count, for example, the nth time something occurs
    - some combination of these
    - a random situation
      - doing different things each time, or nothing most of the time with something dramatic on a particular occasion

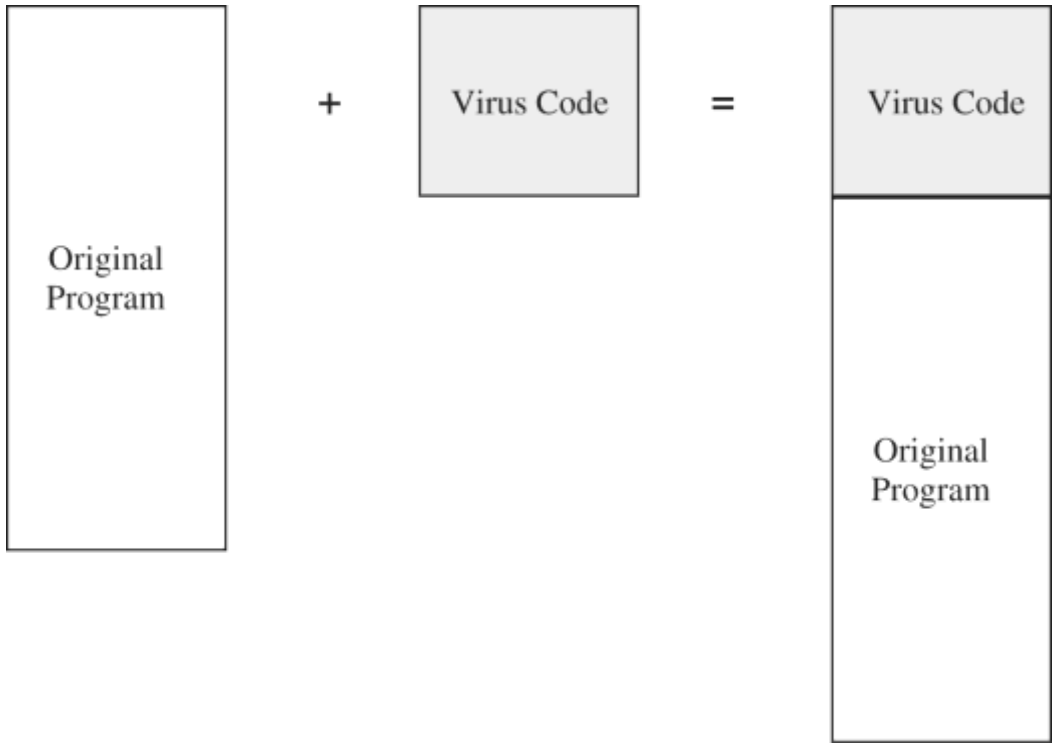
Viruses

- A **virus** is a program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them
  - Traditionally, infect executable files
  - But now can infect many data files because of they contain executable code (e.g., macros, system calls)
  - Document viruses are currently the mot popular type
    - Implemented within a formatted document, such as a document, spreadsheet, slides, picture, etc.
    - structured files contain both data (words or numbers) and commands (such as formulas, formatting controls, links)
  - Viruses can spread between files or computers
- Transient viruses
  - Its life depends on its host program
- Resident viruses
  - Executes as a standalone code in memory

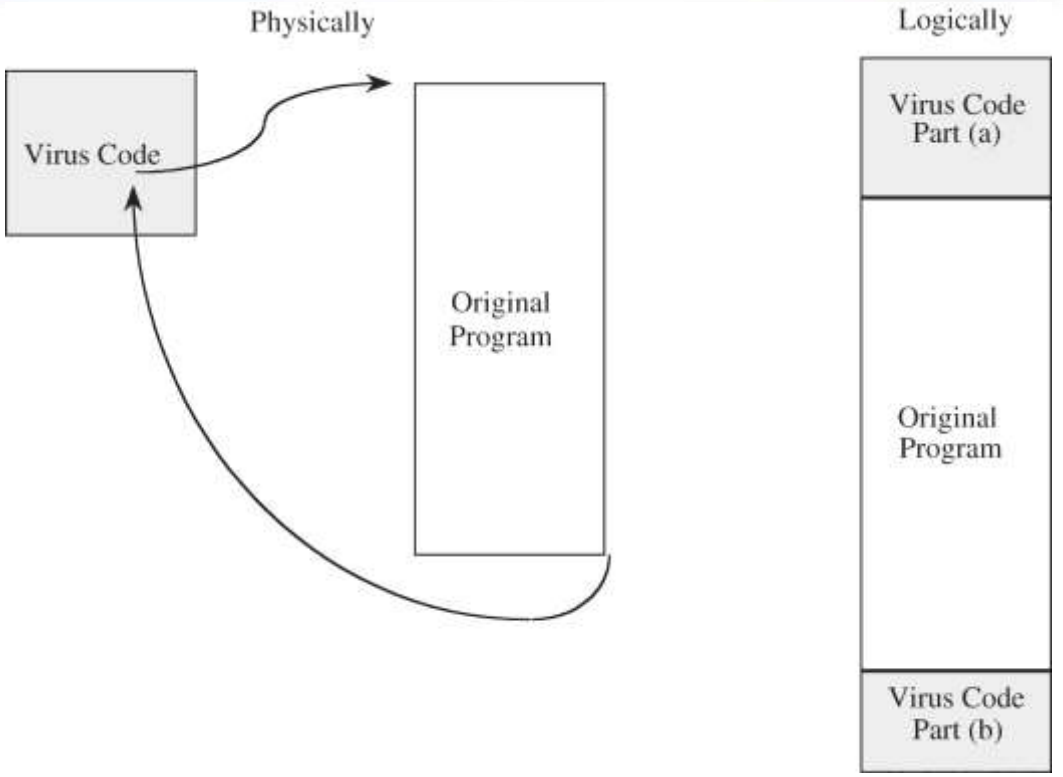
Viruses exist in many forms: Trojan horse, logic bombs, and worms

- A **Trojan horse** is malicious code that, in addition to its primary effect, has a second, nonobvious malicious effect
- Example?
  - A login script designed for user's identification and password, which does normal logging processing plus saving a copy of the information for malicious purposes
- A **logic bomb** is a malicious code that activates when a specified condition occurs
  - A **time bomb** is a logic bomb whose trigger is a time or date
  - Example popular dates: Friday the 13th and April Fool's Day
- A **worm** is a program that spreads copies of itself through a network
- Example?
  - Code Red I worm in 2001, targeted MS IIS web server, for which a patch had been released a month earlier!
  - Exploited a buffer overflow vulnerability using a long string of the repeated character 'N' to overflow a buffer
  - Installed a backdoor and a Trojan horse to prevent disinfection!
  - Infected about 250,000 machines in 9 hours, hit a total of 750,000 servers
  - Estimated damage $2 billion!
- Furthermore, two or more forms of malicious code can be combined to produce a third kind of problem
  - Example: a virus can be a time bomb if the viral code that is spreading will trigger an event after a period of time has passed

How viruses attach?

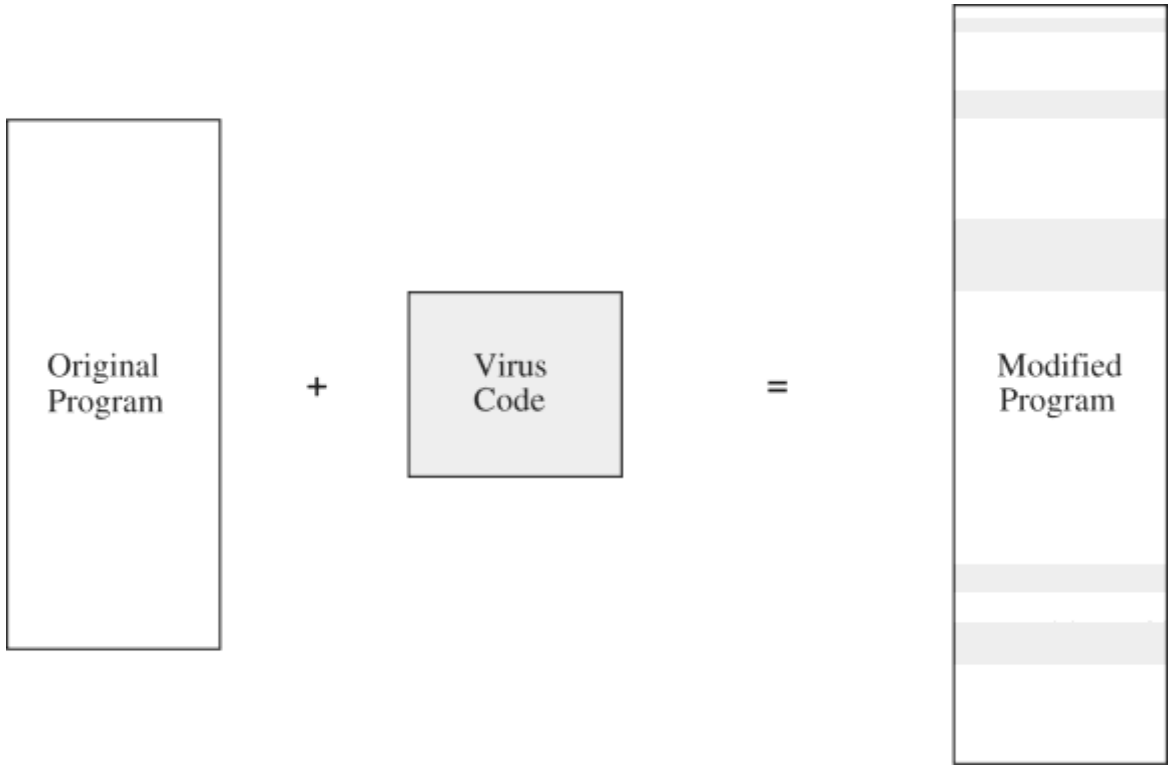


Virus Appended to a Program



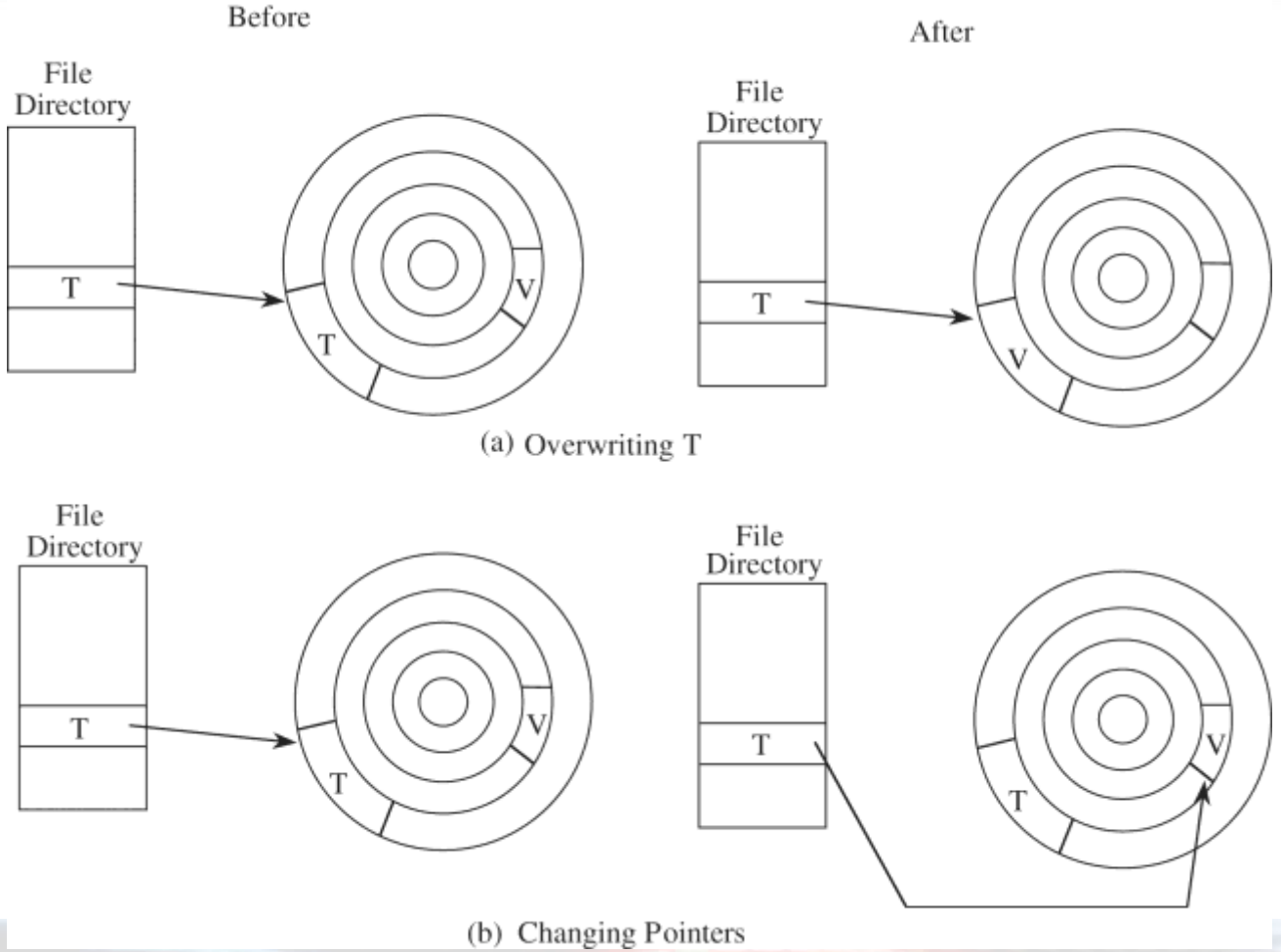
Virus Surrounding a Program

How viruses attach?

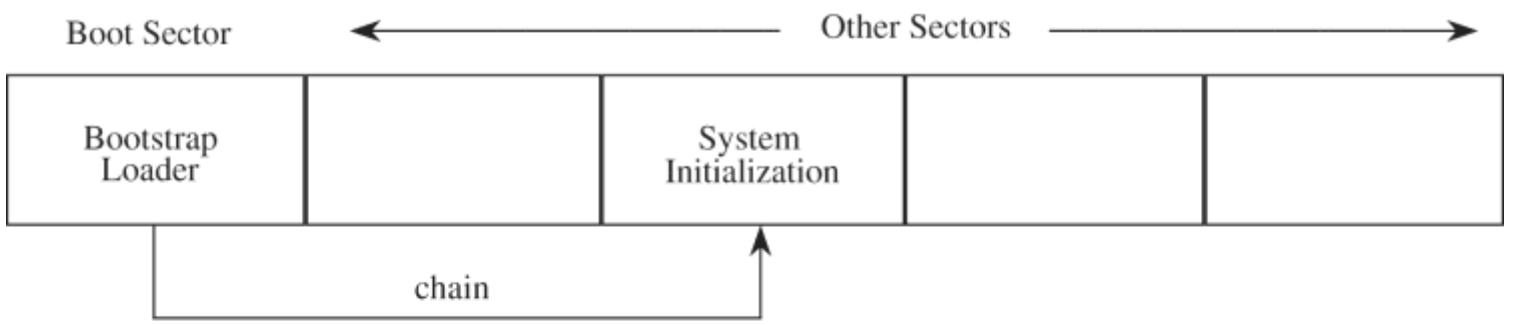


Virus Integrated into a Program

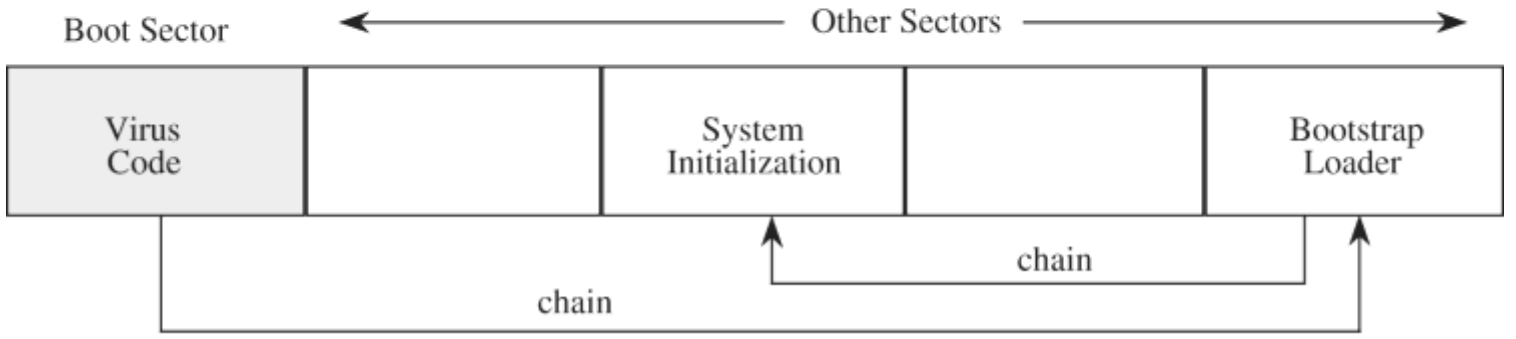
Gaining control by completely replacing a program



Boot Sector Virus Relocating Code



(a) Before infection



(b) After infection

Virus Signatures

- A virus cannot be completely invisible
- Code must be stored somewhere, and the code must be in memory to execute
  - Sometimes with storage particular patterns, e.g., 400 bytes from the top of the target file
- The virus executes in a particular way, using certain methods to spread
- Transmission patterns
- Polymorphic viruses
  - Changing the appearance of a virus (usually by manipulating constant and random codes)
  - Sometimes viruses change into two forms or even more!
- These characteristics builds up a pattern, called a signature, that identifies the virus
- Virus scanners use such features to scan and remove viruses
- Scanners searches memory and storage, and monitor execution code
- For example, a scanner looking for signs of the Code Red worm can look for a pattern containing the following characters:
  - /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN %u9090%u6858%ucbd3

Detecting viruses

used in virus, intrusion, and threat scanners

- Signature-based technique
  - A list of known viruses and their patterns are known and stored
    - Features such as infection code, location, etc.
  - Major limitation: only scans what exists in the list of patterns
- Behavior-based technique
  - A technique to detect polymorphic viruses
  - Behavior-based systems look for suspicious, irregularly patterns of behavior, not a specific code fragments
  - Major limitation: useful for post-infection

Scanner errors: false negatives and positives

- False negatives (type I errors)
  - a test result indicates that a condition failed, while it actually was successful
  - failing to identify a threat that is present
  - classifying *imposters as authorized users*
- False positives (type II errors)
  - a test result that indicates a given condition has been fulfilled, when it actually has not been fulfilled
  - claiming a threat is present when it is not
  - Classifying *authorized users as imposters*
- Which is worse?

Zero-day attacks

- Time between knowledge of vulnerability and security countermeasure (control) is critical
- Zero-day attacks occur when attacks launches before the availability of its control
- Worse than this is when attackers target vulnerabilities unknown to security community/system owners

Several techniques for reasonably safe electronic environment

- Use only commercial software acquired from reliable, well-established vendors
- Test all new software on an isolated computer
- Open attachments only when you know them to be safe
- Make a recoverable system image and store it safely
- Make and retain backup copies of executable system files
- Use updated virus scanners

Truths and Misconceptions About Viruses

- Viruses can infect only Microsoft Windows systems. **False**
- Viruses can modify "hidden" or "read-only" files. **True**
- Viruses can appear only in data files, or only in Word documents, or only in programs. **False**
- Viruses spread only on disks or only through e-mail. **False**
- Viruses cannot remain in memory after a complete power off/power on reboot. **True**, but…
- Viruses cannot infect hardware. **True**

Rootkits

- A piece of malicious code that goes to great lengths not to be discovered or, if discovered and removed, to reestablish itself whenever possible
- It is a tool often used by “script kiddies”
- Typically, it interferes with the normal interaction between a user and the operating system
- Rootkits operate with two main techniques:
  - A method to operate as root,
    - E.g., superprivileged user of a Unix system
  - A way to hide itself
    - Sometimes called “stealth” capabilities, how?
      - Clean up any log messages created by the code
      - Modify commands like ls and ps so that they don't report rootkits files and processes
      - Modify the kernel so that such files and processes become concealed to the user programs

Rootkits example: Sony XCP

- Discovered in 2005 when security expert named Mark Russinovich was developing a rootkit scanner (revealer) for Windows
  - When testing his rootkit revealer, he discovered his machine already had one!
- The source of the rootkit turned out to be Sony audio CDs equipped with XCP for copy protection
- When you insert such an audio CD into your computer, it contains an autorun.exe file that automatically executes autorun.exe to install the rootkit
- The main purpose of the rootkit was to prevent copying music contents
  - modify the CD driver in Windows so that any process that tried to read from an XCP protected CD would get garbled output
- The secondary purpose was to hide itself and make it hard to find and uninstall
  - Hid all rootkit files and processes started with sys
  - So, any virus writer could conceal a virus just by naming it sysvirus…!
- Sony worked with major antivirus vendors so its rootkit would remain undetected
  - “because keeping the user uninformed was better for all of them”!
- After so many complaints, Sony eventually released an uninstaller
  - But again, running the uninstaller left a back door on your system! Opening up another serious vulnerabilities
  - Remember “"penetrate and patch” strategy was not a good security practice!
- How many computers were infected by this rootkit?

Privilege Escalation

- Main concepts
  - Programs normally run in a context, as most systems run with the concept of multiple levels of privileges for accessing objects
    - Examples: Windows and Unix user, social apps, web sites, etc.
  - Malicious code writers want to access objects outside privileged context
- A privilege escalation is an attack that raises the privilege level of the attacker by malicious code launched by a user with lower privileges
- This process tricks the system, which legitimately runs with higher privilege, into executing commands (with that higher privilege) on behalf of the attacker

Keystroke Logging

- A malicious code that retains a secretive copy of all keys pressed
- Remember!
  - Most of the information flow from you (the user) to your computer (or beyond, to the Internet) is via the keyboard
  - A little flow from the mouse/screen and devices like USB keys
- Often exists in two types
  - Independent: retaining a log of every key pressed
  - Linked to a certain program, retaining data only when a particular program, such as when a banking application runs
- An attacker might keep a record of emails, IM, passwords, visited web sites, etc.
  - However, sometimes installed by family member to spy on/monitor children activities as example
- This data can be accessed locally or sent to another machine, which could be located over the Internet

Types of keyboard loggers

- Application-specific loggers:
  - Only logs/record keystrokes pertinent to a particular application, such as an IM client
- System-specific loggers:
  - Record all keystrokes that are pressed (maybe only for one particular target user)
- Hardware keyboard loggers:
  - A small piece of hardware that sits between the keyboard and the computer
  - Works with any OS
  - Undetectable in software

Phishing

- Phishing is the act of attempting to acquire information such as usernames, passwords, and credit card details (and sometimes, indirectly, money) by masquerading as a trustworthy entity in an electronic communication
- Communications claiming to be from popular sources are commonly used to trap unsuspecting public, such as
  - social web sites, auction sites, banks, online payment processors or IT administrators
- Phishing is an example of **interface illusions**
- It looks like you're visiting Paypal's or Samba website, but you're really not
  - If you type in your password, you've just passed it to an attacker
- Advanced phishers can make websites that look every bit like the real thing
  - Even if you carefully check the address bar, or even the SSL certificate

Phishing

- Early phishing targeted AOL users
  - A phisher might pose as an AOL staff member and send an instant message to a potential victim, asking him to reveal his password
  - The message might appear as "verify your account" or "confirm billing information“
- Later, phishers targeted financial institutions
  - The first known direct attempt against a payment system affected E-gold in June 2001, which was followed up by a "post-9/11 id check" shortly after the September 11
- Some phishing techniques
  - Spear phishing: targets specific individuals or companies
  - Clone phishing: creating a cloned email of a legitimate one and replacing its contents with a malicious version, and then sent from a spoofed address of original sender
  - Walling: attacks directed at senior executives and high profile targets
  - Website forgery: creating phishing websites

The Nature of Software Development

- System specification
- Design
- Implementation
- Testing
- Code review
- Documentation
- Management
- Maintenance
- And remember!
  - we must design systems that are both secure and usable

Developmental controls

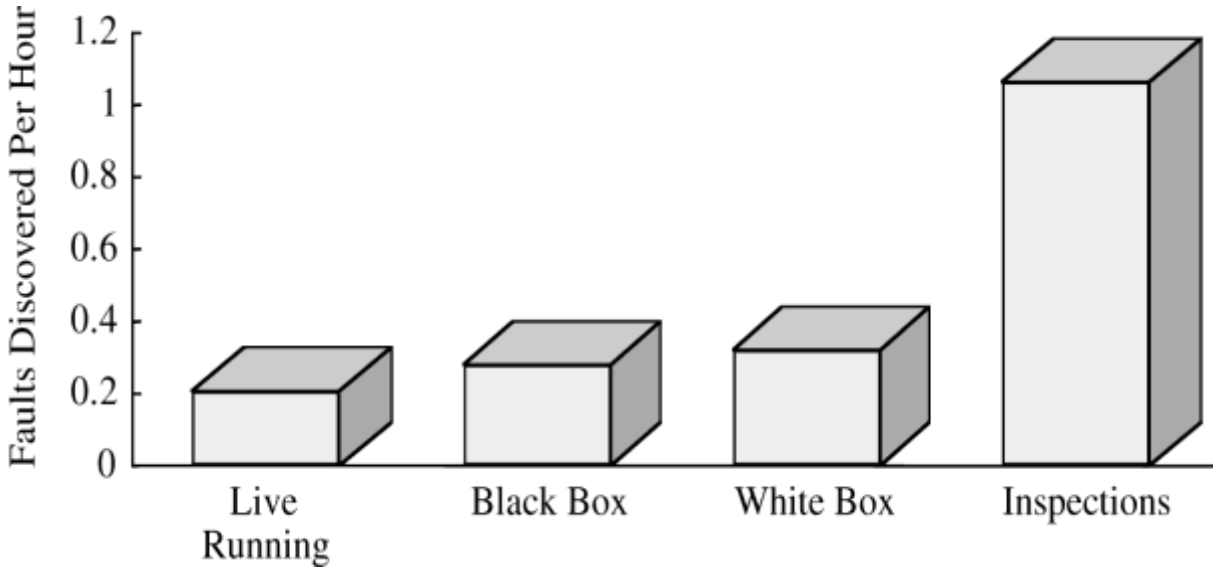
- Code usually has a long shelf-life and is enhanced over time as needs change and faults are found and fixed
- How to **design** secure programs (less likely to have security flaws)?
  - Modularity
  - Encapsulation
  - Information hiding

Modularity

- Design (or code) in small, self-contained units
  - Each performs a single task (single purpose)
  - Small
  - Simple
  - Independent
- Advantages
  - Maintenance
  - Understandability
  - Resue
  - Correctness
  - Testing
  - Easier to check for program flaws, test, enhance, etc.

Administrative controls

- Peer reviews
- Hazard analysis
  - What-if scenarios
  - A system issue, not just a code issue
  - e.g., Fault Trees Analysis
- Testing (both black-box and white-box)
  - Test the behavior the program gets right
  - Test the behavior that makes the program go wrong as well!
  - Unit, integration, function, performance, acceptance, installation, regression testing
  - Penetration testing (unique to computer security)
    - simulates an attack by a malicious party (attack attempts)
    - is an attack on a computer system with the intention of finding security weaknesses, potentially gaining access to it, its functionality and data
  - Vulnerability Assessment (unique to computer security)
    - the process of identifying, quantifying, and prioritizing (or ranking) the vulnerabilities in a system



Fault Discovery Rate Reported at Hewlett-Packard.

Administrative controls– continued

- Configuration management
  - building and documenting an inventory of all components that comprise the system
  - Documenting who is making which changes to what and when
- Proofs of Program Correctness