Sustain desirable properties under intelligent adversaries

Desirable properties - understand what properties are needed

Intelligent adversaries - understand/model adversaries always think about adversaries

Confidentiality

Concealment of information or resources: only those who are authorized to know can know

Secrecy, privacy

Mere existence of data can be revealing!

Resource hiding

Existence of specialized equipment

Location of equipment, e.g. data centers

Integrity

* **Trustworthiness of data or resources**
  + Data integrity: content of the information
  + Origin integrity: source of data, often called authenticity
* Ensure we block
  + Unauthorized attempts to change the data
  + Attempts to change the data in unauthorized ways
* Attempts either malicious or accidental

Availability

* **The ability to use the information or resource desired when required**.

Which of C, I, A were violated?

Confidentiality

* + Credit card digits

Integrity

* + Passwords

Availability

* + Data and devices

What Means do we Use to Support Security?

**Prevention**

* Encryption
  + Encode message so that only authorized parties can access its content
* Authentication
  + Check who is accessing
* Access control
  + Ensure only authorized access is allowed

**Detection**

* Auditing
  + Record action to identify attacks and recover

Unpredictability in Thread Execution

* Thread execution may be interrupted
  + “Time slicing” of threads (and processes) prevents one thread from “hogging” the CPU
  + Higher priority activities may interrupt the thread: e.g., I/O
* Multiple threads do not always proceed at the same rate
* Coordination of multiple threads a challenge
* Java provides low-level and high-level tools to deal with *synchronization of threads*

Thread Safety: a Definition

A class is thread-safe if it behaves *correctly* when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime environment, and with *no additional synchronization* or other coordination on the part of the calling code.

*Java Concurrency in Practice, by Brian Goetz, 2006*

A class is thread-safe if it behaves *correctly* when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime environment, and with *no additional synchronization* or other coordination on the part of the calling code.

*Java Concurrency in Practice, by Brian Goetz, 2006*

What is the final value of counter?

* Two threads
* Each incrementing the value of counter:

for (int i = 0; i < 10000; i++) {

counter++;

}

* Starting from 0, what is the value of counter after both threads have finished executing their respective loops?

Compiling to Byte Code

* Compiler “flattens” the Java code into a linear sequence of instructions that can be executed by the JVM
* Control flow uses “goto” statements
* if and for statements use “conditional gotos”
* Uses a stack to evaluate expressions (like an RPN calculator)

The Fetch-Execute Cycle

* (A simplified view…)
* The JVM performs a “fetch-execute” cycle…
  + Fetch the instruction from the “current location” (the program counter) in memory
  + Execute the instruction
  + Update the “current location” (either to next location in memory or as a side effect of executing the current instruction, e.g., a jump instruction)
  + Repeat
* Think of the PC as an “arrow” pointing to the instruction being executed

Sequential vs. Concurrent

* Sequential:
  + A single “thread of execution” weaves its way through your program
  + A single PC (“program counter”) identifies the current instruction being executed
* Concurrent:
  + Multiple “threads of execution” are running simultaneously through your program
  + Multiple PCs are active, one for each thread

Synchronization Problem:   
Race Condition

* As threads “race” through execution, their instructions are interleaved at the nanosecond level
  + Byte codes within a thread always executed in relative order, as expected
  + Byte codes between threads not executed in predictable absolute order
* Causes problems when accessing and updating shared data

Java Threads

* Thread class with run() method
* Allows creation and manipulation of threads
  + Thread t = new Thread();
* Three important methods:
  + t.start(): start the thread referenced by t
  + t.join(): “join with” (wait for) the running thread t
  + t.run(): called by start() in a different thread
* Note: Your code *does not* call run() directly; instead, the start() method calls run() as part of the new thread sequence

Two Ways to Create Threads

* Create a subclass of Thread and override run()

public class MyThread extends Thread {

public void run() { … }

}

Thread t = new MyThread();

* Create a class that implements the Runnable interface and pass an object of it to Thread()

public class MyTask implements Runnable {

public void run() { … }

}

Thread t = new Thread(new MyTask());

Runnable vs. Thread Subclass

* Preferred: Use Runnable
* Problem with subclassing: Not really “specializing” Thread class, just providing a run() method
* So, “class composition” is the preferred way to think about this problem:
  + “Thread has-a Runnable”
* Another benefit: Allows the class that implements Runnable to be part of a different class hierarchy than Thread

Join: Wait for a Thread to Finish

* A simple kind of synchronization
* For Thread t:

t.join();

* Blocks the “current thread”—the one that called t.join()—until Thread t completes (returns from run())
* join() may throw an InterruptedException, so generally is in try-catch clause

Solution: Synchronize Threads

* Java keyword “synchronized”
* Syntax:

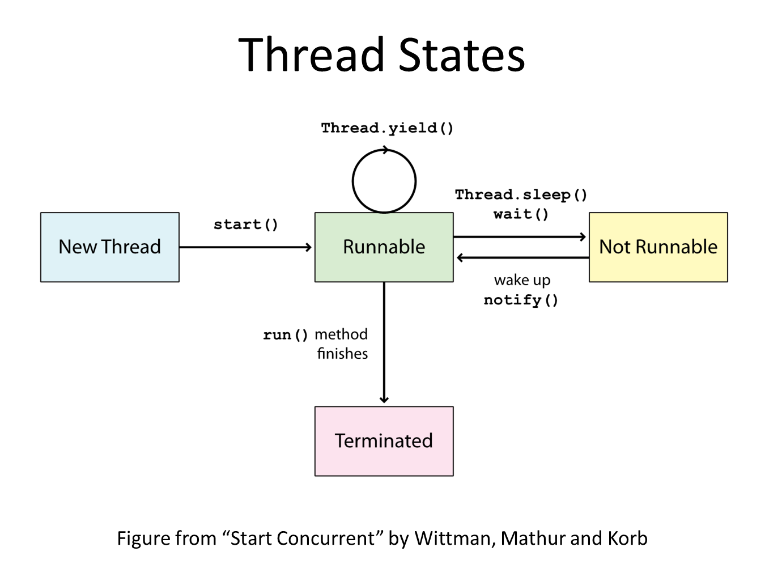
synchronized (object) {

statements; // modify shared data here

}

* Allows two or more threads to use a common object to avoid race conditions
* Among all threads synchronizing using the same object, only one thread can be “inside” the block of statements at a time

Thread States



Impersonation

* Email id well known (gov.palin) from previous news story
* Forgotten password security questions: answers were public knowledge or easily deducted
  + Birth date
  + Postcode
  + Where she met her husband

Password reset to popcorn

Some Definitions

**Identification:** asserting who a person is

* + Identities are often well known and public

**Authentication:** proving the asserted identity

* + Protected process

**Authorization:** associating privileges with an identity

Subjects

Replace person with subject:

* person
* computer process
* network connection
* … any similar active entity

Subjects

Replace person with subject:

* person
* computer process
* network connection
* … any similar active entity

Authentication

**… is the binding of an identity to a subject.**

* What the entity knows
  + Password, pin number, mother’s maiden name
* What the entity has
  + Bagde, card, driver’s license
* What the entity is
  + Biometrics (fingerprints, retinal characteristics, voice pattern)
* Where the entity is
  + In front of a particular terminal

Threats/Attacks to Passwords

* Eavesdropping
  + insecure channel between client and server
* Shoulder surfing, keyloggers
* Login spoofing
  + Access to the victim’s system (as opposed to phishing attacks)
* Social engineering
  + e.g., pretexting: creating and using an invented scenario (the pretext) to persuade a target to release information or perform an action (usually done over the telephone)
* Online guessing (weak passwords)
* Offline dictionary attacks
  + Guessing of a password by repeated trial and error

Examples of Weak Passwords

* Default passwords
* Dictionary words
* Words with numbers appended
* Words with simple obfuscation (*g0ldf1sh*)
* Doubled words (stopstop)
* Common sequences from a keyboard row (fred)
* Well known number sequences (911)
* Anything personally related to an individual
  + Licence plate number, SSN, birthday, address, sports team etc.

Guessing Attacks:   
Two Factors for   
Password Strength

* The average number of guesses the attacker must make to find the correct password.
  + Determined by how unpredictable the password is, including how long the password is, what set of symbols it is drawn from, and how it is created.
* The ease with which an attacker can check the validity of a guessed password.
  + Determined by how the password is stored, how the checking is done, and any limitation on trying passwords.

Mechanisms to Defend Against Dictionary and Guessing Attacks

* Protect stored passwords (use both cryptography & access control)
* Disable accounts with multiple failed attempts
* Require extra authentication mechanism (e.g., phone, other email account, etc.)

Storing Passwords

Store them cleartext?

Use hashing. A **one way hash function**:

* computes a fixed-size string (hash)
* it is computationally infeasible to find two different messages x and y such that H(x) = H(y)

In other words, we cannot (easily) detect the original string from the encoded string.

Examples: MD5, SHA family

Storing Passwords   
(UNIX Case Study)

In Old UNIX

* The file /etc/passwd stores H(password) together with each user’s login name, user id, home directory, login shell, etc.
* The file /etc/passwd must be world readable
  + many applications and system tools require that information to function properly

Dictionary Attack: get hold of the encrypted hash and try different words to produce the exact same hash

Shadow Files

More modern UNIX:

Restrict access to the hash values

Divide /etc/passwd into two files:

* /etc/passwd
* /etc/shadow (readable only by root)

Password Salts

Store [r, H(password,r)] rather than H(password) in /etc/shadow

* r is randomly chosen for each password
* r is public

Benefits?

* dictionary attacks much more difficult
  + the cost of attacking a single account remains the same
* if two users happen to choose the same password, it doesn’t immediately show

Biometrics Pros

Difficult if not impossible to lose, steal, forget, lend, or forge.

Always available, at hand.

Biometrics Cons

* May be hard to obtain
* Some people find their use intrusive
* Recognition devices are costly
* Devices can become a single point of failure
  + can have multiple credit cards, not fingerprints
* False readings still occur
* Tradeoff between accuracy (e.g. multiple readings) and speed
* Forgeries are still possible (“gummy fingers”)

Not Reliable for Identification

* During registration, the biometrics reader produces a “template”.
* During authentication, the device compares the new measurement with the stored template.
* Works well, if it is unlikely an imposter will have the same biometric template as the real user.
* Unless every template is unique, the system cannot uniquely identify subjects!

Two-Factor Authentication

Examples

* Reset an email password after clicking a link sent to an alternative email address AND typing in a number sent to your cellphone.
* Enter a pin AND a dynamic value (a value that depends on the time instance)

RSA SecurID

* Dynamic token, two factor authentication
* Enter a pin and the number displayed on the screen’s device
* The server also has a real-time clock and a database of valid cards with the associated seed records.
* Authenticates a user by computing what number the token is supposed to be showing at that moment in time

PRIVACY

* As it Relates to Authentication
* What is Privacy?
* Closely related to notions of identity and authenticity
* Who has done what to what?
* Often involves *not* being able to associate a particular person with an action.

Anonymity

* Reduces fears of discrimination (e.g. housing, employment)
* Researching a private matter (e.g. health issues)

BUT…

* How can one pay anonymously for a service online?
  + Trusted third party
* Multiple Identities
* Different bank account numbers, credit cards, etc.
* Challenge of linking identities (e.g. based on name, based on house address at the same time period)
* Linking identities *correctly* may break anonymity and create privacy leaks
* Linking identities *incorrectly* may create much more serious risks

Pseudonymity

* Unique identifiers that can be used to link records in a server’s database but that cannot be used to trace back to a real identity (*pseudonyms*).
* Disposable identities/temporary email addresses

USABILITY

* As it Relates to Authentication

Password Guidelines

* Usability and security sometimes seem at odds. For instance:
  + Four rules (“must include… ”)
  + Two exclusions (“do not use…”)
  + Four qualitative judgements (“difficult to guess”, “ease to remember”…)
* What if password expiration date passes without resetting? These could work against security:
  + At the first access attempt, the user forced to changed password
  + Sending increasingly urgent warnings as the expiration approaches
* The practice of masking a password (asterisks while you type it)
  + Considered by some obsolete practice, hurts usability

Single Sign-On

* The user authenticates once per session
* The system forwards the authenticated identity to all other processes
* Easier for the user, but several security concerns
  + The authentication strength everywhere is as good as the strength of the initial authentication
  + Compromise one and you have access everywhere!
  + Microsoft passport: a folder for login credentials for all sites
  + …
* An alternative: a utility to generate and store passwords for web sites and applications (e.g. browser add-ons). Depends on master password

Program Flaws   
Security Implications

* Integrity: incorrect output or action
* Opportunity for Exploitation

Vulnerability: Incomplete Mediation

* Verifying that the subject is authorized to perform the operation on an object is called **mediation.**

Access control triple: what subject can perform what operation on what object

Vulnerability: Race Condition

In a **race condition** or **serialization flaw** two processes execute concurrently, and the outcome of the computation depends on the order in which instructions of the processes execute. Can cause inconsistent outcomes (failure of integrity)

Vulnerability: Time-of-Check to Time-of-Use (TOCTTOU)

* A sculpture costs $100.
* The buyer carefully counts five $20 bills in front of the seller and lays them on the table.
* The seller turns around to write a receipt.
* While the seller’s back is turned, the buyer takes back one $20 bill.
* The access control mediator receives the data structure
* Determines whether the access should be allowed
* If access is allowed, it forwards the data structure to the file handler for processing.

Security Implication

Ineffective access control

* confidentiality failure
* integrity failure
* or both

Vulnerability: Undocumented Access Point

The programmer creates an undocumented entry point or execution mode (e.g. a special debug mode to test conditions)

Such an access point is called a **backdoor** or **trapdoor**.

Ineffective Countermeasure: Penetrate-and-Patch

* The pressure to repair a specific problem encourages developers to take a narrow focus on the fault itself and not on its context. The analysts often pay attention to the immediate cause of the failure and not to the underlying design or requirements faults.

Ineffective Countermeasure: Penetrate-and-Patch

* The pressure to repair a specific problem encourages developers to take a narrow focus on the fault itself and not on its context. The analysts often pay attention to the immediate cause of the failure and not to the underlying design or requirements faults.
* The patch may have addressed the problem in only one place, not in other related places.
* The fault cannot be fixed properly because system functionality or performance would suffer as a consequence. Only some instances of the fault may be fixed; the damage may be reduced but not prevented.

**Countermeasure: Identifying and Classifying Faults**

* Examine programs to see whether they behave as their designers intended or users expected
* Program security flaws can derive from any kind of software fault
  + misunderstanding of program requirements to a one-character error in coding or even typing

Categories of Program Flaws

* inadvertent human errors
* malicious, intentionally induced flaws

*“It doesn’t matter whether the stone hits the pitcher or the pitcher hits the stone, it’s going to be bad for the pitcher.”*

*Sancho Panza in Man of La Mancha*

A system attack often exploits an unintentional security flaw to perform intentional damage.

Security *is* fundamentally hard

* Often conflicts with usefulness and performance
* We use a “should do” checklist to test a system/program. Security is also about a “shouldn’t do” list (almost impossible to ensure it).
* Software engineering evolves far more rapidly than do computer security techniques.

Types of Inadvertent Flaws

* *validation* error (incomplete or inconsistent): permission checks
* *domain* error: controlled access to data
* *serialization* and *aliasing*: program flow order
* inadequate *identification* and *authentication*: basis for authorization
* *boundary condition* violation: failure on first or last case
* other exploitable *logic errors*

**Countermeasure: Secure Software Design Elements**

* Modularity
* Encapsulation
* Information Hiding
* Mutual Suspicion
* Confinement
* Simplicity
* Genetic Diversity
* Design Principles for Security

Modularity: Four Conditions for Each Component

* *single-purpose*: performs one function
* *small*: a human can readily grasp both structure and content
* *simple*: a human can readily understand the purpose and structure of the module
* *independent*: performs a task isolated from other modules

Why?

* *Maintenance:* can be replaced easily with a revised one if necessary
* *Understandability*
* *Reuse*
* *Correctness:* quickly trace failure to its cause since components perform only one task each
* *Testing*

A Modular Component has…

high cohesion and low coupling

* **Cohesion**: all the elements of a component have a logical and functional reason for being there
* **Coupling**: the degree with which a component depends on other components in the system

Encapsulation

* Hides a component’s implementation details
* Sharing is minimized
* Sharing is carefully documented so that a component is affected only in known ways by other components

Mutual Suspicion

Mutually suspicious programs operate as if other routines in the system were malicious or incorrect.

* A calling program cannot trust its called subprocedures to be correct, and a called subprocedure cannot trust its calling program to be correct.

Example

* A procedure to sort the entries in a list cannot be trusted not to *modify* those elements
* That procedure cannot trust its caller to provide any list at all or to supply the number of elements predicted.

Confinement

* A technique used by an operating system on a suspected program to help ensure that possible damage does not spread to other parts of a system.
* A **confined** program is strictly limited in what system resources it can access.
* More later on viruses…

Simplicity

After a system has been running for some time, and the designers and programmers are working on other projects (or perhaps even at other companies), a fault appears and some poor junior staff member is assigned the task of correcting the fault.

Simplicity by Hoare in 1981

“I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies.”

Genetic Diversity

* Many components of a system coming all from the same vendor, e.g. purchasing systems with software and hardware all from Apple or Microsoft
* Tight integration of programs (e.g. Windows, IE, Outlook, Office)

Design Principles for Security

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

*Least Privilege*

* Each user and each program should operate using the fewest privileges possible.
* In this way, damage from an inadvertent or malicious attack is minimized.
* The default condition should be denial of access.

*Separation of Privilege*

* Where feasible, access to objects should depend on more than one condition, such as user authentication plus a cryptographic key.
* Someone who defeats one protection system will not have complete access.

*Complete Mediation*

* Every access attempt must be checked.
* Both direct access attempts (requests) and attempts to circumvent the access-checking mechanism should be considered.
* The mechanism should be positioned so that it cannot be circumvented.

*Open Design*

* The protection mechanism must not depend on the ignorance of potential attackers; the mechanism should be public, depending on secrecy of relatively few key items, such as a password table.
* An open design is also available for extensive public scrutiny

*Economy of Mechanism*

* The design of the protection system should be small, simple, and straightforward.
* Such a protection system can be carefully analyzed, exhaustively tested, perhaps verified, and relied on.

*Least Common Mechanism*

* Minimize the amount of mechanism common to more than one user and depended on by all users.
* Shared objects provide potential channels for information flow.
* Systems employing physical or logical separation reduce the risk from sharing.

Example

* Given the choice of implementing a new function as a supervisor procedure shared by all users or as a library procedure that can be handled as though it were the user's own, choose the latter course.
* Then, users can provide a substitute or not use it at all. Either way, they can avoid being harmed by a mistake in it.

*Ease of Use*

If a protection mechanism is easy to use, it is unlikely to be avoided.

**Countermeasure: Secure Software Development Process**

* Presented properties that many examples of good code reflect
  + the properties are not a cause of good code but are paradigms that tend to go along with it
* Turn our sight from the product to the process: how is good software produced?

Software Development Tasks

* *Specify*
* *Design*
* *Implement*
* *Test*
* *Review*
* *Document*
* *Manage*
* *Maintain*

Techniques for Solid Software

* peer reviews
* hazard analysis
* good design
* prediction
* static analysis
* configuration management
* analysis of mistakes

Peer Reviews

Keep careful track of what each reviewer discovers and how quickly he or she discovers it

* particular reviewers need training
* certain kinds of faults are harder to find than others
* a fault could have been discovered earlier in the process (e.g. during a requirements review)
* Hazard Analysis

Usually involves developing hazard lists, as well as procedures for exploring “what if” scenarios. Examples:

* hazard and operability studies (HAZOP)
* failure modes and effects analysis (FMEA)
* fault tree analysis (FTA)

Good Design

* using a philosophy of *fault tolerance*
* having a consistent *policy* for handling failures
* capturing the *design rationale* and history
* using *design patterns*

Typical Failures

* failing to provide a service
* providing the wrong service or data
* corrupting data

How to Handle a Failure

Restore the system to its previous state and

* Retry; perform the service again, using a different strategy
* Correct; perform the service again, using the same strategy
* Report; do not provide the service again

Prediction

* If we think the risk of a particular security breach is small, we may not want to invest a large amount of money, time, or effort in installing sophisticated controls.
* Or we may use the likely risk impact to justify using several controls at once, a technique called “defense in depth”.

Static Analysis

* More than peer review & performed earlier
* We can examine several aspects of the design and code:
  + control flow structure
  + data flow structure
  + data structure
* Automated tools are available

Techniques for Solid Software  
(continued)

* peer reviews
* hazard analysis
* good design
* prediction
* static analysis
* configuration management
* analysis of mistakes

**Configuration Control**

who is making

which changes

to what

and when

Configuration Management

* ... is the process by which we control changes during development and maintenance.
* It scrutinizes new and changed code to ensure, among other things, that security flaws have not been inserted, intentionally or accidentally.

Configuration Management Activities

1. Configuration Identification
2. Configuration Control and Change Management
3. Configuration Auditing
4. Status Accounting

**Lessons from Mistakes**

* Document decisions—not only what we decided to do and why, but also what we decided *not* to do and why
* Document how we found and fixed the underlying faults
* Build checklists and codify guidelines

**Standards of Program Development**

* *Design* (specified design tools, languages, or methodologies)
* *Documentation, language*, and *coding style*
* *Programming (*peer reviews, periodic code audits, …*)*
* *Testing (*verification techniques, archiving of test cases, …*)*
* *Configuration management (*control access to and changes of stable program units*)*

**Countermeasure: Testing**

*“Developers grow trees;*

*testers manage forests”*

*James Whittaker*

Types of Testing

* Unit testing
* Integration testing
* Function test
* Performance test
* Acceptance test
* Installation test
* Regression testing

Exploratory Testing

* Simultaneous learning, test design and execution.
* On the opposite side of script testing.
* In reality, testing is almost always a combination.

Key Ingredients for Testing

1. Product Expertise
2. Coverage
3. Risk Analysis
4. Domain Expertise
5. Common Vocabulary
6. Variation
7. Boundaries

Testing Especially for Security

* Penetration testing (tiger team analysis or ethical hacking)
* Formal proofs of correctness
  + Halting problem
  + Works with assertions about the input

**Countermeasure: Defensive Programming**

*“offense sells tickets;*

*defense wins championships*”

Examples of wrong input values

* *incorrect number of parameters*
* *incorrect order of parameters*
* *inappropriate data type* (letters in a numeric field*)*
* *out of range for given use (*a negative value for age)
* *unreasonable* (250 kilograms of salt in a recipe*)*
* *out of scale or proportion (1 bedroom and 500 bathrooms)*

Programming by Contract

For each program module, write:

* Preconditions
* Postconditions
* Invariants

Assertions is a way to enforce them.

MALicious softWARE

* Programs or program parts planted by an agent with malicious intent to cause unanticipated or undesired effects
* Although they can result in similar disasters, the definition excludes
  + Unintentional errors
  + Coincidence

Virus

* A program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them
* The infection usually spreads at a geometric rate

Types of Viruses

* Transient: same lifespan as its host (runs when the hosts executes, terminates when host ends)
* Resident: can remain active or be activated as a stand-alone program

Worm

* A program that spreads copies of itself through a *network*.
* The worm spreads copies of itself as a *stand-alone program*.

Trojan Horse

A program that in addition to its primary effect, has a second, nonobvious, malicious effect.

Example: a login script that passes the identification information on to the rest of the system for login processing, but also retains a copy of the information for later, malicious use.

Zero-Day Attack

* Use of malware that exploits a previously unknown vulnerability or a known vulnerability for which no countermeasure has yet been distributed.
* The exploit window is diminishing rapidly.

Exploit Timeline

* The manufacturer becomes aware of the vulnerability.
* Code (called proof of concept) to demonstrate the vulnerability in a controlled setting.
* A patch or workaround that counters the vulnerability.
* Users implement the control.
* Actual attack.

Going Public

* A way to force a company to take action. Sometimes seems necessary.
* But...
  + Shrinking time between knowledge of vulnerability and exploit puts pressure on vendors and users both
  + Time pressure is not conducive to good software development or system management

Technical Details: Malicious Code

* Harm
* Transmission
* Activation
* Stealth

Harm

* Essentially unlimited
* Malware runs under the authority of the user, it can do anything the user can do

Transmission and Propagation

* During initial setup and installation
* Attachment
* Web site download
* Document virus
* Embedded scripts
* Autorun (e.g. .profile)
* Drive-by Downloads

“Good quality” Malware

* Hard to detect
* Not easily destroyed or deactivated
* Spreads infection widely
* Can reinfect its home program or other
* Easy to create
* Machine and OS independent

Memory-Resident Viruses

TSRs (terminate and stay resident) routines, such as the routine that:

* interprets keys pressed on the keyboard
* handles error conditions that arise during a program’s execution

Combination of techniques: a boot sector virus might attach itself to a piece of resident code

Interpretive Data

Data files can also bring trouble.

Example:

* The Adobe Reader program is an interpreter for PDF files.
* Adobe Reader does many things in response to commands in the PDF file.

Other Homes for Viruses

* Popular applications such as word processors and spreadsheets. Virus macro can
  + Get embedded in startup macros
  + Copy itself in other data files
* Libraries
* Compilers, loaders, linkers, runtime monitors, runtime debuggers, and even virus control programs

Detection

* Develop a screening program to reliably separate all infected modules from uninfected ones: “Are these two programs equivalent?” is undecidable (for general programs; it could be answered for specific pairs)
* However, we can look for specific instances of viruses (signatures)

Stealth in Storage: Avoid Patterns

* Rearrange the order of modules
* Rearrange the order of instructions
* Insert instructions (such as A:= A) that have no result
* Insert random strings (e.g. constants)
* Replace instructions with others of equivalent effect
* Insert instructions that are never executed

Vulnerability: Voluntary Introduction

* Click on link, open attachment,…
* Transmission is contained not because of limited contact but because of limited contact *outside the community*.
* Governments often run disconnected network communities for handling military or diplomatic secrets.

Vulnerability: Unlimited Privilege

* Many personal computers: one user who is all-powerful
* If the user inadvertently activates a piece of malicious code, that code runs with unlimited user privileges
* More prudent:
  + ordinary user: modify only the user data space
  + to modify system files requires the user to log out and log in as an administrator

Vulnerability: Stealthy Behavior—Hard to Detect and Characterize

* Malicious code can attach itself to benign programs
* Writers of malware vary their code’s effect

An Overlapping Controls Strategy of Countermeasures

* Hygiene
* Detection Tools
* Error Detecting and Correcting Codes
* Memory Separation
* Basic Security Principles

**Countermeasure: Hygiene**

* Use only commercial software acquired from reliable, well-established vendors.
* Open attachments—and other potentially infected data files—only when you know them to be “safe”.
* Test all new software on an isolated computer.
* Make a recoverable system image and store it safely.
  + keep the image write-protected during reboot
  + prepare an extra copy of the safe boot image
* Make and retain backup copies of executable system files.
* Recognize that any web site can be potentially harmful. Obvious sites:
  + run by and for hackers
  + serving pornography
  + scalping tickets
  + selling contraband

Also, be wary of sites located in certain countries: US, Russia (.ru), China (.cn), India, Brazil,…

**Countermeasure: Detection Tools**

* Generally effective
* Limitations:
  + Timeliness: retrospective, looking for patterns of known infections
  + Variation: patterns are necessarily static
* Herd immunity: free riding
  + Same logic discourages a victim company from reporting an attack

Virus Signature

* Malicious code must be stored somewhere.
* The code must be in memory to execute.
* The virus executes in a particular way, using certain methods to spread.
* There may be a recognizable pattern of the instructions.

The scanner searches memory and storage, monitoring execution, watching for the telltale signatures of viruses.

**Countermeasure: Error Detecting and Correcting Codes**

* Error detecting codes: detect when an error has occurred
* Error correcting codes: can actually correct errors without requiring a copy of the original data

Hash Codes

* A small change in a file can easily go undetected
* We need a “wax seal”
* Compute a cryptographic function, sometimes called a **hash** or **checksum** or **message digest** of the file
* Desired property: one-way function

One-Way Functions

* Functions that are much easier to compute than their inverses (e.g. x2, x3)
* Create a change detection algorithm: change to even a single bit will alter the checksum result
* Each time the file is accessed or used, the checksum is recomputed
* An attacker cannot “undo” the function to see what the original file was

Example: Tripwire

* You run Tripwire on a new system, and it generates a hash value for each file
* Save these hash values in a secure place (offline)
* If you later suspect your system may have been compromised, you rerun Tripwire, giving it the saved hash values.

Cryptographic Checksum

* MD4, MD5 (where MD stands for Message Digest)
* SHA/SHS (Secure Hash Algorithm or Standard)
* Attacks on SHA, MD4, and MD5
  + For SHA, find two plaintexts with the same hash digest in approximately 263 steps, far short of the 280 steps that would be expected of a 160-bit hash function

**Countermeasure:   
Memory Separation**

* Block processes from writing to areas where executable code is stored.
* Sensitive data are stored separately and are untouchable

Types of Separation

* Physical. Different processes use different physical objects (e.g. printers)
* Temporal. Processes having different security requirements are executed at different times.
* Logical. A program cannot access objects outside its permitted domain.
* Cryptographic. Processes conceal their data and computations.

**Countermeasure: Basic Security Principles**

* Least Privilege
  + Separating user and administrative rights
  + A subject should have access to the smallest number of objects necessary
* Complete Mediation
  + Every potential access to an object must be checked
  + Protects user data, programs, devices (e.g. USB) and services (such as email)

Recurring Thread: Legal

* Computer Crime

Legal Definitions Do Not Always Translate

* Property is tangible, unlike magnetic impulses.

From ‘84 California Supreme Court ruling on the theft of a trade-secret proprietary software package:

*It is the opinion of the Court that such impulses are not tangible and hence do not constitute an “article.”*

Another Example

* Computer Services
  + Unauthorized access to a computing system is not always punished in court as a serious crime. No physical object.
  + Compare it to a stranger entering a house

Rules of Evidence

* Courts prefer an original source document to a copy.
* The biggest difficulty with computer-based evidence in court is being able to demonstrate the authenticity of the evidence.

Computer Crime is Hard   
to Prosecute

* Lack of understanding
* Lack of physical evidence (fingerprints?)
* Lack of recognition of assets
* Lack of political impact
* Complexity of the case
* Age of defendant
* Negative publicity and copycat crimes

Social Engineering

* Use social skills and personal interaction to get someone to reveal security-relevant information.
* “People are *by nature* trusting and helpful”?

“*Hello, this is John Davis from IT support. We need to test some connections on the internal network. Could you please run the command ipconfig/all on your workstation and read to me the addresses it displays?”*

* The attacker often impersonates someone inside the organization who is in a bind:
  + “I have to get out a very important report quickly and I can’t get access to the following thing.”
* Often, the attacker impersonates someone in a high position, such as the division vice president or the head of IT security.

Presumed Innocence

* We tend not to ask a stranger why she is here, who she works for, or how to verify her identity.
* We are more likely to challenge a faceless individual at the other end of a telephone call or email message than someone in person.

**Vulnerability: Insiders**

* *Perimeter security*: envision a system’s security as a strong defensive wall surrounding the sensitive system and data inside.
* Divides the world into *outsiders* and *insiders*

Who is an Insider?

* Someone with legitimate access to an organization’s computers and networks
* Examples: contractor, auditor, ex-employee, temporary business partner (think of corner cases)

Factors Preventing   
Insider Threats

* Insiders hold positions of trust
* Layers of internal controls
* Separation of duty and least privilege

Rethink of the high school example.

Other Ways of   
Extracting Sensitive Data

* Sense the electromagnetic waves
* Measure how long it takes to process one kind of input versus another
* Write a monitor program to intercept the system call when a device driver receives a signal that a new character has arrived
* Look for data in deleted files

“Out, Damned Spot! Out, I Say!”

* Early Microsoft OSes didn’t erase a deleted file; they flagged its directory entry
* Even after the “recycle bin”, it was possible for one user to read the trash of another (create new large file)
* Magnetic devices retain some memory
  + erasure of seven-or-more passes
  + destroy the medium

**Failed Countermeasure: Security Through Obscurity**

The belief that a system can be secure as long as nobody outside its implementation group is told anything about its internal mechanisms. Examples:

* Hiding account passwords in binary files or scripts
* Deleting text (not actually deleted). See Sidebar 5-5 in the textbook.

**Countermeasure:   
Physical Access Control**

* Preventing access
  + Guard
  + Locks
  + Cards (radio transmitters, magnetic stripe)
* Detecting access
  + Logs
  + The equivalent of a “Honey Pot”

Countermeasure (Reminder):   
Strong Authentication

* One Time (dynamic) Passwords
* Password-Generating Tokens
* Challenge-Response Systems
* Response-Generating Token
* Continuous Authentication
* Increase Password Change Frequency

**Countermeasure:   
Trust/Least Privilege**

In the high school keylogging story…

* the operating system: detected a new piece of hardware and installed drivers to handle the device
* the grading program: one faculty member has little reason to change another teacher’s grades