

CS574

Computer Security

San Diego State University

Spring 2008

Lecture #13

Today's Structure

- Administrivia
- Recent News
- Questions
- Lecture

Administrivia

- Schedule
 - Assignment #1
 - Spring Break
 - Assignment #2
 - Midterm #2
- Assignment #1 Turnin Script available for beta-test
- Midterm appeals/question rewrites due next class

Recent News

- Cryogenically bypassing encryption.
- Ten Immutable Laws of Security
- Breaking out of a chroot jail.

Questions?

Lecture

Where We've Been

- Physical Security
- Cryptography
- Software Flaws/Malware
- Access Control & Security Models
- User Authentication
- Basic Operating System Protections

What's The Goal?

- Security
 - Confidentiality
 - Integrity
 - Availability
- “Secure”
 - implies an absolute have/don't have status.
- Is this really practical?

“Trusted” Systems

- We want a usable system.
- We need confidence that the security mechanisms are “good enough” for us.
- High-security applications have more at stake, and so require more security, and can tolerate less convenience.
- Need to have confidence in the security features before we can trust the system.

“Trustable” vs “Secure”

- A totally secure system may not be very usable, at least not for most users.
- A securable system may not be trustworthy.
- Need OS *and* hardware features.
- Security mechanisms must be in place and *working* at all times.

Pfleeger's Comparison

Secure	Trusted
<i>Either-or:</i> Something either is or is not secure.	<i>Graded:</i> There are degrees of "trustworthiness".
Property of the <i>presenter</i> .	Property of the <i>receiver</i> .
<i>Asserted</i> based on product characteristics.	<i>Judged</i> based on evidence and analysis.
<i>Absolute:</i> Not qualified as to how, where, when, or by whom used.	<i>Relative:</i> Viewed in context of use.
A <i>goal</i> .	A <i>characteristic</i> .

Underpinnings

- Policy
Need to know what should be done, up front.
- Model
Need to analyze the policy.
- Design
Need to be able to implement the policy/model.
- Trust
Need a basis for believing that it works correctly.

Design Elements

- Least Privilege
- Economy of Mechanism
- Open Design
- Complete Mediation
- Permissions-based
- Separation of Privilege
- Least Common Mechanism
- Ease of Use

Design Elements

Least Privilege

- Give no more access than is necessary to complete an assigned task.
- Intended to limit the harm a subverted or malicious subject can do.

Design Elements

Economy of Mechanism

- The simpler the protection system, the easier it is to trust that it works. (i.e., KISS)
- Analysis is *hard*. The less code there is to examine, the better the examiner will do.

Design Elements

Open Design

- You should not depend on the ignorance of your attackers – assume an attacker knows all about your protection mechanisms.
- This also can help identify (and remove) bugs in your design. (“Many eyes make for shallow bugs.”)

Design Elements

Complete Mediation

- You should not allow *anything* to circumvent your protection mechanism.
- Once a protection mechanism has been circumvented, you can't trust it anymore.

Design Elements

Permission based

- You should deny access by default / default should not be to grant access to the object.
- Also known as *fail-safe defaults*.

Design Elements

Separation of Privilege

- You should have more than one reason to grant access.
- If one security mechanism fails, all is not necessarily lost.

Design Elements

Least Common Mechanism

- Limit or otherwise control shared resources.
- Sharing resources is a potential means for “leaking” information.

Design Elements

Ease of Use

- Protection mechanisms that aren't difficult to use are less likely to be circumvented.
- Also known as Psychological Acceptability.

Security Features ('Regular' OS)

- (Identification and) authentication of users.
- Memory protection.
- Access control.
- Sharing enforcement.
- Fair service.
- IPC and synchronization.
- OS protection data is also protected.

Trusted Operating System

- Need all the features from ordinary OSes, except more stringent.
- Features *and* assurance that they work as expected (i.e., correctly).
- Assurance is *hard*.
- Correct design does not automatically result in a correct implementation.

Security Features (‘Trusted’ OS)

- Identification and authentication of all users
- MAC / DAC
- Object reuse protection
- Complete mediation.
- Auditing & audit log reduction.
- “Trusted Path”
- Intrusion Detection

User Identification and Authentication

- Higher standards than a 'regular' OS
- Multi-factor authentication
- Enforcement of strong pass{words,phrases}
- Password expiration policies

MAC / DAC

- MAC
 - *mandatory access control*
 - enforces site policy over owner's desires
- DAC
 - *discretionary access control*
 - for fine-grained access control
 - often dynamic
 - often a richer set of permissions

Object Reuse Protection

- Object Reuse Attacks
 - against memory / files / devices
 - anything shared that has “storage”
- Any object that can be reused must be *scrubbed*.
- Can scrub on release or on allocation.
- Must be a responsibility of the Operating System.

Complete Mediation

- Validate authorization for specific access.
- All accesses for all subjects to all objects.
- Validation done by a trusted component.
- When to validate:
 - every read/write
 - at open/request

Complete Mediation – Revocation

- Revocation of access
 - It's a good idea
 - It's “expensive” in time/resources.
- Results in a sizable performance hit.

Auditing

- Every access attempt by every subject to every object should be recorded, along with the result (granted/denied).
- *Logging* is the recording of events or statistics to provide information about system/user performance and activities.
- *Auditing* is the analysis of log records to obtain data of interest in a usable format.

Auditing

- If you don't know what's happening, you can't control the system.
- Accountability depends on knowing *who* did *what* and *when* they did it.
- Intruders (attackers) often wipe or modify logs to hide evidence of the attack and their activities.

Auditing

- Log records may be in textual or binary form.
- A single event may be logged in several places at various levels/categories.
- Who looks at the logs anyway?
- Typical automated system:
 Logger -> Analyzer -> Notifier -> Admin

Auditing

- Logging at too fine-grained a level results in recursive logging.
- Logging is expensive:
 - performance: every access writes to the logger
 - storage: the data needs to be put somewhere
- How to handle the glut of data?

Audit Log Reduction

- Too much information is hard to analyze. Two approaches:
 - We could reduce the amount of information (data volume) stored in logs.
 - We could summarize logs and analyze the summary (referring back to the logs when something suspicious is found in the summary).
- Can be part of the OS or an “external” function.

Log Sanitization

- Confidential data can “leak” into logs
- Two types of sanitizers:
 - anonymous
 - psuedo-anonymous
- Where to put the sanitizer?
 - between the logger and the log file
 - between the log file and analyzer/summarizer

Trusted Path

- Security-critical (trusted) functions require protection from malicious subjects.
- Needs a means of unmistakable communication with the OS - the *trusted path* to the operating system/trusted function.
- Goal is to avoid a man-in-the-middle attack.

Trusted Path

- Often a magic-key sequence (BREAK, CTRL+ALT+DEL, etc.)
- Can be at system startup (e.g., boot to single-user mode).

Intrusion Detection

- Look for patterns of intrusion.
- Signature versus Anomaly detection.
- Active versus Passive.

Kernelized Design

- The *kernel* (aka nucleus or core) of an OS is that part that handles the lowest-level functions.
- Designing security around the kernel is an old idea (but not the only way).
- Typically, the *security kernel* is contained within the operating system's kernel.

End Lecture

Lecture References

- Pfleeger, *Security in Computing*, 3rd & 4th edition
- Bishop, *Introduction to Computer Security*

Reading

- Chapter 5

Finis