Learning Objectives

- Overarching: understand the links between secure programming and software engineering
- Be able to identify and state secure programming principles
- Be able to explain assurance concepts
- Vulnerability reporting: Be able to name steps
Assurance

- Axiom: It is impossible to demonstrate with absolute certainty that a moderately complex application doesn't have any vulnerabilities.

- Second Best: We can provide assurance that an application was designed, implemented, tested, deployed and maintained in ways that decrease the chances of having vulnerabilities and other defects.
  - e.g., training in secure programming provides assurance
  - software engineering processes designed for assurance
How do you measure assurance?

- International Standard: Common Criteria
- Defines Evaluation Assurance Levels (EALs) 1-7
- EALs 3-4 commonly requested by governments and security-demanding organizations
- EAL 4 evaluation typically costs $1 million
- High assurance (EALs 5-7) is out of the scope of this class
- This section provides an overview of selected topics related to assurance
Question

- What is the name of the international standard that specifies assurance levels?
  b) Common Criteria
  c) Frequent Criteria
  d) The EAL
Question

- What is the name of the international standard that specifies assurance levels?
  
b) Common Criteria
  
c) Frequent Criteria
  
d) The EAL
List of Design Principles

1. Least Privilege
2. Fail Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation of Privilege
7. Least Common Mechanism
8. Psychological Acceptability

1. Least Privilege

- "A subject should only be given those privileges it needs in order to complete its task."

- Access control problem: how closely can access control (role-based, etc...) or capabilities match the needed privileges? At what cost?

- CWE ID 250
  - Often Misused: Privilege Management
  - “Cases in which an application grants greater access rights than necessary”
Example: IIS (web server)

- IIS 5 ran under the Local System account, equivalent to root privileges. It was possible, but a convoluted task, to get it to run under lower privileges.

- Fixed in IIS 6
  - Worker processes by default run as Network Service user accounts, a new, built-in account type with limited operating system privileges, on Windows Server 2003. All ASP built-in functions always run as low-privileged accounts (anonymous user).
Compartmentalization

- Technique to separate the code in different parts, so that each part runs with least privilege.
  - if a part is compromised, others may still be OK
  - Example: Separating a user interface from the program running with special privileges (e.g., root)
    - Good implementation examples in Linux
    - Bad idea: Windows task bar tray icons running with Local System privileges
      - Secunia advisory SA10949, Dell TrueMobile WLAN card utility
        - tray icon launches help with SYSTEM privileges
        - can be exploited to execute arbitrary code with SYSTEM privileges
2. Fail-Safe Defaults

- "Unless a subject is given explicit access to an object, it should be denied access to that object"
- Apache access control through .htaccess:
  first (default) rule: deny from all
  allow from ...
- Tied with the issue of failing “safe” vs failing “functional”
  - Switches that fail open as hubs under unusual circumstances
  - “Brittle” or catastrophic failures vs graceful
3. Economy of Mechanism

- Security mechanisms should be as simple as possible
- Complex mechanisms may not be correctly:
  - understood
  - modeled
  - configured
  - implemented
  - used
- Complex mechanisms may engender partial implementations and compatibility
Example Economy of Mechanism Problems

- **IPSEC:** Can do almost everything to secure TCP/IP but is complex
  - Sub-protocols with their own headers, which may be nested...
  - Every vendor’s implementation is slightly different
    - Vista compatibility
  - Partial implementations

- Some use SSL VPNs instead (e.g., OpenVPN)
  - Proven, robust
  - Simpler, easier to install and maintain
  - Compatible
4. Complete Mediation

- “All accesses to objects must be checked to ensure that they are allowed”

- Performance vs security issue
  - Results of access check are often cached
  - What if permissions have changed since the last check?
  - Mechanisms to invalidate or flush caches after a change are often missing

- Architecture issue
  - Capability granting and management
    - How did a capability given to Alice end up in Malory's hands?
Example Failure of Complete Mediation

- Access is checked only when opening a file, which returns a file descriptor.
- UNIX: forked and exec’ed processes inherit file descriptors.
- Even if processes call setuid to relinquish access to a high privilege, there may remain open files that stay open even if the process should now not be able to access them.
- Emacs used to have this problem.
Another example: CAN-2002-0871

- xinetd 2.3.4 leaks file descriptors for the signal pipe to services that are launched by xinetd, which could allow those services cause a denial of service via the pipe.
5. Open Design

- The security of a mechanism should not depend on the secrecy of its design or implementation.
- If the details of the mechanism leaks (through reverse engineering, dumpster diving or social engineering), then it is a catastrophic failure for all the users at once.
- If the secrets are abstracted from the mechanism, e.g., inside a key, then leakage of a key only affects one user.
- This does not mean you should reveal source code!
Example Failure of Open Design

- Electronic voting machines! Diebold voting machines source code analysis:
  - Passwords embedded in the source code.
  - Unauthorized privilege escalation and other vulnerabilities
  - Incorrect use of cryptography
  - Undetected, unlimited votes by voters
  - Insider threats - company workers or election officials can alter voters' ballot choices without their knowledge
  - (source: Kohno, Stubblefield, Rubin and Wallach, 2003 Johns Hopkins University)
Notes on Open Design

- Some hackers would rather have the binary than the source code when designing exploits
- Obscurity is OK if the design is secure regardless
  - If details are leaked, the software is still secure
6. Separation of Privilege

- “A system should not grant permission based on a single condition.”
- Removes a single point of failure
- Example: two-factor authentication
  - Requiring both biometric and token recognition systems reduces risks
- Analogous to the separation of duty:
  - By requiring multiple factors, collusion becomes necessary, and risks due to bribery (compromise of one factor) are reduced
  - Dual-signature checks
Successful Example

- UNIX: sudo allows the execution of commands with root privileges
- Possible only if
  - user (or attacker of compromised process) knows the appropriate password and
  - user is already member of an authorized group (e.g., wheel)

- (This example is from Bishop M., "Computer Security: Art and Science")
Notes on the Separation of Privilege

- Often confused with the principle of least privilege
  - e.g., OpenSSH has a "UsePrivilegeSeparation" option which really is an implementation of least privilege, in two parts of the code.
  - Compartmentalization is the technique used to separate code so that the principle of least privilege can be applied on these parts.
7. Least Common Mechanism

- “Mechanisms used to access resources should not be shared”

- Concept: You have two different services, of different priorities and value, provided to two different sets of users. The more they share resources, the more likely one can influence the other to:
  - Transmit forbidden data (covert channels issue)
  - Limit availability (denial of service)
Failed Example of Least Common Mechanism

- Microsoft NT architecture: FTP and Web services on the same computer shared a common thread pool. Exhausting the FTP thread pool caused failed connection requests for the Web service.

- CVE-1999-1148
  IIS processes passive FTP connection requests by assigning a thread to each port waiting for a client to connect
  - What if they never connect?
8. Psychological Acceptability

- Security mechanisms should not make the resource more difficult to access than if the security mechanism were not present.
- Example: Commercial where users have become bald and lost (all?) other hair in order to comply with a biometric authentication mechanism requesting hair samples.
- Problem: Users looks for ways to defeat the mechanisms and “prop the doors open”
- In practice, difficulty proportionate to the value of the protected asset is accepted
Example mechanism defeated for convenience's sake

- Trusted hosts -- if you're logged into host 'A', then you automatically have access to host 'B'
- .rhosts mechanism bypasses password security check
- .rhosts file in / directory allows remote root access without a password
- Authentication is based on IP addresses, which can be mapped to a different host through ARP poisoning
Recap of Design Principles

1. Least Privilege
2. Fail Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation of Privilege
7. Least Common Mechanism
8. Psychological Acceptability

Instructor: This list is to help with the exercise on the next slide
Processes

- Provide guarantees
- Areas using processes:
  - Development environment
  - Vulnerability response
  - Production (network administration, etc...)
  - Engineering
    - What assurances do you have about the safety of your code against unwarranted changes?
    - Who can get control version software accounts?
    - Are the access controls configured for security?
Development Environment

- Processes that we follow affect the security of the code/applications that we develop.
  - Security measures in the development environment are an EAL 3 assurance component
  - Do we lock our workstations when we leave?
  - Does the workstation lock automatically?
  - Do we hire convicted felons to work on our source code?
  - Is there a change control process?
  - How are backups handled? Are they secure?
Development Tools

- Can you trust the compiler? The linker?
- Perhaps the compiler on your workstation has been modified to insert a trojan...
  - See "Reflections on Trusting Trust" by Ken Thompson
Change Control

- Do we have authentication and access control for who can submit source code?
- Is there a QA process for code acceptance?
- Are changes justified and linked to a mandate (requirement, specification, bug report)?
- Is the integrity of the installer guaranteed?
- Are the source code repository and version control system secure?
  - Real life example: trojan in OpenBSD OpenSSH code (2002)
Vulnerability Response Process

- **Goals**
  - Protect customers
  - Protect sales
  - Produce a timely and robust patch
  - Present a consistent and accurate vendor viewpoint
  - Respond in a coordinated manner

- **Actors:**
  - Researcher
  - Vendor
  - 3rd Party Coordinator
  - Arbitrator
Vulnerability Lifecycle

Time

Vendor

Verify and Acknowledge → Create and Distribute Fix

Create Advisory

Researcher

Vulnerability Discovery → Report → Evaluate
Researcher

- Finds and reports vulnerabilities to:
  - Vendors
  - 3rd party coordinators
  - Public forums
    - full-disclosure
    - Bugtraq lists
- May create proof-of-concept code
Researcher Motivation

- Improving security
- Career advancement
- Gratification
- Curiosity
- Vendetta
- Other reasons known only to themselves
Vendor

- The software provider
  - Commercial entity (Symantec)
  - Open Source Software community
- Responsible for fixing vulnerabilities
3rd Party Coordinator

- Liaison between Vendors and Researchers
  - CERT

- Alternative to contacting vendor
  - Vendor is unavailable, unknown or unresponsive
  - Existing hostilities between vendor and researcher
  - Vulnerability affects multiple vendors
    - Protocol issues
    - Common components
Arbitrator

- Used when Researcher and Vendor cannot agree
- Acceptable to both parties
- Independent and impartial
Process

1. Vulnerability reported
2. Reported to product(s) primary and secondary contacts
3. Vulnerability is evaluated
4. Evaluation results reported to researcher
5. Fix timeline identified
6. PR / IT / Support notified
7. Fix created and distributed
8. Advisory written and posted
Vulnerability Lifecycle
Reporting Vulnerabilities

- NIAC: National Infrastructure Advisory Council
- Vulnerability disclosure working group
- 2004 report (Chambers and Thompson)
  - Public process documents, PGP keys, and advisories
    - www.example.com/security/
  - Email contact
    - security-alert@example.com
    - security@example.com
    - secure@example.com
    - support@example.com
    - info@example.com
Role of Cryptography

- Cryptography is involved in many design issues to make attacks much more difficult
  - Man-in-the-middle attacks
  - Sniffing
  - Replay attacks
  - Etc...
- Cryptography is not a do-it-yourself task!
  - 802.11b cryptography fiasco
- Cryptography doesn't solve common programming errors or design flaws
Responsible Disclosure

- Organization for Internet Safety
  - Responsible disclosure guidelines
- Idea: give the vendor a chance to produce patches, with a deadline
- Attempt to minimize the number of compromises
- Balance with claims that administrators should know (or have a right to know) about vulnerabilities in their systems
- There is no perfect solution that will make everyone happy. The “best” approach is to try to minimize harm
  - Even better is to not have vulnerabilities in the first place.
Any Questions?

- To do:
  - Get your VMWare workstation license and install VMWare
  - Start looking at the labs, to pick two that you want to do
  - Read 4 more pages of the Classes of Vulnerabilities and Attacks paper

- Next week: buffer overflows