Chapter 14: Design Principles

Overview

Secure system design follows key principles to minimize security risks. These principles help create systems that are easier to protect, manage, and understand.

Key Design Principles

1. Least Privilege
    - A subject (user, program, or process) should only have the minimum privileges needed to complete its task.
    - Access is granted based on function, not identity.
    - Rights are added as needed and removed after use to reduce risk.
    - Limits the protection domain to prevent unnecessary access.
2. Fail-Safe Defaults
    - Default action is to deny access unless explicitly allowed.
    - If an action fails, the system should remain as secure as before to prevent accidental breaches.
3. Economy of Mechanism (KISS Principle)
    - Keep It Simple and Secure (KISS)—simpler designs mean fewer errors and easier fixes.
    - Complex systems have more chances of failure and are harder to secure.
    - Simple interfaces and interactions reduce vulnerabilities.
4. Complete Mediation
    - Every access must be checked to ensure security.
    - Access control should be enforced every time a resource is used, not just on the first request.
    - Example: UNIX checks access only when opening a file—if permissions change later, unauthorized access may occur.
5. Open Design
    - Security should not rely on secrecy of the system's design.
    - Often misunderstood as requiring public source code.
    - "Security through obscurity" is not a real defense—true security comes from strong encryption and good practices.
    - Passwords and cryptographic keys must remain secret, but the overall design should not depend on secrecy.

6. Separation of Privilege
    - Granting privilege should require multiple conditions instead of a single factor.
    - Separation of duty ensures that no single person has too much control.
    - Defense in depth adds multiple layers of security.
7. Least Common Mechanism
    - Minimize shared resources between users and processes.
    - Shared mechanisms can create covert channels, where information leaks unintentionally.
    - Techniques like isolation, virtual machines, and sandboxes help reduce shared risks.
8. Psychological Acceptability
    - Security should not make systems harder to use.
    - Complexity introduced by security should be hidden from users.
    - Ease of installation, configuration, and use is critical.
    - Human factors matter—if security is too difficult, users will find ways to bypass it.

Key Takeaways

- Secure design is based on fundamental principles that apply to all security mechanisms.
- Good security requires a clear understanding of:
    - The system's goal and environment
    - Careful analysis, design, and implementation
- Applying these principles ensures stronger, more reliable, and user-friendly security.

Chapter 15: Representing Identity

15.1 Introduction to Identity

- What is Identity?
    - Identity is how a system uniquely identifies an entity (user, process, or object).
    - It enables security policies, access control, and user tracking.
- Multiple Identities
    - A single entity may have different identities in different contexts.
    - Example: A person may use their real name on government documents but a pseudonym on social media.

- Pseudonymity vs. Anonymity
    - Pseudonymity: Using a consistent alias instead of a real identity (e.g., a forum username).
    - Anonymity: Keeping identity completely hidden (e.g., using an anonymous email service).

15.2 Identity in Computing

- Principal:
    - A unique entity (user, process, or system) that can perform actions.
- Identity:
    - A way to refer to a principal within a system.
    - Examples:
        - A username (john_doe).
        - An IP address (192.168.1.1).
- Authentication:
    - The process of verifying a claimed identity.
    - Ensures that only authorized users access the system.
    - Examples:
        - Password-based login.
        - Biometric authentication (fingerprint, face recognition).

15.3 Names and Objects

- Objects in a System:
    - Files, processes, devices, and other system resources have identities.
    - Objects may have multiple names based on how they are accessed.
- Different Naming Contexts:
    - Human-readable names: File names like report.docx.
    - Process-level names: File descriptors (fd = 3 in UNIX).
    - Kernel-level names: Inodes, memory addresses (inode 1024).
- Example: UNIX Files
    - A file name is mapped to an inode (a unique identifier for the file).

- The same file name can refer to different files over time.
- A file descriptor always refers to the same file from opening to closing.

15.4 User Identities

- User Identification in UNIX Systems
  - Login Name: Used to log in (alice).
  - User ID (UID): A unique integer assigned to each user (UID = 1001).
  - The kernel identifies users by their UID, not their login name.
- Multiple UIDs in UNIX
  - Real UID: The original identity at login (unchangeable).
  - Effective UID: Used for access control (can change with setuid).
  - Saved UID: Holds the previous UID before switching privileges.
  - Audit/Login UID: Tracks the user's original identity (cannot be changed).

15.5 Groups and Roles

- Groups:
  - A group is a collection of users who share access rights.
  - UNIX has two models:
    1. Static groups: Users remain in a group permanently.
    2. Dynamic groups: Users can switch groups, changing their privileges.
- Roles:
  - A role is a set of privileges based on function rather than identity.
  - Example:
    - A system administrator (sysadmin) role has access to system settings.
    - A network administrator (netadmin) role manages network configurations.

15.6 Naming and Certificates

- Certificates:
  - Digital certificates bind a principal's identity to a cryptographic key.

    - Used in secure communications (e.g., HTTPS websites, encrypted emails).
- X.509 Distinguished Names:
    - Provides a structured way to uniquely identify people or organizations.
    - Example:
      /O=University of California/OU=Computer Science/CN=John Doe
    - O = Organization, OU = Organizational Unit, CN = Common Name.
- Certification Authorities (CAs):
    - CAs issue certificates after verifying identity.
    - VeriSign CA Levels:
        - Class 1: Email verification only (low trust).
        - Class 2: Name/address lookup (moderate trust).
        - Class 3: Background checks (high trust).
        - Class 4: Web server authentication (very high trust).

15.7 Certification Hierarchy

- Structure:
    - Root CA (IPRA) → Policy CAs (PCAs) → Regular CAs → Users.
- Example: University Certification
    - Students: Verified using student ID (low assurance).
    - Staff: Verified using biometrics (high assurance).

15.8 Identity on the Web

- Host Identity:
    - Static identifiers (IP address, MAC address) remain unchanged.
    - Dynamic identifiers change over time (assigned by DHCP).
- DNS and Spoofing:
    - Maps domain names (example.com) to IP addresses.
    - Attackers can poison DNS caches, redirecting users to malicious sites.

15.9 Dynamic Identifiers

- DHCP:
    - Assigns temporary IP addresses to devices.
    - Example: Your laptop gets a new IP address every time you connect to WiFi.
- Network Address Translation (NAT):
    - Allows multiple devices to share a single public IP address.

15.10 Cookies and Tracking

- Cookies:
    - Store user session data on the client side.
    - Helps websites remember login states, shopping carts, etc.
- Privacy Risks:
    - Third-party tracking: Advertisers can track users across different websites.

15.11 Anonymity and Privacy

- Anonymous Browsing:
    - Uses proxies or anonymizers to hide user identity.
- Anonymous Email:
    - Removes sender details before forwarding messages.
- Example: anon.penet.fi
    - Provided anonymous email forwarding.
    - Shut down after a court ordered user data disclosure.

15.12 Remailers and Traffic Analysis

- Cypherpunk Remailer (Type I): Strips sender identity before forwarding.
- Mixmaster Remailer (Type II): Encrypts messages and adds padding to prevent tracking.
- Traffic Analysis Attack:
    - Observes message flow to infer identities.
    - Countermeasures: Delayed sending, message padding, randomized forwarding.

15.13 Trust in Certificates

- Certificate Assurance:
    - Strong authentication = high trust (biometrics, passports).
    - Weak authentication = low trust (email verification).
- PGP Certificates:
    - Four trust levels: Generic, Persona, Casual, Positive.

15.14 Privacy and Anonymity Considerations

- Pros:
    - Protects whistleblowers.
    - Enables free speech.
- Cons:
    - Can be misused for cybercrime.
    - Harder to track criminals.

Final Thought:

- Anonymity is powerful but must be used responsibly.
- Balancing privacy and security is critical.

15.15 Final Takeaways

- Identity is crucial for security.
- Authentication ensures trust in systems.
- Certificates and CAs help verify digital identities.
- Anonymity and privacy have both benefits and risks.
- Responsible use of identity protection tools is essential.

Chapter 16: Access Control Mechanisms

16.1 Introduction to Access Control

- What is Access Control?
    - Access control determines who can access what in a system.
    - It protects data and resources from unauthorized access.
    - Examples include file permissions, user roles, and security policies.
- Types of Access Control Mechanisms:
    - Access Control Lists (ACLs) – Lists specifying user permissions for each object.
    - Capability Lists (C-Lists) – Users have tokens specifying their access rights.
    - Locks and Keys – Objects have locks, and subjects need matching keys to access them.
    - Ring-Based Access Control – Hierarchical privilege levels that control access.
    - Propagated Access Control Lists (PACLs) – Permissions spread across users and objects dynamically.

16.2 Access Control Lists (ACLs)

- What are ACLs?
    - ACLs define which users have what rights over specific objects (files, directories, etc.).
    - Each file has an associated list of (user, permission) pairs.
- Example ACL:
  File1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
  File2: { (Andy, r) (Betty, r) (Charlie, rwo) }
  File3: { (Andy, rwo) (Charlie, w) }
- Default Permissions in ACLs:
    - If a user is not listed, they do not have access by default (Fail-Safe Defaults Principle).
    - Some systems use groups or wildcards to grant permissions:
        - (holly, *, r): Holly can read the file regardless of her group.
        - (*, gleep, w): Any user in group "gleep" can write to the file.
- ACL Abbreviations (Shorter ACLs):
    - Instead of listing every user, UNIX uses 3 categories:

        - Owner
        - Group
        - Others
    - Example: rwx rwx rwx
        - First rwx: Owner’s permissions.
        - Second rwx: Group’s permissions.
        - Third rwx: Others’ permissions.
- Conflicts in ACLs:
    - If any ACL entry denies access, access is denied (Deny Overrides Rule).
    - Some systems (e.g., AIX) apply the first matching entry to determine access.

16.3 Capability Lists (C-Lists)

- What are C-Lists?
    - A row-based access control method where each user has a list of objects they can access.
    - Instead of storing access rules per file, rights are stored per user.
- Example C-Lists:
  Andy: { (file1, rx) (file2, r) (file3, rwo) }
  Betty: { (file1, rwxo) (file2, r) }
  Charlie: { (file1, rx) (file2, rwo) (file3, w) }
- C-List Characteristics:
    - Like a bus ticket – possession grants access.
    - Harder to revoke because users own their capabilities.
    - Must prevent unauthorized modification of C-Lists (e.g., a user should not upgrade their own rights).

16.4 Locks and Keys

- How Locks and Keys Work:
    - Objects have associated "locks".
    - Users (subjects) have "keys".
    - If a user’s key matches an object’s lock, access is granted.

- Differences from ACLs and C-Lists:
    - Dynamic – Locks/keys can change without modifying a user's access list.
    - Used in Cryptographic Systems:
        - Example: An encrypted file requires a decryption key to access.

16.5 Ring-Based Access Control

- What is Ring-Based Access Control?
    - A hierarchical access system using privilege levels (rings).
    - Lower-numbered rings have more privileges than higher-numbered rings.
- Example:
  Ring 0 (Highest Privilege) – Kernel Access
  Ring 1 – System Services
  Ring 2 – User Processes with Limited Privileges
  Ring 3 (Lowest Privilege) – Standard User Programs
- Processes in Ring 3 cannot directly access memory or hardware.
- Only Ring 0 processes can modify critical system data.
- Examples of Systems Using Rings:
    - Multics: 8 rings (0 to 7).
    - VAX (Digital Equipment Corporation): 4 rings (User, Monitor, Executive, Kernel).
    - Older systems: Only 2 privilege levels (User, Supervisor).

16.6 Propagated Access Control Lists (PACLs)

- What are PACLs?
    - Unlike regular ACLs, PACLs spread access rights dynamically based on user interactions.
- How it works:
    - When a user reads an object, the object's PACL attaches to the user.
    - When a user writes an object, the user's PACL attaches to the object.
- Example of PACLs:
    - If Betty reads Ann's file, Betty's PACL now contains Ann's PACL.
    - If Betty creates a new file, it inherits permissions from Ann's PACL.

16.7 Revocation of Access Rights

- How to Remove Access:
    - ACL-Based Systems: Owner removes the subject’s entry from the ACL.
    - C-List Systems: More difficult – requires scanning all user capability lists.
    - Solution: Use a global object table, where each capability refers to an entry in the table.
- Revocation in Windows NT ACLs:
    - Basic rights: read, write, execute, delete, change permission, take ownership.
    - Generic rights: no access, read, change, full control, special access.
    - Directories have additional rights: list, add, change, full control.

16.8 ACLs vs. Capability Lists

- ACLs (Access Control Lists):
    - Easy to answer: "Who can access this object?"
    - Used in most modern operating systems.
    - Harder to determine all files a user can access.
- C-Lists (Capability Lists):
    - Easy to answer: "What can this user access?"
    - Harder to manage revocation of rights.

16.9 Summary and Key Takeaways

- Access control mechanisms regulate user permissions over system resources.
- Different approaches include ACLs, C-Lists, Locks & Keys, Rings, and PACLs.
- ACLs are the most widely used, while C-Lists provide an alternative model.
- Revoking access is simpler in ACLs but harder in C-Lists.
- Ring-based access control is used for kernel vs. user privilege separation.
- Propagated ACLs dynamically inherit permissions based on user actions.

Each system has strengths and weaknesses, and the right approach depends on security needs and performance requirements.

Chapter 17: Information Flow

17.1 Introduction to Information Flow

- What is Information Flow?
    - Information flow refers to how data moves within a system.
    - It ensures that sensitive data does not flow to unauthorized locations.
    - Used in security policies to control access and prevent leaks.
- Types of Information Flow Mechanisms:
    - Compiler-Based Mechanisms – Detect information flow violations at compile time.
    - Execution-Based Mechanisms – Monitor and restrict information flow at runtime.
    - Security Pipeline Interface – Controls how data is transferred securely.
    - Secure Network Server Mail Guard – Prevents unauthorized information exchange between different security levels.

17.2 Basics of Information Flow

- Bell-LaPadula Model and Information Flow:
    - Information can only flow from a lower security level to a higher one.
    - If a compartment A has data and B is more privileged, data can flow from A to B only if B dominates A.
- Variables and Security Levels:
    - Each variable has a security compartment assigned to it.
    - Example:
        - If variable x belongs to security level A and y belongs to level B, data can flow from x to y only if $A \leq B$.
        - However, if $A > B$, then y := x is not allowed.

17.3 Types of Information Flow

- Explicit Information Flow:
    - Data moves directly from one variable to another through assignment.

- Example:
  x := y + z;
    - Here, information flows explicitly from y and z to x.

- Implicit Information Flow:
  - Data influences another variable without direct assignment.
  - Example:
    if x = 1 then y := 0 else y := 1;
    - The value of y depends on x, even though there is no direct assignment (y := x).
    - Implicit flow is harder to detect and can lead to security risks.

17.4 Compiler-Based Mechanisms for Information Flow

- How it Works:
  - The compiler analyzes code to detect unauthorized data flows.
  - It ensures that no insecure paths exist for sensitive information.
- Example of Compiler-Based Analysis:
  if x = 1 then y := a;
  else y := b;
  - Information flows from x and a to y or from x and b to y.
  - The compiler will check if x, a, and b can legally flow into y.
- Security Classes and Type Checking:
  - Variables are assigned security classes like Low and High.
  - Example:
    x: int class { A, B }
    - Means x belongs to the lowest upper bound (lub) of A and B.
    - Constants are always Low, meaning they don't carry sensitive data.

17.5 Information Flow in Procedures

- Input Parameters:
  - Carry information into a procedure.
  - Their class matches the security class of the actual argument.

- Output Parameters:
    - Carry information out of a procedure.
    - Security class must include input classes, since output can be influenced by input.
    - Example:
      proc sum(x: int class { A };
      var out: int class { A, B })
        - Requires x ≤ out and out ≤ out to maintain security.

17.6 Information Flow in Arrays

- Reading an array element (... := a[i]):
    - Information flows from i and a[i], so the security class is lub{ a[i], i }.
- Writing to an array element (a[i] := ...):
    - Information flows only to a[i], so its class remains the same.

17.7 Information Flow in Statements

- Assignment Statements:
  x := y + z;
    - Information flows from y and z to x, so this requires lub{ y, z } ≤ x.
- Conditional Statements:
  if x + y < z then a := b else d := b * c - x;
    - The statement executed reveals information about x, y, z.
    - Security rule: lub{ x, y, z } ≤ glb{ a, d }.
- Loops (while Statements):
  while i < n do begin a[i] := b[i]; i := i + 1; end
    - Must ensure the loop terminates.
    - Security rule: lub{ x1, …, xn } ≤ glb{ y | y is assigned in the loop }.

17.8 Execution-Based Information Flow Control

- How it Works:
    - Instead of checking at compile-time, these mechanisms monitor information flow during execution.

    - Stops execution if an unauthorized data flow is detected.
- Example Problem in Execution Monitoring:
  if x = 1 then y := a;
    - If x is High and y is Low, there is an implicit flow that an execution-based system must detect.
- Fenton's Data Mark Machine:
    - Each variable has an associated security class.
    - The program counter (PC) also has a security class.
    - Branches (if statements) are treated as assignments to PC, preventing implicit leaks.

17.9 Secure Execution Techniques

- Handling Errors in Information Flow:
    - If an error occurs (e.g., division by zero), revealing it could leak information.
    - Solution: Errors should only be reported if the user has the appropriate clearance.
- Preventing Infinite Loops with Sensitive Data:
  while x = 0 do
  (* nothing *);
  y := 1;
    - If x = 0, this loops forever.
    - If x = 1, y is set to 1.
    - This creates an implicit flow from x to y.

17.10 Secure Communication and Networking

- Security Pipeline Interface (SPI):
    - Controls how data moves between a host and external storage.
    - Example:
        - First Disk: Stores files.
        - Second Disk: Stores cryptographic checksums.
        - SPI retrieves the file, checks its checksum, and only allows access if they match.
- Secure Network Server Mail Guard (SNSMG):
    - Controls the flow of email between secure and non-secure networks.

- Analyzes messages, checks sender authorization, and sanitizes sensitive content before forwarding.

17.11 Summary and Key Takeaways

- Information flow control ensures sensitive data does not leak to unauthorized locations.
- Explicit flows occur through direct assignments, while implicit flows happen through control structures like if statements.
- Compiler-based approaches detect issues at compile-time, while execution-based approaches monitor at runtime.
- Loops and errors must be handled carefully to prevent unintended information leakage.
- Security mechanisms like SPI and SNSMG help enforce secure data communication between different security levels.

Each of these techniques is important for preventing data breaches and ensuring proper confidentiality policies are followed.

Chapter 18: Confinement Problem

18.1 Introduction to the Confinement Problem

- What is the Confinement Problem?
  - The problem of preventing a server from leaking confidential information to unauthorized entities.
  - Important for systems handling sensitive data such as banking, healthcare, and military systems.
- General Scenario:
  - A client sends a request to a server.
  - The server processes the request and returns a result.
  - The server must not access unauthorized resources or leak the client's data.

18.2 Total Isolation as a Solution

- Complete Isolation:
  - A process is fully isolated from all other processes and cannot communicate with them.
  - In theory, this prevents all leaks but is not practical because:

        - Processes need to access CPU, storage, and networks.
        - Complete isolation would prevent useful functionality.

18.3 Covert Channels

- Definition:
    - A covert channel is an unauthorized method of communication that exploits shared resources.
    - It allows data to be secretly transferred between processes that are not supposed to communicate.
- Example of a Covert Channel:
    - Two processes, P and Q, are not allowed to communicate.
    - However, they share a file system.
    - P writes either "0" or "1" as a filename, then creates a "send" file.
    - Q detects "send," reads the file to get "0" or "1," then deletes the files and waits for the next bit.
    - This exploits file creation as a storage covert channel.

18.4 Rule of Transitive Confinement

- Definition:
    - If a process P is confined to prevent leaks, and it invokes another process Q, then Q must also be confined.
    - Ensures that an unconfined process cannot be used to bypass security.

18.5 Timing Attacks and Side Channels

- Kocher’s Attack Example:
    - Computes x = az mod n, where z = z0 … zk-1.
    - Run-time depends on the number of 1 bits in z.
    - By measuring execution time, an attacker can infer z and break encryption.

18.6 Isolation Techniques

- Virtual Machines (VMs):

    - Simulate a separate computer environment.
    - Each VM runs independently, preventing direct access to the main system.
    - Example: IBM VM/370, DEC VAX VMM.
- Sandboxes:
    - Restrict what a process can access within the system.
    - Often used for running untrusted programs safely.
    - Examples:
        - Java Virtual Machine (JVM): Restricts applet access to system resources.
        - Sidewinder Firewall: Uses Type Enforcement to confine processes.

18.7 Trapping System Calls for Security

- Janus Execution Environment:
    - Restricts objects and limits modes of access at runtime.
    - Uses configuration files to define allowed and denied actions.
- Example Janus Configuration:
    - Deny access to all files except /usr/*.
    - Allow reading of /lib/* and execution of /bin/*.

18.8 Covert Channel Types

- Covert Storage Channel:
    - Uses shared resources like file names, memory locations, or disk space.
    - Example: Using a directory as a communication medium by creating or deleting files.
- Covert Timing Channel:
    - Uses timing or order of execution to transmit data.
    - Example: CPU scheduling time can be used to send "0" or "1" bits.

18.9 Example of Covert Timing Channel

- KVM/370 Virtual Machine Covert Timing Channel:
    - Two virtual machines VM1 and VM2 share a CPU.

- VM1 delays execution to send a "1" bit.
- VM2 measures response time to receive the bit.
- The CPU time acts as a covert channel.

18.10 Detecting Covert Channels

- Covert Flow Trees:
  - Represent information flow within a system.
  - Help identify attributes that can be modified and detected.
- Example:
  - If a file attribute (e.g., "locked" or "open") is modifiable and detectable, it can be exploited as a covert channel.

18.11 Mitigating Covert Channels

- Goal:
  - Obscure the amount of resources a process uses.
  - Make it hard for attackers to distinguish actual signals from noise.
- Methods:
  - Fixed Resource Allocation:
    - Each process gets the same fixed CPU time, disk space, and memory.
  - Randomized Scheduling:
    - Introduce random delays to obscure timing-based attacks.

18.12 Example: The Pump Method

- How it Works:
  - Acts as an intermediate buffer between High and Low security processes.
  - Ensures that Low processes do not learn if High processes are transmitting data.
  - Helps reduce the bandwidth of covert timing channels.

18.13 Summary and Key Takeaways

- The confinement problem is about preventing unauthorized information leaks.
- Covert channels exploit shared resources to bypass security restrictions.
- Isolation techniques like VMs and sandboxes help reduce information leakage.
- System call monitoring (e.g., Janus) can enforce security policies dynamically.
- Covert channels can be minimized but are difficult to eliminate completely.
- Techniques like resource allocation and randomization help mitigate covert channels.

By using a combination of virtualization, sandboxing, and covert channel analysis, systems can limit unauthorized data leaks and protect sensitive information.

Chapter 19: Assurance

19.1 Introduction to Assurance

- What is Assurance?
    - Assurance is the confidence that a system meets its security requirements based on evidence.
    - It provides justification that the system is trustworthy.
- Why is Assurance Important?
    - Prevents security failures due to design flaws, implementation errors, and operational mistakes.
    - Helps detect vulnerabilities before attackers exploit them.
- Trust and Assurance
    - Trustworthy System: A system that has credible evidence proving it meets security requirements.
    - Trust: The level of confidence in a system's trustworthiness, based on available evidence.

19.2 Problems Due to Lack of Assurance

- Sources of Security Problems:
    - Mistakes in requirements definition.
    - Flaws in system design.
    - Hardware implementation issues, such as chip defects.
    - Software bugs, including programming and compiler errors.
    - Operational mistakes, such as misconfigurations.

    - Intentional system misuse, such as insider threats.
    - Hardware failures and environmental factors, like power outages.
    - Poor maintenance and faulty upgrades, leading to vulnerabilities.
- Real-World Examples of Failures:
    - Challenger explosion: Sensors removed from booster rockets caused system failure.
    - Therac-25 radiation therapy failures: Software design flaws led to fatal overdoses.
    - Bell V-22 Osprey crashes: Failure to correct malfunctioning components resulted in deadly accidents.
    - Intel 486 bug: A flaw in trigonometric functions caused incorrect calculations.

19.3 Role of Security Requirements

- Security Requirements: Statements of security goals that must be met.
- Types of Security Requirements:
    - High-level objectives: Broad goals like "The system must protect confidential data."
    - Concrete requirements: Specific, testable rules like "All files must be encrypted using AES-256."

19.4 Types of Assurance

- Policy Assurance:
    - Ensures that security requirements in the policy are complete, consistent, and technically sound.
- Design Assurance:
    - Provides evidence that the system design meets security requirements.
- Implementation Assurance:
    - Ensures that the implementation follows the security design and does not introduce vulnerabilities.
- Operational Assurance:
    - Ensures that the system remains secure during installation, configuration, and day-to-day use.
    - Also called Administrative Assurance.

19.5 Security Life Cycle and Assurance

- Stages of the Security Life Cycle:
    1. Conception: Idea generation, feasibility analysis, and security considerations.
    2. Manufacture: Planning, system development, and security integration.
    3. Deployment: Delivering the system, ensuring secure installation and configuration.
    4. Fielded Product Life: Ongoing maintenance, patching, and eventual system decommissioning.

19.6 Life Cycle Models and Assurance

- Waterfall Model:
    - Stages:
        1. Requirements definition and analysis.
        2. System and software design.
        3. Implementation and unit testing.
        4. Integration and system testing.
        5. Operation and maintenance.
    - Security requirements should be considered at every stage.
- Other Development Models:
    - Exploratory Programming:
        - Focuses on building a system quickly, often without clear security requirements.
        - Low assurance because security is not a priority.
    - Prototyping:
        - Develops an initial system to test security requirements before full implementation.
        - Future iterations allow for applying assurance techniques.
    - Formal Transformation:
        - Uses mathematical proofs to ensure the system is secure.
        - High assurance but complex to implement.
    - System Assembly from Reusable Components:
        - Requires assurance that each component is secure and that they work securely together.
        - Difficult to verify, especially in large systems.
    - Extreme Programming (XP):

        - Focuses on rapid development and iteration.
        - Security is often considered after development, requiring additional assurance efforts.

19.7 Building Security In vs. Adding Security Later

- Security Should Be Designed In:
    - Performance and security should be considered from the beginning, not as an afterthought.
    - Example:
        - If a system lacks fundamental security features, later modifications will be expensive and difficult.
- Reference Validation Mechanism (RVM):
    - Implements the Reference Monitor, which enforces access control rules.
    - Must be:
        - Tamperproof: Cannot be modified by attackers.
        - Complete: Must always be invoked and cannot be bypassed.
        - Simple: Small enough to be analyzed and tested for correctness.

19.8 Security Mechanisms for Assurance

- Security Kernel:
    - Combines hardware and software to enforce security policies.
- Trusted Computing Base (TCB):
    - Includes all protection mechanisms responsible for enforcing security policies.
    - Composed of both hardware and software.

19.9 Challenges of Adding Security Later

- Difficulties in Post-Development Security Implementation:
    - Increased complexity makes analysis harder.
    - Security may be spread throughout the system, making testing difficult.
    - Hard to validate security mechanisms when they are patched in later.
- Example: UNIX Security Enhancements by AT&T

- SV/MLS (System V Multi-Level Security):
    - Added Mandatory Access Controls (MAC) to UNIX.
    - Used existing kernel but lacked Least Privilege enforcement.
- SVR4.1ES:
    - Rebuilt UNIX kernel to include MAC and Least Privilege from the start.
    - More secure and efficient than modifying an existing system.

- Comparison of File Security:
    - SV/MLS Approach:
        - Used a separate table for MAC labels.
        - Required two accesses to check permissions, increasing complexity.
    - SVR4.1ES Approach:
        - Integrated MAC labels into the inode structure.
        - One access check, reducing security risks and improving efficiency.

19.10 Summary and Key Takeaways

- Assurance is essential for determining the trustworthiness of a system.
- Different levels of assurance exist, from informal evidence to rigorous mathematical proofs.
- Assurance must be applied throughout the system life cycle, from design to operation.
- Building security in from the start is more effective than trying to add it later.
- Reference Validation Mechanisms (RVMs) and Trusted Computing Bases (TCBs) help enforce security.
- Adding security after development increases complexity and reduces effectiveness.

By focusing on early security integration and rigorous assurance methods, organizations can reduce vulnerabilities and build trustworthy systems.

Chapter 22: Evaluating Systems

22.1 Introduction to System Evaluation

- What is System Evaluation?
    - The process of determining whether a system meets specific security requirements under specific conditions.

    - The goal is to assess how trusted a system is based on evidence and evaluation methodologies.
- Why Evaluate Systems?
    - Ensures security compliance with standards.
    - Identifies vulnerabilities and weaknesses.
    - Provides independent expert assessment of security functionality.
    - Helps organizations manage risk by validating security claims.

22.2 System Evaluation Methodology

- Provides a set of requirements defining security functionality for a system.
- Establishes assurance requirements to determine if a system meets security expectations.
- Uses formal methodologies to measure trustworthiness based on security evidence.
- Produces a final rating that quantifies the system's level of trust.

22.3 Trusted Computer System Evaluation Criteria (TCSEC)

- Also known as the Orange Book.
- Developed by the U.S. Department of Defense (DoD) in 1983.
- Part of the Rainbow Series of security guidelines.
- Main Focus:
    - Based on the Bell-LaPadula model and reference monitor concept.
    - Emphasizes confidentiality over integrity and availability.

22.4 Functional Security Requirements in TCSEC

- Discretionary Access Control (DAC)
    - Controls sharing of named objects.
    - Manages access rights, ACLs, and object reuse.
- Mandatory Access Control (MAC) (B1 level and above)
    - Uses security labels to enforce strict access control.
    - Labels define security classifications and clearances.
- Identification and Authentication (I&A)

    - Ensures each user has a unique identity.
    - Protects authentication data from unauthorized access.
- Audit Requirements
    - Defines what events must be recorded and monitored.
- System Architecture Security
    - Enforces process isolation, least privilege, and trusted path.
- Trusted Facility Management (B2 and above)
    - Separates administrator and operator roles.
    - Ensures trusted recovery after failures.

22.5 Assurance Requirements in TCSEC

- Configuration Management (B2 and above)
    - Ensures consistency between documentation, code, and tools.
- Design Specification and Verification
    - B1: Informal security policy model.
    - B2: Formal security policy model.
    - B3: High-level design consistency checks.
    - A1: Formal security verification methods.
- Testing Requirements
    - Evaluates penetration resistance and flaw correction.
- Trusted Distribution (A1 level)
    - Ensures software and security updates remain secure during distribution.

22.6 TCSEC Evaluation Classes

- Class A (Highest Security Level)
    - A1: Verified protection with strict security verification.
- Class B (High Security Level)
    - B3: Full security domains and reference validation mechanisms.
    - B2: Structured security with MAC for all objects.

    - B1: Labeled security protection with MAC for some objects.
- Class C (Medium Security Level)
    - C2: Controlled access protection with audit features.
    - C1: Discretionary access control with minimal assurance.
- Class D (Lowest Security Level)
    - D: Systems that fail to meet higher security requirements.

22.7 Evaluation Process in TCSEC

- Step 1: Application Submission
    - Vendors request evaluation from the government.
    - The request may be denied if the government does not require the product.
- Step 2: Preliminary Review
    - Evaluators assess technical content, security design, and schedule.
- Step 3: Evaluation Phase
    - Three stages:
        - Design Analysis: Reviewing system documentation.
        - Test Analysis: Testing security mechanisms and penetration resistance.
        - Final Evaluation Report: System is rated based on results.

22.8 Ratings Maintenance Program (RAMP)

- Allows vendors to update their security evaluation when making small product changes.
- Major changes may require a full re-evaluation.

22.9 Limitations of TCSEC

- Scope Limitations:
    - Focused only on operating systems, not networks or applications.
    - Only addressed confidentiality, ignoring integrity and availability.
    - Primarily designed for U.S. government use, not commercial needs.
- Process Limitations:

- Evaluation was slow and resource-intensive.
- Criteria evolved over time, making it inconsistent.
- Evaluations were only recognized within the U.S.

22.10 Federal Information Processing Standard (FIPS) 140

- FIPS 140 is a standard for evaluating cryptographic modules.
- Developed by U.S. and Canadian security agencies.
- Security Levels in FIPS 140:
  - Level 1: Basic cryptographic security with no physical protection.
  - Level 2: Tamper-resistant security with role-based authentication.
  - Level 3: Enhanced physical security and identity-based authentication.
  - Level 4: Advanced protections against environmental and physical attacks.
- Impact of FIPS 140:
  - Improved quality and security of cryptographic modules.
  - Helped vendors identify security flaws in cryptographic implementations.

22.11 Common Criteria (CC)

- Developed in 1998 as an international standard (ISO 15408).
- Recognized globally by multiple countries.
- Key Components of Common Criteria:
  - Target of Evaluation (TOE): The system being tested.
  - Security Target (ST): The security claims for a specific product.
  - Protection Profile (PP): A generic set of security requirements for a category of products.
- Evaluation Assurance Levels (EALs):
  - EAL1 – Functionally tested
  - EAL2 – Structurally tested
  - EAL3 – Methodically tested and checked
  - EAL4 – Methodically designed, tested, and reviewed
  - EAL5 – Semi-formally designed and tested

    - EAL6 – Semi-formally verified design and tested
    - EAL7 – Formally verified design and tested
- CC Evaluation Process:
    - Products are evaluated based on their ST and PP.
    - The evaluation process verifies that the security requirements are met.

22.12 Systems Security Engineering Capability Maturity Model (SSE-CMM)

- Based on Software Engineering CMM (SE-CMM).
- Focuses on evaluating the security engineering process, not the product.
- Uses maturity levels to assess an organization's security practices.
- Key Process Areas in SSE-CMM:
    - Administer security controls.
    - Assess security risks and threats.
    - Verify and validate security measures.
- Maturity Levels in SSE-CMM:
    - Level 1: Performed informally.
    - Level 2: Planned and tracked.
    - Level 3: Well-defined processes.
    - Level 4: Quantitatively controlled.
    - Level 5: Continuously improving.

22.13 Summary and Key Takeaways

- System evaluation ensures products meet security requirements.
- TCSEC (Orange Book) was the first major security evaluation standard.
- FIPS 140 focuses on cryptographic module security.
- Common Criteria (CC) is the modern global standard for evaluating system security.
- SSE-CMM evaluates the security engineering process rather than the product.
- Evaluations help organizations verify security claims, improve trust, and manage risks effectively.