Chapter 14: Design Principles

1. Least Privilege

Definition: Each user or program should operate using the least amount of privilege necessary.

Example: A backup program should only have read access, not write access.

2. Fail-Safe Defaults

Definition: Deny access by default and only allow explicitly.

Example: If a user is not listed in the access control list, they are denied access.

3. Economy of Mechanism

Definition: Keep system design simple.

Example: A small codebase is easier to audit for security issues.

4. Complete Mediation

Definition: Every access should be checked against access rights.

Example: In UNIX, once a file is opened, access checks are not repeated; this violates complete mediation.

5. Open Design

Definition: Security should not depend on secrecy of the system design.

Example: Sharing cryptographic algorithm openly but keeping the key secret.

6. Separation of Privilege

Definition: Require more than one condition for access.

Example: A withdrawal may need both a password and OTP.

7. Least Common Mechanism

Definition: Avoid shared mechanisms between users to prevent leakage.

Example: Use virtual machines to isolate users.

8. Psychological Acceptability

Definition: Security features should not make systems hard to use.

Example: Auto-login with short timeouts rather than frequent password prompts.

▧ Chapter 15: Identity

Identity Concepts

Principal: Unique entity (user, host, etc.).

Identity: Internal representation.

Authentication: Verifying principal matches identity.

File & Object Identity
Different systems assign different identifiers (e.g., file names vs. inodes).

Multiple Identities in UNIX
Real UID, Effective UID, Saved UID, Login UID

Example: setuid program changes effective UID temporarily.

Groups & Roles
Groups: Share permissions.

Roles: Function-based, e.g., netadmin vs. sysadmin.

Certificates & Names
Use X.509v3 to uniquely identify individuals.

Example:

sql
Copy
Edit
/O=University/OU=Dept/CN=User
Certificate Hierarchies
IPRA → PCA → CA → End user

Example: Verisign Class 1, 2, 3 with increasing identity verification.

Persona Certificates
Used for anonymity.

Example: A whistleblower using anon certificate for media.

Cookies
Store state between web sessions.

Example: Shopping cart remembers book IDs using cookies.

Anonymity on the Web
anon.penet.fi: Replaced real identity with anonymous email.

Cypherpunk Remailer: Strips headers, uses encryption.

Mixmaster: Adds padding, fragmentation for secure anonymity.

▧ Chapter 16: Access Control Mechanisms
1. Access Control Lists (ACLs)
Definition: Permissions attached to objects.

Example:

scss
Copy
Edit
file1: {(Andy, rx), (Betty, rwxo)}
2. Capability Lists (C-Lists)
Definition: Permissions attached to users.

Example:

scss
Copy
Edit
Andy: {(file1, rx), (file2, r)}
3. Locks and Keys
Definition: Object has a lock, user has a key.

Example: IBM 370 uses access keys and fetch bits.

4. Ring-Based Access Control
Definition: Privileges determined by execution rings.

Example: Ring 0 = kernel, higher rings = less privilege.

5. Propagated ACLs (PACLs)
PACLs track access through objects.

Example: If Betty reads Ann’s file, new files have PACL = PACLBetty ∩ PACLAnn.

Other Concepts
Revocation: Difficult in C-list; use indirection.

Dynamic ACLs: Cisco routers use time-based access rules.

Type Checking: Prevent misuse of object types (e.g., no writing to directory object).

▧ Chapter 17: Information Flow
Basics
Bell-LaPadula Model: Data can flow from lower to higher security but not vice versa.

Explicit vs Implicit Flow
Explicit: Direct assignment.

Implicit: Based on control flow.

Example:

go
Copy
Edit
if x = 1 then y := 0 else y := 1
Compiler-Based Mechanisms
Detect info flow violations at compile-time.

Example:

go
Copy
Edit
if x = 1 then y := a; else y := b;
Execution-Based Mechanisms
Runtime enforcement.

Example: Fenton’s Data Mark Machine uses a stack to track data classes.

Special Constructs
Semaphores: Require checking flow from semaphore to next write.

Loops and Conditional: Flow conditions checked.

Example:

arduino
Copy
Edit
while x < 10 do y := y + 1;
Anonymity and Integrity
SPI (Security Pipeline Interface): Checks file integrity using cryptographic checksums.

SNSMG (Mail Guard): Filters and sanitizes email between secure and public networks.

UNIT 3: Chapter 18 - Confinement Problem
🔒 1. The Confinement Problem
Definition: Prevent a service (server) from leaking confidential data.

Example: Bank server handling account balances should not leak client data or misuse access.

🔄 2. Total Isolation (Ideal but Impractical)
Processes can't communicate or be observed—unrealistic due to shared resources.

🧱 3. Covert Channels
Definition: Unauthorized communication paths via shared system resources.

Types:

Storage Channels: Use shared files, variables.

Example: Two processes use file creation/deletion to communicate bits.

Timing Channels: Use time-based resource access.

Example: Use of CPU or disk seek time to transmit bits covertly.

📏 4. Rule of Transitive Confinement
If process p is confined and calls q, then q must also be confined.

💻 5. Isolation Techniques
Virtual Machines (VMs):

Emulate hardware; each VM acts as a separate subject.

Example: IBM’s KVM/370 restricts VM communication by policy.

Sandboxes:

Restrict environment via security policies.

Example: Java VM limits applet file/network access.

🧰 6. Sandboxing Tools
Janus:

Runtime restriction framework using configuration files.

Example: Denies access to all files except /usr/* and specific library/executable paths.

🔍 7. Detecting Covert Channels
Covert Flow Trees: Analyze flow of information through shared resources.

Identify:

Modification of attributes.

Recognition by the receiver (direct or inferred).

Example: Use of Lockfile, Openfile, Filelocked to identify and recognize covert file locking behavior.

🧯 8. Mitigation Techniques
Goal: Obscure usage of shared resources.

Techniques:

Equal resource allocation

Random delays

Example: The Pump adds delay in acknowledgments to obscure timing information between high and low processes.

▧ UNIT 3: Chapter 19 - Introduction to Assurance
☑ 1. What is Assurance?
Definition: Confidence that system meets security requirements based on evidence.

Trust vs. Assurance:

Trust: Belief the system will behave correctly.

Assurance: Evidence-based confidence.

⚠ 2. Sources of Security Problems
Requirement flaws

Design flaws

Hardware/software bugs

Misuse and operational errors

Environmental events

Example: Challenger disaster due to rushed decisions and ignored sensor inputs.

🏗 3. Types of Assurance
Policy Assurance: Policy completeness and correctness.

Design Assurance: Design aligns with policy.

Implementation Assurance: Code matches design.

Operational Assurance: Secure installation, configuration, and operation.

🔄 4. Software Development Life Cycle (SDLC) Models
Waterfall Model:

Sequential stages: requirements → design → implementation → testing → maintenance.

Other Models:

Prototyping: Build prototype to understand requirements.

Formal Transformation: Rigorous, mathematically provable.

Extreme Programming: Rapid iterations, ongoing requirement updates.

🔐 5. Building Security In vs. Adding Later
Key Idea: Security should be integrated from the start, like performance.

Problems with Add-On Security:

Harder analysis, potential inconsistencies, limited assurance.

Example Comparison:

SV/MLS: Retrofits security into UNIX, leads to inefficiencies.

SVR4.1ES: Redesigned UNIX kernel with integrated MAC and least privilege principles.

◐ 6. Reference Validation Mechanism (RVM)
RVM: Implementation of access control (reference monitor).

Must be:

Tamperproof

Always invoked

Simple enough for verification

Trusted Computing Base (TCB): Includes all components enforcing security—hardware and software.

1. Goals of Evaluation
Purpose: Demonstrate that a system meets specific security requirements under defined conditions, making it a trusted system.

Methodology: Uses formal evaluation techniques to measure trust based on security requirements and assurance evidence.

Example: An independent assessment by experts ensures that security requirements are consistent, complete, and technically sound, while also verifying proper system documentation.

2. Evaluation Methodologies
The document discusses several evaluation frameworks:

a. Trusted Computer System Evaluation Criteria (TCSEC) – Orange Book (1983–1999)
Developed by: U.S. Department of Defense.

Focus: Primarily on confidentiality (based on the Bell-LaPadula model).

Functional Requirements:

Discretionary and mandatory access control (e.g., B1 level requires MAC for some objects).

Object reuse, labeling, audit, trusted path (e.g., B2 requires a trusted path between user and TCB).

Assurance Requirements:

Design specification, testing, and documentation (e.g., A1 requires formal methods for verification).

Evaluation Classes:

A (Verified Protection): Highest assurance (e.g., formal methods used).

B (Mandatory Protection): Structured protection (e.g., B2 requires covert channel analysis).

C (Discretionary Protection): Basic security (e.g., C2 includes auditing).

D: Minimal or no protection.

Example: The Orange Book's influence led to commercial awareness of security needs, though its limitations (e.g., lack of integrity/availability focus) spurred newer methodologies.

b. FIPS 140 (1994–Present)
Purpose: Evaluate cryptographic modules.

Levels:

Level 1: Basic security (e.g., FIPS-approved algorithms).

Level 2: Physical tamper-proofing (e.g., tamper-evident seals).

Level 3: Enhanced physical security (e.g., identity-based authentication).

Level 4: Highest protection (e.g., environmental attack detection).

Impact: Improved cryptographic module quality, though early evaluations revealed flaws in 50% of modules.

c. Common Criteria (CC) (1998–Present)
International Standard: ISO 15408, adopted by multiple countries.

Key Concepts:

Protection Profile (PP): Generic security requirements for a product family (e.g., firewall PP).

Security Target (ST): Specific requirements for a product (e.g., derived from a PP).

Evaluation Assurance Levels (EALs): Levels 1–7 (e.g., EAL4 requires methodical design testing).

Example: A firewall ST might reference a PP to ensure it meets standardized security objectives.

d. SSE-CMM (1997–Present)
Focus: Evaluates an organization’s security engineering process (not the system itself).

Maturity Levels:

Level 1 (Informal): Base processes are performed.

Level 5 (Continuously Improving): Processes are optimized.

Example: Assessing an organization’s "Threat Identification" process to determine if threats are systematically monitored.

3. Key Contributions and Limitations
TCSEC: Pioneered evaluation but was rigid and limited to U.S. government needs.

FIPS 140: Specialized for cryptography, with measurable improvements in module security.

Common Criteria: Flexible, international, and adaptable to diverse products (e.g., firewalls, databases).

SSE-CMM: Focuses on organizational maturity (e.g., how well a company manages security engineering).

4. Conclusion
The evolution of evaluation methodologies—from TCSEC to Common Criteria and SSE-CMM—reflects the growing complexity of security needs. Each framework addresses specific gaps:

TCSEC: Confidentiality in government systems.

FIPS 140: Cryptographic module integrity.

Common Criteria: Broad, international product evaluations.

SSE-CMM: Organizational security practices.

Example: A government agency might use TCSEC for OS evaluation, while a commercial vendor adopts Common Criteria for a globally recognized certification. Meanwhile, a software firm uses SSE-CMM to improve its development lifecycle.