

Malicious Logic - Overview

Malicious logic refers to code or software designed to violate a system's security policy.

🔗 Types of Malicious Logic

1. Trojan Horse

Definition: A program with a legitimate (overt) function and a hidden (covert) malicious function.

Example: A shell script that looks like an `ls` command but creates a `setuid` shell, giving the attacker elevated access.

Real-world: NetBus—disguised as a game, gives remote access to Windows NT systems.

📁 Replicating Trojans

Propagates itself automatically.

Example: Ken Thompson's compiler Trojan, which inserts a backdoor during compilation.

2. Computer Viruses

Definition: Code that inserts itself into other programs/files, often to replicate or cause harm.

Phases: Insertion & Execution

Examples:

Lehigh Virus - inserts into boot files if uninfected.

Jerusalem Virus - reinfects .EXE files endlessly due to a bug.

📖 History

Fred Cohen formally described viruses.

Early experiments on UNIX and UNIVAC systems.

🦠 Types of Viruses

Boot Sector Infectors: Infects boot sector (e.g., Brain virus).

Executable Infectors: Infects .EXE/.COM files.

Multipartite Viruses: Infects both boot & executable.

TSR Viruses: Remains in memory post execution (Terminate and Stay Resident).

Stealth Viruses: Conceals presence (e.g., hides size or cleans file before scan).

Encrypted Viruses: Enciphered code with a decryption routine to evade signature detection.

Polymorphic Viruses: Changes appearance each infection to avoid detection.

Example: Randomly uses different but equivalent instructions like xor 0, add 0, etc.

Macro Viruses: Use interpreted macros (e.g., Melissa virus in MS Word).

3. Computer Worms

Definition: Self-replicating programs that spread across computers.

Examples:

Internet Worm (1988) – Took down thousands of UNIX systems.

Christmas Worm – IBM networks; sent itself via email after drawing a tree.

4. Other Malicious Logic Types

Rabbits/Bacteria: Consume all system resources.

Example: Infinite loop creating directories in UNIX.

Logic Bombs: Hidden code triggered by specific conditions.

Example: Deletes payroll data when a certain employee is removed.

🛡 Defenses Against Malicious Logic

1. Distinguish Data and Instructions

Use systems like LOCK to treat data and executable as different types.

Prevent automatic conversion without certification.

2. Limit Accessibility

Restrict file access and object manipulation by processes.

💡 Example: UNIX execute/read permissions

Changing a file to “data” type after modification.

Only certifier can revert to “executable”.

3. Information Flow Metrics

Restrict spread based on distance (fd) a virus can travel.

If a file's fd exceeds a threshold V, access is denied.

4. Reducing Protection Domain

Apply least privilege principle.

Use Access Control Lists (ACLs) or Capability Lists (C-Lists).

💡 Example:

If user A runs program P that shouldn't write to file F owned by user B, system should block that unless explicitly allowed.

5. Authorization Denial Subset

Define operations that no one else can perform on your files.

6. Karger's Scheme / Lai & Gray's VAL System

Trusted vs. Untrusted processes.

Allow file access only if it's listed in a Valid Access List (VAL).

💡 Example:

Assembler creates /tmp files, which are auto-added to VAL. Trojan horse can't access other files.

7. Guardians/Watchdogs

Programs intercept file access and decide based on policy.

8. Trust Management

Trust user actions, system mechanisms, and specific programs.

9. Sandboxing

Restrict program behavior using traps or virtual environments.

Monitor for race conditions (e.g., file name changed between system calls).

10. Inhibit Sharing

Using integrity policies like *-property and ss-property.

Example: DG/UX system places all user code below virus protection level.

11. Detect File Alterations

Use cryptographic checksums or MDCs (e.g., Tripwire).

12. Proof-Carrying Code

Code comes with a proof that it is safe.

Modified code invalidates the proof.

13. Statistical Detection

Analyze coding patterns for anomalies (e.g., new author style or excessive conditionals).

14. N-Version Programming

Run multiple implementations simultaneously and check for consistency.

1. What is a Vulnerability?

A vulnerability is a failure in a system's security controls, policies, or implementation that allows unauthorized access or actions.

When an attacker takes advantage of this failure, it's called exploitation.

🔍 2. Penetration Testing / Studies

Used to find vulnerabilities by simulating attacks.

✳️ Flaw Hypothesis Methodology (5 Steps):

Information Gathering - Understand system architecture.

Flaw Hypothesis - Predict potential vulnerabilities.

Flaw Testing - Attempt to exploit predicted flaws.

Flaw Generalization - Find similar vulnerabilities.

Flaw Elimination (optional) - Fix the flaws.

🔧 3. Examples of Vulnerabilities (Case Studies)

📍 Michigan Terminal System

Goal: Modify memory segment controlling privileges.

Exploit: Use parameter list trick to write to privileged memory.

Result: Full control of system due to improper validation.

📍 Burroughs B6700

Exploit: Off-line editing of tape headers to convert programs into privileged compilers.

Result: Unauthorized privilege escalation via manipulation of file types.

📍 Corporate System

Goal: Social engineering test.

Methods: Impersonation via phone, directory access, password phishing.

Result: Gained access using user credentials and modem numbers.

📍 UNIX System with Sendmail

Exploit: Used "wiz" command in Sendmail 3.1 to gain root access.

Result: System compromise with no password required.

📍 loadmodule (UNIX)

Exploit: Modified environment variables to trick privileged programs using system(3).

Result: Executed malicious ld.so as root by exploiting environment inheritance.

📍 Windows NT

Exploit: Admin used weak password "Admin".

Result: Used domain privileges to take over systems.

📄 4. Vulnerability Classification Frameworks

📊 RISOS

Categories:

Incomplete Parameter Validation

Inconsistent Validation

Implicit Sharing

Asynchronous Validation

Inadequate Identification

Violable Limits

Logic Errors

✓ Examples:

xterm flaw → Asynchronous validation (race condition)

fingerd flaw → Incomplete parameter validation (buffer overflow)

⚙️ Program Analysis (PA)

Focuses on coding-level issues:

Improper Initialization

Improper Isolation

Improper Change

Improper Naming

Improper Deletion

Improper Validation

Improper Sequencing

Improper Operand Choice

✓ Examples:

xterm flaw → Improper change

fingerd flaw → Improper validation

🧠 NRL Taxonomy

Focus on when, how, and where flaws occur.

Genesis (intentional/unintentional)

Time (development, maintenance, operation)

Location (privileged utilities, kernel, etc.)

✓ Examples:

xterm: Development phase, location = privileged utilities

fingerd: Development phase, same location

🌳 Aslam's Model

Fault tree-based classification (binary decision model).

Categories:

Coding Faults (e.g., xterm, fingerd)

Emergent Faults (e.g., misconfigurations)

✓ Examples:

xterm: Synchronization error (race condition)

fingerd: Condition validation error (input not checked)

🔑 5. Famous Vulnerability Examples

📁 xterm Flaw:

Log file created using name given by user.

Attack: Use race condition to change file binding between validation and opening.

Vulnerability: TOCTTOU (Time-of-check to time-of-use).

📁 fingerd Flaw:

Internet worm exploited overflow in input buffer.

Injected shell code via large input.

Vulnerability: Buffer overflow + stack manipulation.

📊 6. Comparison Across Frameworks

Flaw	RISOS	PA	NRL
Aslam			

xterm	Asynchronous validation	Improper change	Dev time / Privileged util.
	Synchronization error		
fingerd	Incomplete validation	Improper validation	Dev time / Privileged util.
	Boundary condition		

1. What is Auditing?

☒ Logging

The act of recording system events or statistics (e.g., logins, file access).

☒ Auditing

Analyzing logs to understand what happened, detect policy violations, or monitor system use.

🔗 2. Components of an Audit System

Logger – Records events.

Analyzer – Scans logs for specific patterns or anomalies.

Notifier – Informs stakeholders or takes action based on the analysis.

🔗 3. Example Audit Systems

☒ RACF (IBM MVS/VM)

Logs: failed access, privilege changes.

Command: LISTUSERS used to view entries.

☒ Windows NT

Logs: system, application, and security events.

Tool: Event Viewer.

Example: Process creation by Internet Explorer is logged with timestamps, users, process ID, etc.

🔍 4. Auditing in Practice

☒ Examples of Auditing:

Telnet Detection: Use swatch to monitor logs for external telnets.

Intrusion Detection Systems: Use sensors to monitor suspicious activity.

☒ 5. Designing an Audit System

☒ Goal-Driven Logging

Determine what to log based on security policy.

Example: In Bell-LaPadula:

Log: subject level, object level, action (read/write), result.

? Challenges

Ambiguities in log format.

Multiple names for the same object.

Over-logging vs. under-logging.

🔒 6. Log Sanitization

🔒 Why?

To protect privacy or classified information when sharing logs.

Types:

Anonymizing: Cannot be reversed.

Pseudonymizing: Reversible using mapping or encryption.

🧠 Example: IP sweep detection

Sequential IP sanitization keeps pattern (e.g., attack on IP range).

Random IP sanitization loses pattern, making analysis harder.

📦 7. Application vs. System Logging

Feature	Application Logging	System Logging
Focus	High-level events (e.g., login)	Low-level system calls (e.g., exec)
Detail	Abstract	Technical
Size	Small	Large
Example	su: bishop to root	CALL execve(...)

📋 8. Audit Types

📋 State-Based

Check system state (e.g., file permissions).

Requires consistent snapshot of the system.

📋 Transition-Based

Monitor actions or changes (e.g., TCP connection attempts).

🧠 Example: TCP Land Attack

Malicious packet with same source/destination IP and port.

Auditing must log IPs and ports to detect such packets.


🔒 9. Secure Audit Systems

🔒 Example: VAX VMM

Integrated auditing in OS design.

Reliable, layered kernel logging.

Log Criteria: Subject, object, severity, event, timestamp.

 Example: Compartmented Mode Workstation (CMW)

Commands: audit_on, audit_write, audit_suspend, audit_resume, audit_off

Logs based on user-defined control.


Binary logs processed via redux.


 10. Non-Secure Systems

 Example: Basic Security Module (BSM - SunOS/Solaris)

Logs: Binary records made of tokens (user info, file, system calls, etc.).

Predefined event classes determine what gets logged.

 11. File System Auditing

 Network File System (NFS) v2

Stateless; server unaware of who accessed files.

Logging needed for:


File handle creation

User & host info

File attributes

Results

File names

 LAFS (Logging and Auditing File System)


Built on NFS with user-level policies.

Adds:

file%log: access history

file%policy: access policy


file%audit: compare logs to policy

 Example Policy

prohibit:0900-1700:*:*:wumpus:exec

→ Block execution of 'wumpus' from 9 AM to 5 PM

 12. Audit Browsing Techniques

 Browsing Modes:

Text Display: Basic but lacks relationships.

Hypertext: Local relationships.

Relational DB: Can query complex relationships.

Replay: Timeline of events.

Graphing: Visual entity-relationship.

Slicing: Filter to focus on specific objects.

🔍 Example Tools

Frame Visualizer: Shows graphical structure.

Movie Maker: Shows events in sequence.

Hypertext Generator: Creates pages per user/file.

Focused Audit Browser: Visual drill-down on node activity.

🔍 13. Visual Audit Use Case

🔧 Scenario:

A file was altered.

Use focused browser to see which process changed it.

Trace the attacker via hypertext logs.

Replay events to show law enforcement or management.

🧐 Mielog Tool

Visual + text-based log analyzer.

Flags abnormal patterns using color-coded tags.

1. Principles of Intrusion Detection

A system not under attack:

Follows statistically predictable user/process behavior.

Avoids sequences violating security policy.

Adheres to program specifications.

⚠️ A system under attack violates one or more of the above.

★ 2. Example: Rootkit Attack

Rootkit: Toolset to maintain stealthy access.

Replaces system utilities (ps, ls, find, etc.) with Trojan versions.

Hides backdoors and files.

Detection: Compare output of `ls` and `dirdump`; discrepancies reveal tampering.

🔗 3. Denning's Model

Hypothesis: Abuse = abnormal use of normal commands.

Uses statistical modeling and deviation from expected behavior.

🔗 4. Goals of an Intrusion Detection System (IDS)

Detect a wide variety of attacks (known & unknown).

Do it in real-time, if possible.

Provide clear analysis (e.g., dashboards, alerts).

Maintain accuracy: low false positives & negatives.

🔗 5. IDS Detection Models

Model	Idea	Pros
Cons		
Anomaly Detection	Unusual = bad	Detects unknown attacks
High false positives		
Misuse Detection	Known bad = bad	Accurate on known attacks
Cannot detect unknown		
Specification-based	Not allowed = bad	Precise for defined programs
Requires detailed specs		

📊 6. Anomaly Detection Techniques

📊 6.1 Threshold Metrics

If count of events (e.g., failed logins) exceeds a threshold → alert.

📊 6.2 Statistical Moments

Uses mean, deviation, and correlations.

Weights recent behavior more heavily (e.g., IDES system).

📊 6.3 Markov Models

Looks at sequences of events, not isolated actions.

Requires training with normal data.

📊 6.4 Example: TIM

Learns sequences like `ab → c`.

If `abd` appears, it raises an alert since `d` is unexpected.

🔍 7. Misuse Detection

Matches rule sets of known attacks.

Fails to detect new/unknown exploits.

🔗 Example: NFR

Uses filters written in a custom language (n-code) to specify what traffic to watch.

🔗 8. Specification-Based Detection

Programs are allowed to perform only specified behaviors.

Any deviation → possible attack.

📄 Example: rdist Attack

Creates temp file → attacker swaps it with a symlink → rdist changes real system file.

Detection: chown/chmod actions must match created files only.

🔗 9. IDS Architecture

Components:

Agent: Collects data (e.g., from logs or traffic).

Director: Analyzes data for suspicious patterns.

Notifier: Alerts security personnel or triggers action.

🔗 10. Types of Agents

Type	Description
Host-based	Analyzes OS logs, system events.
Network-based	Monitors packets, traffic patterns.
Application-based	Focuses on app behavior and logs.

🔗 11. Director Functions

Aggregates and correlates info from agents.

Removes noise, redundant data.

Can be adaptive, using machine learning (e.g., neural networks).

📄 12. IDS Implementations

🔗 NSM (Network Security Monitor)

3D matrix: source, destination, service.

Compares observed vs expected traffic profiles.

🔗 DIDS (Distributed IDS)

Combines host + network data.

Uses NIDs (Network IDs) to track user activity across systems.

Expert system scores and flags intrusions.

👤 AAFID (Autonomous Agents)

Each agent monitors a specific resource.

Distributed detection; avoids single point of failure.

Agents communicate and collaborate.

🚨 13. Incident Handling Lifecycle

Preparation – Plans, tools, configurations.

Identification – Recognize attack signatures/anomalies.

Containment – Limit attack scope.

Eradication – Remove attacker, kill processes.

Recovery – Restore systems.

Follow-up – Analyze, improve defenses.

👤 14. Containment & Deception

🏠 Jails

Isolate attackers in fake environments.

👤 Deception Toolkits

Mimic vulnerable systems.

Lure attackers to waste time, while defenders learn about their goals.

🔒 15. Eradication Techniques

Use wrappers to intercept system calls and deny access.

Log and kill connections dynamically.

Employ firewalls and content filters (e.g., block Java .class files).

🔒 16. IDIP Protocol

Intrusion Detection and Isolation Protocol

Coordinates response across firewalls and boundary systems.

Alerts and blocks malicious traffic across networks.

⚔️ 17. Counterattacking (Controversial)

Legal: Requires proper evidence (log chains, timestamps).

Technical: Launch counterattacks like counterworms.

⚠ Issues:

Might hit innocent parties.

Legally risky.

May worsen network load.

1. Introduction: The Drib

📁 Company Profile

Builds and sells "dribbles" (products).

Needs internet connectivity (email, web, etc.).

Facing a hostile takeover, so corporate data must be secure.

🔒 2. Security Goals & Policy Development

🎯 Main Goals:

Protect sensitive data.

Ensure 99% availability of systems.

Support secure internal/external communication.

🏢 Organizational Groups:

Customer Service Group (CSG): Handles customer data.

Development Group (DG): Builds products.

Corporate Group (CG): Legal, patents, management.

📁 3. Data & User Classifications

📁 Data Types:

PD (Public Data) – everyone can read.

DDEP (Existing Product Data) – CG, DG access.

DDFP (Future Product Data) – DG access only.

CpD (Corporate Data) – CG only.

CuD (Customer Data) – CSG only.

👤 User Types:

O (Outsiders) – public users.

D (Developers) – access DDFP, DDEP.

C (Corporate Executives) – access all except DDFP write.

E (Employees) – customer service.

Access Control Matrix

Defines read (r) and write (w) permissions for all user/data class combinations.

4. Policy Type & Reclassification

Mandatory Policy: Users can't elevate others' access.

Discretionary: Group-internal sharing allowed.

Reclassification Rules:

DDFP → DDEP: needs C & D approval.

DDEP → PD: needs C & E approval.

CpD → PD: requires two C members (separation of privilege).

5. Consistency Checks

Each goal aligns with the access matrix:

Developers can't access customer/corporate data.

Only CSG and customers can change customer data.

Only executives can reclassify data → separation of privilege enforced.

6. Network Organization

Subnetting:

Internal network divided into subnets (CG, DG, CSG).

Firewalls placed between each for segmentation and containment.

DMZ (Demilitarized Zone):

Buffer zone between internal network and Internet.

Contains public-facing servers (web, email, DNS, log).

If compromised, doesn't affect internal systems.

7. Firewall Design & Proxying

Outer Firewall (Internet to DMZ)

Allows SMTP, HTTP/HTTPS.

Proxy-based filtering and malware scanning.

Inner Firewall (DMZ to Internal)

Enforces fail-safe default: blocks all unless explicitly allowed.

Manages internal email routing and secure SSH.

8. Server Design in DMZ

📧 DMZ Mail Server

Sanitizes headers/content, hides internal structure.

Uses public key encryption before transferring order data internally.

🌐 DMZ Web Server

No confidential data stored.

CGI scripts hardened.

Web content updated from internal WWW-clone.

📁 DMZ Log Server

Stores logs offline and on write-once media.

Runs SSH; used for audit and forensic analysis.

🔒 9. Application of Security Principles

Principle	Implementation
Least Privilege	Servers only know what they need
Complete Mediation	Every access mediated by firewalls
Separation of Privilege	Two-step approvals for reclassification
Least Common Mechanism	Services separated, DMZ DNS is a minor exception

🛡️ 10. Defense in Depth

Even if outer firewall fails, inner firewall compensates.

Public IPs masked using NAT.

Orders are encrypted before leaving DMZ.

🛡️ 11. Attack Handling & Intrusion Detection

📦 SYN Flood Attack:

Intermediate Hosts: TCP intercept, SYN cookies.

Endpoint Defense: adaptive timeouts, Linux-style SYN cookie validation.

🔒 IDS Tactics:

DMZ log server runs anomaly & misuse detection.

Successful and suspicious behavior gets priority over generic failures.

📦 12. Updating & Maintenance

Web updates happen on internal clone.

SSH from a trusted internal admin host is the only method for DMZ maintenance.

📦 13. Internet Ordering Flow

Customer fills order via DMZ web server.

Order is encrypted using internal system's public key.

Web server places file in spool.

Internal trusted admin retrieves the file.

🔒 14. Final Assumptions & Summary

Assumes hardware/software works as intended.

Emphasis on assurance and policy-driven design.

Logs and IDS tuned regularly to reduce alert fatigue.

1. Introduction

Focuses on administering system security.

Case studies:

DMZ Web Server: Highly restricted, public-facing.

Devnet Workstation: Used by developers internally.

📄 2. Security Policies

🔒 DMZ Web Server Policy

Accepts only HTTP/HTTPS from outer firewall and SSH from a trusted admin host.

Logs to DMZ log server only.

Runs CGI scripts but limits control.

Public keys and minimal software on server.

Highly locked-down system.

📁 Devnet Workstation Policy

Only authorized developers can access.

Admins have access anytime.

Encrypted, authenticated communication.

Users can't change base config.

Regular audits and backups.

Workstations store only transient data.

🔒 3. Network-Level Security

🔒 Firewalls

Both systems rely on inner/outer firewalls.

DMZ replicates firewall rules for defense in depth.

🔑 DMZ Web Server Config

apache:

```
order allow, deny
allow from outer_firewall
allow from inner_firewall
deny from all
SSH allows only trusted admin host.
```

🔑 Devnet Workstation

Uses TCP wrappers to control access to services.

Centralized FTP/Web servers, remote-mounted directories to minimize services.

🔍 4. Availability

DMZ Web Server auto-restarts using a script if it crashes.

👤 5. User Management

👤 DMZ Web Server

Users:

webbie: reads/writes web content.

ecommie: handles encrypted transactions.

Admins use personal accounts (never log in as root remotely).

👤 Devnet Workstation

Each developer has a unique account.

Uses UINFO/NIS system for central account management.

Ensures identity consistency but introduces naming collisions risk.

🔑 6. Authentication

System	Method
DMZ Web Server (PAM)	SSH with public key, MD5-hashed passwords, smart cards
Devnet Workstation password aging	SSH (no remote root), DES-hashed passwords (8-char max),

⚙️ 7. Processes

🖨️ DMZ Web Server

Minimal processes:

Web server

Commerce server

SSH, Login

OS background services

Uses wrappers for privilege reduction (bind low ports securely).

 Devnet Workstation

More processes for developer tasks.

Each server runs as nobody:nogroup.

Logs system calls and usage for IDS.

 8. File Access & Storage

 DMZ Web Server

Base system on CD-ROM (immutable).

Only web pages are writable (on hard drive).

CGI scripts & keys on CD-ROM.


If compromised: reboot → reformat → reload from internal WWW-clone.

 Devnet Workstation

Base config from bootable CD-ROM.

Logs and transient data on hard disk.

Frequent IDS scans; any breach = reformat.

 9. Logging & Intrusion Detection

Both systems log to central servers.

DMZ logs directly to DMZ log server, which uses IDS for anomaly/misuse detection.


Devnet logs analyzed and checked against IDS monthly.

 10. Interprocess Communication

Web → Commerce server via:

Shared directory (trnsnnnn files).

Optional: UNIX signal for triggering.

 11. Security Summary

Aspect
Use

DMZ Web Server
Public-facing, minimal

Devnet Workstation
Developer use, general-purpose

Users	Very limited	Many, trusted
Processes	Few, essential	More, varied
Auth	Public key + SSH	SSH, local login
Storage	CD-ROM + minimal disk	CD-ROM + transient disk
Logging	Direct to DMZ log	To central log server
Recovery	Reboot, wipe, reload	Reboot, wipe, reimage
☑ 12. Key Security Principles Applied		
Least Privilege: Limit permissions (e.g., minimal user accounts, directory access).		

Separation of Privilege: Admins have tiered responsibilities.

Complete Mediation: All actions tied to authenticated users.

Economy of Mechanism: Simple systems are more secure.

Defense in Depth: Multiple firewalls, logging, immutable storage.

1. Policy Overview

🔑 User Security Policies (U1-U4):

U1: Only account owners access their accounts.

U2: Others cannot access/modify a user's files without permission.

U3: Users must protect their files' integrity, confidentiality, and availability.

U4: Users must be aware of all commands executed by or on their behalf.

🔒 2. Account Access Control

🔑 Passwords:

Random passwords are hard to remember → often written down.

Danger depends on how and where they're stored.

🏠 Isolated System:

Access only in secure, locked room. Passwords written on whiteboard (safe due to physical access restrictions).

🖨 Multiple Systems:

Centralized user DB for non-infrastructure systems.

Infrastructure systems may have separate or transformed passwords.

🧠 Memorized Algorithms (for admins):

Admins (e.g., Anne & Paul) use transformations to remember complex passwords.

🔒 3. Login Procedures

⚠ Risks:

Trojan login prompts

Shoulder surfing

Lack of mutual authentication

🛡️ Defenses:

Trusted path (special keys trigger trusted login).

Encrypted login traffic (SSH, SSL).

Display last login time/location.

Limit trusted hosts; Drib uses cryptographic challenge-response.

👤 4. Session Control

Users must lock systems when away (e.g., xlock).

Drib enforces policies—failure = disciplinary action.

No modems allowed on devnet → protects against remote hijacking.

📁 5. File & Device Access

📁 File Protections:

ACLs and umask used to deny group/other access by default.

Group roles allow shared access for project teams.

Deletion wipes sensitive files (secure delete).

🗑️ File Examples:

Deleting a file only removes a reference—if other users have aliases (links), the file remains.

Must revoke permissions before deletion.

📁 Devices:

Writeable devices (tapes, terminals) are tightly controlled.

Smart terminals disabled unless all sequences shown as printable.

🖥️ Window Systems:

X Window uses xhost (hostname-based) or xauth (token-based) for control.

Transparent windows can be used maliciously to log inputs (e.g., passwords).

🗑️ 6. Process Awareness (U3)

🔥 Key Areas:

Copy/Move Semantics – May change permissions.

Overwriting by mistake – e.g., `rm * .o` error.

Encryption Trust - Users must trust software and OS (memory leaks).

Start-Up Settings - Insecure files may lead to malicious behavior.

Privilege Limitation - Avoid setuid programs without strong justification.

Malicious Logic - Avoid risky search paths (Trojan ls example).

🔒 7. Electronic Communications (U4)

⚠️ Key Risks:

Auto-execution of attachments (e.g., NIMDA worm).

Invalid certificates accepted by default.

Hidden/confidential data leaks via spreadsheet/software behavior.

🛡️ Mitigations:

Drib disables auto-execution of email content.

Enhanced certificate checks in mail apps.

Educated users on dangers of “deleted” content still present in files.

☑️ 8. Summary of Security Measures

Area	Strategy
Passwords	Central DB, algorithm-based memory, proactive checkers
Login	Trusted path, SSH, certificate validation
Leaving system	Screen lock, physical security
Files	ACLs, group controls, secure deletion
Devices	Writable access limited; no smart terminals
Processes	Copy/move safety, avoid setuid, startup file control
Email	No auto-run, strict cert checking, no hidden data leaks

1. Introduction to Program Security

🎯 Goal:

To implement a secure role-switching utility (like UNIX su) that:

Authenticates user identity

Validates access based on user, location, time

Grants access to role account securely

📄 2. Requirements & Policy

☑️ Key Requirements:

Role access based on user, location, and time.

Role environment replaces user's relevant environment.

Only root can alter role access control data.

Supports restricted (specific commands) and unrestricted (interactive shell) access.

Role file access is limited to authorized users or root.


3. Threats & Countermeasures

Group 1: Unauthorized Users (UU)

Impersonation, insecure channels, modifying access files, Trojan horses.

Group 2: Authorized Users (AU)

Abuse of permissions, unauthorized commands, altering their own access levels.

 Requirements mitigate threats via:


Trusted access points

Time/location filters

Logging and command restrictions

Environment isolation

4. System Design Overview

 Interface:

CLI format:

role [role_account] [command]

 High-Level Steps:

Collect user, time, location, command.

Verify access.

Change privilege to role.

Execute specific command or shell.

5. Access Control Module

Function:

c

Copy

Edit

int accessok(uid_t rname, char *cmd[]);

Uses access control records (role, user list, location, time, commands).

Time is represented in human-readable intervals.

Supports command wildcards (e.g., /bin/install *).

🔧 6. Implementation Details

🔍 Steps:

Obtain user ID (getuid()), time (time(NULL)), location (via utmp).

Read access control file securely.

Match entries.

Close file and return access decision.

👉 Error Handling:

All parsing/IO errors are logged.

Errors result in access denial.

⚠️ 7. Security Programming Concerns

Chosen Framework: Program Analysis (PA)

🔧 8. Common Programming Problems

A. Improper Protection Domain

Apply least privilege.

Use modular design to isolate privileged operations.

B. Improper Memory Sharing

Disable shared memory and dynamic loading.

Mark memory as non-executable, read-only where needed.

C. Improper Validation

Check types, bounds, values, and all return codes.

Examples: prevent buffer overflows, validate time formats, avoid format string bugs.

D. Improper Indivisibility

Prevent TOCTTOU (Time-of-check-to-time-of-use) race conditions.

E. Improper Naming

Use absolute paths.

Avoid reliance on environment variables or search paths.

🔪 9. Secure Deallocation

Sensitive objects (passwords, control files) are wiped before deletion.

Clear memory of passwords after use.

🔧 10. Validation by Design

Inputs checked thoroughly (especially from users).

Command execution limited to validated patterns.

Example: prevent execution of unauthorized shell commands.

⚙️ 11. Testing

🔍 Testing Types:

Normal, boundary, exception, random, and composed module testing.

🔗 Best Practices:

Independently test installation & configuration.

Document every assumption and behavior.

🚚 12. Distribution & Integrity

📦 Key Practices:

Distribute through secure, controlled channels (e.g., CD-ROM, HTTPS).

Protect master copies (e.g., signed binaries).

Warn of past examples (e.g., backdoored tcp_wrappers case).

🧠 13. Key Principles & Rules Summary

☑️ Implementation & Management Rules:

Keep privileged code minimal and modular.

Validate all program assumptions.

Enforce memory protection.

Isolate trusted/untrusted data.

Check error returns thoroughly.

Avoid dynamic loading unless absolutely necessary.

Ensure objects are properly named and managed.

Sanitize environments and inputs.