1. Malicious Logic
Definition:
Malicious logic is code or instructions designed to violate a system's security policy.

Types of Malicious Logic:

Trojan Horses: Appear to do something legitimate but perform hidden malicious tasks.

Example: A fake “ls” command that grants attacker a shell with the user's privileges.

NetBus: A Windows-based Trojan that opens backdoors for remote control.

Replicating Trojan Horse: Copies itself to other programs or systems.

Example: Thompson’s compiler that inserts a Trojan into login programs undetectably.

Computer Viruses: Code that inserts itself into programs and spreads.

Types:

Boot Sector Infectors (e.g., Brain Virus)

Executable Infectors (e.g., Jerusalem)

Multipartite Viruses (infect both boot & executable)

TSR Viruses (stay resident in memory)

Stealth Viruses (hide infection signs)

Encrypted/Polymorphic Viruses (change form to evade detection)

Macro Viruses (embedded in documents; e.g., Melissa)

Worms: Self-replicating programs that spread over networks.

Examples: Internet Worm (1988), Christmas Worm

Logic Bombs: Trigger malicious actions based on events.

Example: Deletes payroll when an employee's record is removed.

Rabbits/Bacteria: Consume resources without doing anything productive.

Example: Infinite directory creation loop in UNIX.

Defenses Against Malicious Logic:

Distinguish between data & instructions

Restrict access and reduce sharing

Use trusted compilers (LOCK system, Duff’s executable control)

Information flow metrics to slow virus spread

Use Access Control Lists (ACLs), Capability Lists

Apply the principle of least privilege

2. Vulnerability Analysis
Definition:
A vulnerability is a flaw that allows attackers to violate security policies.

Techniques:

Formal Verification: Mathematically proving system correctness.

Penetration Testing: Trying to break in to find real-world vulnerabilities.

Flaw Hypothesis Methodology: 5-step approach:

Information Gathering

Hypothesize Flaws

Test Hypotheses

Generalize Flaws

(Optional) Eliminate Flaws

Examples of Exploits:

Michigan Terminal System: Exploited parameter passing in system calls to overwrite memory.

Burroughs B6700: Manipulated tape headers to give users compiler privileges.

Corporate System: Social engineering used to collect credentials from new employees.

Sendmail ‘wiz’ Exploit: Remote shell gained via legacy vulnerability.

Loadmodule Exploit: Gained root by manipulating environment variables and using symbolic links.

Windows NT Attack: Easy password guessed; used to escalate to domain admin via service account.

Vulnerability Classification Frameworks:

RISOS, PA, NRL, Aslam’s Model: Provide structured ways to categorize flaws by origin, exploit technique, or impact.

3. Auditing
Definition:
Auditing involves logging events and analyzing those logs to detect and understand violations.

Components:

Logger: Records events.

Analyzer: Reviews logs to detect anomalies.

Notifier: Alerts administrators or takes actions.

Audit Systems:

Example (RACF, IBM): Logs failed access, privilege escalations.

Windows NT: Logs security, system, and application events.

Designing Audit Systems:

Must log based on defined security policy violations.

Choose between:

State-Based Auditing: Snapshot system state.

Transition-Based Auditing: Log events that change the state.

Issues:

Log Format: Needs clear syntax.

Sanitization: Hide sensitive info in logs (pseudonymization, anonymization).

Correlation: Match application-level logs with system-level events.

Examples:

Bell-LaPadula Enforcement: Logs read/write actions with security labels.

LAND Attack Detection: Identify malformed TCP packets where source equals destination.

Audit System Examples:

VAX VMM: Logs security-critical kernel events.

CMW: Audits multi-level security events with user-controlled settings.

Basic Security Module (BSM): Token-based, logs system and application actions in Solaris.

4. Intrusion Detection
Principles:

Systems not under attack follow predictable behavior and process actions.

Intrusions are deviations from this behavior or policy.

Detection Models:

Anomaly Detection: Unusual behavior = attack.

Techniques:

Thresholds (e.g., login failures)

Statistical moments

Markov Models (sequential patterns)

Example: IDES system using weighted user activity statistics.

Misuse Detection: Known bad behavior matched against signatures.

Example: NFR with N-code filters for network packets.

Specification-Based: Define what's allowed; deviations are flagged.

Example: Monitoring rdist program operations against a defined grammar.

IDS Architecture:

Agents: Collect data (host-based or network-based).

Director: Analyzes data.

Notifier: Takes action or alerts admins.

Example IDS Systems:

NSM: Network traffic matrix analysis.

DIDS: Combines host and network monitoring; uses expert systems.

AAFID: Autonomous agents for IDS.

Incident Response:

Alerting, logging, shutting down sessions, notifying administrators.

Visualization tools (e.g., GrIDS GUI) for worm detection.

Unit 4.5 – Network Security
1. Introduction & The Drib Case
The "Drib" company case is used to contextualize security design.

It needs to support e-commerce, internal development, and legal needs while protecting sensitive info.

2. Security Policy
Divides data into classes: Public, Development (current/future), Corporate, Customer.

User classes: Outsiders, Developers, Corporate Executives, Employees.

Uses an Access Control Matrix to define what each user can do.

3. Data Reclassification
Data can move between classes based on status changes (e.g., future to current dev data).

Requires multiple approvals to ensure separation of privilege.

4. Network Architecture
Network is segmented with DMZ (demilitarized zone) between internal and external.

Use of firewalls:

Packet filtering firewalls check ports/IPs.

Proxy firewalls inspect content.

5. Key Mechanisms
Firewalls, DMZ, Proxies ensure that internal systems don’t talk directly to the Internet.

Security pump: Only way for public-facing systems to send/receive data.

Unit 4.6 — System Security
1. Two Systems Compared
DMZ Web Server: Public-facing, minimal access, high assurance.

Development System: Used by employees, secured via internal policies.

2. Policies for Each System
DMZ Web Server Policies:

Accepts only HTTP/HTTPS and SSH from specific hosts.

Runs restricted services like CGI scripts.

Extensive logging and minimal software.

Development System Policies:

Central control, backups, consistent config, encrypted comms.

Only admins can modify base configurations.

3. Consequences
Emphasis on logging, auditing, system isolation, and controlled access.

User identity tracking and role-based privileges enforced at system and network level.

Unit 4.7 — User Security
1. Policy Principles
U1: Only the owner can access the account.

U2: Others can't read/write files unless allowed.

U3: Protect confidentiality and integrity of files.

U4: Be aware of commands executed.

2. Access Controls
Password strategies:

Password transformations for memorization.

Proactive password checking.

Login security:

Trusted path: authentic login interface.

Shoulder surfing, Trojan login programs, and network sniffing covered.

3. File Security
Use of groups, ACLs, umask to manage file permissions.

File deletion: Files may still exist via aliases (hard links) even after rm.

4. Smart Terminals & Devices
Trojan horse via smart terminals (e.g., email that changes file permissions).

Writable devices (e.g., tapes) need strict access control.

5. Window Systems
X Windows: Access control via hostnames or authentication cookies.

Attackers may overlay fake windows to capture input.

Unit 4.8 — Program Security
1. Objective
Create a secure tool like su, allowing users to switch roles without sharing passwords.

Program uses access control rules: based on user, location, time, command.

2. Design Requirements
Access control records contain role, users, times, commands.

Root must be the only entity that can modify these records.

3. Threats
Unauthorized users (UU) may try to gain access via sniffing or trojans.

Authorized users (AU) may abuse unrestricted access.

4. Implementation Details
Modules collect user info (user ID, time, location).

Uses a function accessok() to check access permissions.

Interface design: Pass minimal info; rest retrieved from the system.

5. Security Programming Problems
Based on the PA framework:

Improper privilege assignment: follows least privilege.

Memory protections: no shared memory; constants marked const.

Race conditions (TOCTTOU): validate access after file is opened.

Input validation: strict checks on user input (e.g., against format string attacks).

6. Best Practices & Rules

Implementation and management rules emphasize:

Validation of inputs

Isolation of privileged modules

Error handling

Documenting trade-offs

Testing in realistic environments

7. Example

Prevented format string exploit: printf(str) where str = "log%n" might overwrite memory.

Solution: Separate trusted/untrusted memory zones.