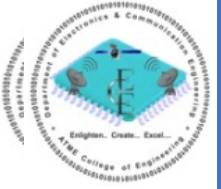




A T M E
College of Engineering



BEC714B

COMPUTER & NETWORK SECURITY

Course Objectives:

- **Preparation:** To prepare students with fundamental knowledge/ overview in the field of Network Security with knowledge of security mechanisms and services.
- **Core Competence:** To equip students with a basic foundation of Network Security by delivering the basics of Transport Level Security, Secure Socket Layer, Internet Protocol security, Intruders, Intrusion detection and Malicious Software, Firewalls, Firewall characteristics, Biasing and Configuration

Module-1

Attacks on Computers and Computer Security: Need for Security, Security Approaches, Principles of Security Types of Attacks. (Text2: Chapter1)
Security Mechanisms, Services and Attacks, A model for Network security (Text1: Chapter1: 3, 4, 5, 6)

Module-2

Malicious Logic: Introduction, Trojan Horses, Computer Viruses, Computer Worms, Other Forms of Malicious Logic, Defenses (Text 3: Chapter 12)
Vulnerability Analysis: Introduction, Penetration Studies, Vulnerability Classification, Frameworks (Text 3: Chapter 13)

Module-3

Auditing: Definitions, Anatomy of an Auditing System, Designing an Auditing System, A Posterior Design, Auditing Mechanisms, Examples, Audit Browsing (Text 3: Chapter 14)
Intrusion Detection: Principles, Basic Intrusion Detection, Models, Architecture, Organization of Intrusion Detection Systems, Intrusion Response (Text 3: Chapter 15)

Module-4

Network Security: Introduction, Policy Development, Network Organization, Availability and Network Flooding,

Anticipating Attacks (Text 3: Chapter 16)

System Security: Introduction, Policy, Networks, Users, Authentication, Processes, Files, Retrospective (Text 3: Chapter 17)

Module-5

User Security: Policy, Access, Files and Devices, Processes, Electronic Communications (Text 3: Chapter 18)

Program Security: Introduction, Requirements and Policy, Design, Refinement and Implementations (Text 3: Chapter 19: Section 1, 2, 3, 4)

Course outcomes:

At the end of the course the student will be able to:

1. Explain the various types of attacks on computer and network security from malicious logic and intruders.
2. Explain how to analyze the various vulnerabilities in the system which can compromise the security.
3. Explain how auditing is essential to detect intrusion or suspicious activities in the system.
4. Explain the process involved to provide security with respect to network, system, user and program..

- TEXT BOOKS:

1. Cryptography and Network Security Principles and Practice, Pearson
- William Stallings, 5th Edition, 2014, ISBN: 978-81-317- 6166-3.
2. Cryptography and Network Security, Atul Kahate, TMH, 2003.
3. Matt Bishop, Sathyanarayana S Venkatramanayya, “Introduction to Computer Security”, Pearson Education, 2006, ISBN 81-7758-425-1

REFERENCE BOOKS:

1. Cryptography and Network Security, Behrouz A. Forouzan, TMH, 2007.

MODULE 2

Malicious Logic: Introduction, Trojan Horses, Computer Viruses, Computer Worms, Other Forms of Malicious Logic, Defenses (Text 3: Chapter 12)
Vulnerability Analysis: Introduction, Penetration Studies, Vulnerability Classification, Frameworks (Text 3: Chapter 13)

Introduction to Malicious Logic

- Malicious logic is hardware, firmware, or software intentionally included in a system to perform an unauthorized function, negatively impacting the confidentiality, integrity, or availability of an information system.
- The broader term for this is malware, encompassing viruses, worms, and Trojan horses.

Trojan Horses

Definition: A Trojan horse is a type of malware that deceives users about its true intent by appearing to be a legitimate program.

Method of delivery: Trojans spread through social engineering, where users are tricked into executing them, such as by opening an innocuous-looking email attachment or clicking a fake online advertisement.

Purpose: The payload can vary, but modern Trojans often create a backdoor that allows an attacker unauthorized access to the affected device.

Distinction: Unlike viruses and worms, Trojans typically do not self-replicate by infecting other files or systems.

Computer Viruses

Definition: A computer virus is a self-replicating type of malware that attaches itself to legitimate programs or files. When the infected program is executed, the virus's code also runs, spreading the infection.

Requires a host: Viruses need a host program to spread and cannot run independently.

Examples of virus types:

- Boot sector viruses: Infect the boot sector of a storage disk, executing when the system starts.
- Executable infectors: Attach to executable files like .exe files.
- Multipartite viruses: Can infect both boot sectors and executable files.
- Stealth viruses: Use special techniques to conceal themselves and avoid detection by security software.

Computer Worms

Definition: A computer worm is a standalone malware program that self-replicates and spreads to other computers, typically over a network.

Independent program: Unlike a virus, a worm does not require a host program to spread. It exploits network security vulnerabilities to access and infect other machines.

Impact: While some worms just spread, they can consume network bandwidth and cause major disruption.

Many also carry payloads to damage or modify systems, steal data, or install other malware.

Other Forms of Malicious Logic

Logic bombs: Malicious code that is dormant until a specific event or condition is met, such as a specific date or time, before executing.

Backdoors: A method of bypassing normal authentication and gaining unauthorized remote access to a computer system. They are often installed by Trojans.

Spyware: Software that secretly observes the user's activity, such as keystrokes, and collects sensitive information.

Ransomware: A type of malware that encrypts a user's data and demands a ransom payment to restore access. Trojans are often used to deliver ransomware payloads.

Defenses Against Malicious Logic

Security software: Antivirus and anti-malware programs are essential for detecting and removing malicious logic.

User education: Training users to recognize and avoid social engineering tactics, such as suspicious email attachments, is a crucial defense.

System and software updates: Keeping operating systems and applications patched and up-to-date helps prevent worms and other malware from exploiting known vulnerabilities.

Firewalls: Network and personal firewalls can block malicious network traffic and prevent unauthorized access.

Malicious logic (viruses, worms, Trojans) can harm systems in different ways. Defenses are designed to **detect, block, or limit their effects**. But no defense is perfect – sometimes safe programs may get blocked, or malicious ones may sneak through.

1. Malicious Logic as Both Data and Instructions

A **virus** behaves both like **data** (when it hides in a file) and like **instructions** (when it runs).

Defense idea: **separate data and executable programs clearly**.

- Example: A program should remain just “data” until a trusted authority certifies it as “executable.”
- Once certified, it cannot be modified (to avoid infection).

2. Malicious Logic Using a User’s Identity

When a user unknowingly runs malicious code, it can act as if the user gave permission.

a) Information Flow Metric

- Idea: Limit how far information can spread.
- Each time info is copied, its “flow distance” increases.
- If flow distance becomes too high, stop further spreading.

b) Reducing Rights (Least Privilege)

A user should **limit permissions** when running untrusted programs.

For example, if you only need to *read* a file, don't also allow *write*.

This way, if the program is malicious, it cannot harm files it doesn't need.

c) Sandboxing

- **Sandboxing = run the program in a safe, isolated environment.**
- The program can only act inside the sandbox, not on the whole system.
- Virtual machines or special restricted libraries are often used.

3. Malicious Logic Crossing Protection Boundaries

- Viruses often spread by **sharing files/programs** among users.
- Defense: **restrict sharing across different user domains.**

Example:

If User A shares a program, the system checks:

- Does User B trust User A?
- If not, the program won't run.

4. Malicious Logic Altering Files

- Attackers may **change files secretly**.
- Defense: Use **signatures or cryptographic checksums** to detect tampering.
 - A function creates a **signature block** (like a fingerprint) of the file.
 - If the file changes, its fingerprint changes too → system detects it.

👉 Example:

Tripwire (for UNIX) stores fingerprints of system files. Later, it checks again → if a fingerprint doesn't match, the file was altered (maybe by a virus).

5. Malicious Logic Performing Actions Beyond Specification

- Sometimes programs **don't follow their intended instructions** (like a virus sneaking in extra steps).
- Defense: Use **fault-tolerant techniques** to catch misbehavior.

a) Checksums for Program Execution

- Break program into small chunks.
- Each chunk is verified during execution.
- If checksum doesn't match → program might be infected.

👉 **Example:** If a cooking recipe suddenly has an extra step “add poison,” the checksum fails because instructions changed.

b) N-Version Programming

- Write **different versions** of the same program.
- Run them together and compare results.
- If one version gives a different answer, it's considered faulty (or infected).
- Final output = majority vote.

c) Proof-Carrying Code (PCC)

- Code producer attaches a **mathematical proof** with the program to show it is safe.
 - User verifies the proof before running the code.
 - If the program changes, the proof breaks → code rejected.
- 👉 **Example:** Like a sealed medicine with a lab test certificate. If seal is broken, you won't trust it.

6. Malicious Logic Altering Statistical Characteristics

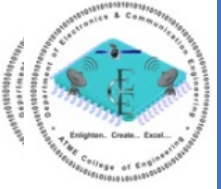
- Programs usually follow certain **patterns** (like writing style).
- If the pattern suddenly changes → might indicate malicious logic.

👉 Examples:

- A program grows much larger than usual → may be infected.
- A file that was rarely executed is suddenly run very often → suspicious.
- Comparing **source code vs. object code**: If extra conditions appear in object code that weren't in the source, infection may have occurred.

7. The Notion of Trust

- **No defense works if the base system itself is not trustworthy.**
- Security depends on:
 - Trusting the hardware/software.
 - Correct implementation.
 - Balance between **cost** and **desired level of protection**.



Vulnerability Analysis (Chapter 13)

Introduction to Vulnerability Analysis

Vulnerability analysis is the process of identifying, classifying, and mitigating vulnerabilities (flaws or weaknesses) in computer systems and networks.

A robust analysis is crucial for preventing security breaches.

• Systems are complex, so flaws occur due to:

- Faulty **design**
- Faulty **implementation**
- Faulty **operation**
- Faulty **maintenance**

👉 Example: A software update (maintenance) may open a new security hole if not tested properly.

Techniques for Detecting Vulnerabilities

•Formal Verification

- A mathematical proof that the system works as intended.
- Can show **no flaws exist** (but only inside the program or design).
- Problem: Hard to include policies, human actions, and environment.

•Property-Based Testing

- Test if system behavior satisfies certain properties (security rules).

•Penetration Testing

- Tries to attack the system to find vulnerabilities.
- Can only **prove flaws exist**, not that the system is 100% safe.

Preconditions & Postconditions (Figure 20-1 Explained)

•Formal Verification

- Preconditions = constraints before running system.
- Postconditions = what should happen after running.
- Check if postconditions follow the **security policy**.

•Penetration Testing

- Preconditions = the state where a flaw might exist.
- Tester sets system into that state and tries exploiting.
- Postconditions = what actually happens after the test.
- If results break the security policy → vulnerability confirmed.

Penetration Studies (or Penetration Testing)

Definition: A penetration study is a simulated cyberattack against a computer system to check for exploitable vulnerabilities. It is a proactive and authorized attempt to find and fix security weaknesses before malicious actors can exploit them.

Purpose: To evaluate the security of a system and provide a clear, actionable report of findings. Testers use the same tools and techniques as real attackers.

Goals of a Penetration Test

- Main goal: Try to **violate security policies** (like accessing files, stealing passwords, or shutting down services).
- Goals must be **clear and measurable**.

👉 Example:

- Specific goal:** Can testers open a password-protected file without a password?
- Broad goal:** Find as many weaknesses as possible within 30 days.

Layers of Penetration Testing

Pen tests can simulate different types of attackers:

1. External attacker (no access)

1. Outsider who doesn't know much.
2. Uses tricks like **social engineering** or scanning networks.
3. Example: Pretending to be IT staff to get info.

2. External attacker (with limited access)

1. Outsider who can connect to the system (e.g., via email or guest login).
2. Attacks with password guessing, unprotected accounts, weak servers.

3. Internal attacker (insider)

1. Has a valid account.
2. Tries to **gain extra privileges** (like becoming admin/root).
3. Example: A normal employee trying to read the boss's emails.

Methodology at Each Layer

- Pen tests use the **attacker's point of view**.
- The goal is not just “did we break in?” but also **why the flaw exists**.
- The **degree of penetration** matters:
 - Reading one user's file = small issue.
 - Gaining system-wide admin = big issue.

Flaw Hypothesis Methodology (Steps)

A common method for pen tests:

- 1.Information Gathering** → Learn everything about the system.
- 2.Flaw Hypothesis** → Guess possible weaknesses.
- 3.Flaw Testing** → Try to exploit them safely.
- 4.Flaw Generalization** → Find similar flaws elsewhere.
- 5.Flaw Elimination** → Suggest fixes (sometimes skipped).

Examples

- 1. Michigan Terminal System** → Students exploited a bug in how parameters were checked, and gained full control of the computer.
- 2. Burroughs System** → Students tricked the system into treating a fake compiler as a real one, giving themselves admin power.
- 3. Corporate System** → Testers didn't even hack computers — they used **phone calls and impersonation** to steal employee numbers and passwords → proved operations were weak.
- 4. UNIX System** → Testers scanned network ports, found an old **send mail** vulnerability, and got root access.

Vulnerability Classification

Vulnerabilities can be categorized in several ways, such as by:

- Type of flaw: Such as memory safety issues (e.g., buffer overflows), insecure configurations, or design errors.
- Location in the system: For example, network-level, operating system, or application-level vulnerabilities.
- Severity: Graded by the potential impact if exploited (e.g., high, medium, low).
- Source: Whether the vulnerability is due to software design, implementation, or user configuration.

Two Classic UNIX Flaws

These help illustrate classification challenges:

1.xterm flaw (TOCTTOU – Time of Check to Time of Use)

- xterm checks if a user can write to a file, then opens it.
- Attacker quickly swaps the file with `/etc/passwd`.
- Result → logs get written into password file → attacker gains **root** access.

👉 **Type:** race condition / inadequate serialization.

2.finger daemon flaw (Buffer Overflow)

- fingerd expected ≤ 512 characters.
- Attacker sent 536 characters → overflowed buffer, overwrote return address.
- Program jumped into attacker's code → attacker got a shell with privileges.

👉 **Type:** buffer overflow / incomplete parameter validation.

Vulnerability Frameworks

Common Vulnerabilities and Exposures (CVE): A dictionary of standardized identifiers for publicly known cybersecurity vulnerabilities. A CVE identifier provides a common reference point for sharing information about a specific vulnerability.

Common Vulnerability Scoring System (CVSS): A framework for rating the severity of a vulnerability. It provides a numerical score based on a standard set of characteristics, such as exploitability and impact.

Open Web Application Security Project (OWASP): A non-profit foundation that provides resources for improving software security. It is well known for its OWASP Top 10, a list of the most critical security risks to web applications.

RISOS Study (1970s)

- One of the **first systematic studies**.

- Looked at operating system flaws.

- **7 Classes of Flaws:**

- Incomplete parameter validation → e.g., buffer overflow.
- Inconsistent parameter validation → different modules expect different formats.
- Implicit sharing of confidential data → side-channel leaks, timing attacks.
- Asynchronous validation / inadequate serialization → race conditions.
- Inadequate authentication/authorization → weak logins, Trojan horses.
- Violable limits → poor bounds checking.
- Exploitable logic errors → mistakes in program logic.

👉 **Legacy:** Showed vulnerabilities are **few fundamental types that repeat**.

Protection Analysis (PA) Model

- Broke security into **smaller, manageable pieces**.
- Developed **pattern-based evaluation**.
- **4 Major Classes (with subcategories):**
 - **Improper protection domain initialization/enforcement** → wrong privileges assigned.
 - **Improper validation** → not checking inputs properly.
 - **Improper synchronization** → race conditions, sequencing errors.
 - **Improper choice of operation** → e.g., weak random numbers.

👉 **Legacy:** First to explore **automatic detection** of flaws.

NRL Taxonomy (1992)

- Asked **3 key questions** about vulnerabilities:
 - **Genesis (how)** → accidental or intentional?
 - Accidental: coding mistakes.
 - Intentional: Trojan horse, trapdoor, logic bomb.
 - **Time (when)** → design, development, maintenance, or operation.
 - **Location (where)** → OS, application, privileged utilities, hardware, etc.
- 👉 **Legacy:** Added the **social dimension** (malicious insiders vs mistakes).

Aslam's Model

- Based on **software fault studies**.
 - Focused on **UNIX implementation flaws**.
 - **2 Broad Types:**
 - **Coding faults** → synchronization errors (timing) + condition validation errors (input checks).
 - **Emergent faults** → appear due to configuration or environment.
 - Example: misconfigured service exposing files.
- 👉 **Legacy:** Introduced a **decision procedure** → each flaw belongs to **exactly one category** (good for vulnerability databases).

Framework	Focus / Approach	Main Classes of Flaws
RISOS Study (1970s)	Operating system flaws	7 Classes: 1. Incomplete parameter validation 2. Inconsistent validation 3. Implicit sharing of data 4. Asynchronous validation / inadequate serialization 5. Inadequate authentication/authorization 6. Violable limits 7. Exploitable logic errors
Protection Analysis (PA)	Pattern-based flaw evaluation	4 Major Classes: 1. Improper protection domain init/enforcement 2. Improper validation 3. Improper synchronization 4. Improper choice of operation
NRL Taxonomy (1992)	High-level classification	3 Key Dimensions: 1. Genesis (how) : accidental vs intentional (Trojan, logic bomb, trapdoor) 2. Time (when) : design, development, maintenance, operation 3. Location (where) : OS, application, hardware, privileged utilities, etc.
Aslam's Model	Based on UNIX implementation faults	2 Broad Types: 1. Coding faults : sync errors, condition validation errors 2. Emergent faults : due to configuration/environment