

# UNIT 8

## Security Management

(3 Hrs) — Operating System (CACS251)

---

### 8.1 Introduction

**Security** is concerned with protecting a system's resources (data, programs, hardware) from unauthorised access, alteration, destruction and denial of service. Where *protection* is an internal mechanism controlling access to resources, **security** requires a broader view — the external environment (users, networks, physical premises) in which the system operates.

#### Goals of security (the CIA triad)

| Goal                   | Meaning                                                              | Threat                     |
|------------------------|----------------------------------------------------------------------|----------------------------|
| <b>Confidentiality</b> | Secret data remains secret — only authorised users can read it       | Data exposure / theft      |
| <b>Integrity</b>       | Unauthorised users cannot modify data without the owner's permission | Tampering, data corruption |
| <b>Availability</b>    | Nobody can disturb the system to make it unusable                    | Denial of service          |

### 8.2 Security Problems

A system is secure if its resources are used and accessed **as intended** under all circumstances. Perfect security is unattainable, but the cost of a violation should be made higher than the potential gain. Security violations may be **intentional** (malicious) or **accidental** (mistakes, hardware faults, acts of nature). Types of malicious violations:

- **Breach of confidentiality** — theft of data (e.g., credit card numbers, identity).
- **Breach of integrity** — unauthorised modification of data (e.g., altering exam marks or bank balances).
- **Breach of availability** — unauthorised destruction of data or making a service unavailable (website defacement, DoS).
- **Theft of service** — unauthorised use of resources (e.g., a daemon installed on a machine to act as a file server or spam relay).
- **Denial of service (DoS)** — preventing legitimate use of the system.

Attack methods include **masquerading** (pretending to be an authorised user), **replay attack**, **man-in-the-middle attack**, **session hijacking** and **privilege escalation**. Security must be maintained at four levels: **physical** (machine rooms, locks), **human** (social engineering, phishing, dumpster diving), **operating system** and **network**.

## 8.3 User Authentication

**Authentication** is the process of verifying the identity claimed by a user before granting access. It is based on one or more of: **something the user knows** (password, PIN), **something the user has** (smart card, token, phone), **something the user is** (fingerprint, iris — biometrics). Combining two or more of these is called **multi-factor authentication (MFA)**.

### 8.3.1 Passwords

The most common approach: the user supplies a **user name** (identity) and a **password** (proof). The system compares it against stored information. Advantages: simple, cheap, easy to change. Weakness: passwords can be guessed, sniffed, shared or stolen.

### 8.3.2 Password Vulnerabilities

| Vulnerability                            | Description                                                                                                                             |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Guessing</b>                          | Using the user's name, birth date, pet's name or common words ("123456", "password"); intelligent guessing succeeds surprisingly often. |
| <b>Brute force attack</b>                | Trying every possible combination of characters; time grows exponentially with password length.                                         |
| <b>Dictionary attack</b>                 | Trying every word in a dictionary and common variants (p@ssw0rd) — or comparing the hashes against a precomputed list (rainbow tables). |
| <b>Shoulder surfing</b>                  | Watching a user type the password over their shoulder or through a camera.                                                              |
| <b>Sniffing / Eavesdropping</b>          | Capturing passwords in transit over an unencrypted network (Telnet, HTTP, FTP).                                                         |
| <b>Social engineering / Phishing</b>     | Tricking users into revealing the password via fake e-mails or websites, or by phone.                                                   |
| <b>Keyloggers, Trojan login programs</b> | Malware that records keystrokes or imitates the login screen.                                                                           |
| <b>Password reuse and sharing</b>        | One leaked password opens many accounts.                                                                                                |

**Good practice:** long passphrases ( $\geq 12$  characters) mixing letters, digits and symbols; different passwords for different systems; password managers; account lock-out after several failed attempts; forced periodic change only when compromise is suspected; multi-factor

authentication.

### 8.3.3 Encrypted Passwords

Storing passwords in plain text is dangerous: anyone who reads the password file learns every password. Instead the system stores only a **one-way (hash) function** of the password. UNIX historically used **crypt()** (a modified DES). At login the typed password is hashed and the result compared with the stored value — the original password can never be recovered from the file.

- **Salt** — a random value appended to the password before hashing (stored with the hash). Identical passwords then produce different hashes, defeating precomputed rainbow tables.
- **Slow hashes** — bcrypt, scrypt, PBKDF2 and Argon2 add computational cost to slow down brute force.
- **Shadow password file** — in modern UNIX/Linux, the hashes are kept in */etc/shadow*, readable only by root, while */etc/passwd* (world-readable) has no hashes.

#### Example — Salted hash entry

*/etc/shadow*: ram : \$6\$Qw3rTy9\$X7p... : 19850 : 0 : 99999 : 7 : ::

Here **\$6\$** = SHA-512 algorithm, **Qw3rTy9** = salt, remainder = hash of (salt + password).

### 8.3.4 One-Time Passwords (OTP)

A **one-time password** is valid for only **one login session or transaction**, so even if it is captured it cannot be reused (defeats sniffing and replay). Generation approaches:

- **Lists / pads** of pre-issued passwords (bank scratch cards).
- **Hash-chain (Lamport's scheme)**: the server stores  $H^n(\text{password})$ ; the user sends  $H^{n-1}(\text{password})$  at first login, and so on backwards, so each password is used once.
- **Time-based OTP (TOTP)** — an app or hardware token (e.g., RSA SecurID, Google Authenticator) computes a code from a shared secret and the current time, valid for ~30 seconds.
- **SMS/e-mail OTP** sent to the registered phone or address.

### 8.3.5 Biometrics

Uses a unique **physical or behavioural** characteristic of the user: fingerprint, palm/hand geometry, iris or retina scan, face recognition, voice, signature dynamics and keystroke rhythm. Advantages: cannot be forgotten or easily shared. Drawbacks: cost, **false accept rate (FAR)** and **false reject rate (FRR)**, privacy concerns, and the fact that a compromised biometric **cannot be changed**. Biometrics are usually combined with a password or token (two-factor).

| Method             | What is used              | Strength            | Weakness                                         |
|--------------------|---------------------------|---------------------|--------------------------------------------------|
| Password           | Something you know        | Cheap, simple       | Guessable, phishable                             |
| One-time password  | Something you have / know | Not reusable        | Needs token or phone; man-in-the-middle possible |
| Smart card / token | Something you have        | Hard to copy        | Can be lost or stolen                            |
| Biometrics         | Something you are         | Cannot be forgotten | Cost, error rates, cannot be revoked             |

### 8.4 User Authorization

**Authentication** answers “*who are you?*”; **authorization** answers “*what are you allowed to do?*” Once a user is authenticated the OS decides which resources they may use and how. It is implemented through **access-control mechanisms** — permissions (UNIX rwx), **ACLs**, **capabilities**, **role-based access control (RBAC)** where rights are attached to roles (admin, student, teacher) rather than individuals, and **domains of protection** with the **principle of least privilege** (each program and user gets the minimum rights needed to do its job). Mechanisms such as **setuid** in UNIX or “Run as administrator/UAC” in Windows allow controlled temporary elevation of privilege.

| Authentication                     | Authorization                                         |
|------------------------------------|-------------------------------------------------------|
| Verifies identity                  | Determines permitted actions                          |
| Done first (login)                 | Done after successful authentication, on every access |
| Uses passwords, tokens, biometrics | Uses ACLs, permissions, roles, capabilities           |

### 8.5 Program Threats

A process running with the user’s privileges can misuse them. Threats that originate from programs are known as **program threats**.

### 8.5.1 Trojan Horse

A **Trojan horse** is a program that **appears useful or harmless** (a game, utility or free software) but secretly performs malicious actions when executed with the user's privileges — stealing data, installing back doors or deleting files. Named after the wooden horse of Troy. It does not replicate by itself. A classic example is a **fake login program** that records the user name and password and then displays “login incorrect”. Another risk is a Trojan in the search path: a malicious *ls* program placed in a directory that comes before /bin in \$PATH (**long search path**, “.” in path). *Defences*: download software only from trusted sources, verify checksums and signatures, do not put “.” in the PATH, use least privilege and antivirus.

### 8.5.2 Trap Door (Back Door)

A **trap door** is a secret entry point deliberately left in a program by its designer or programmer — a hidden way to bypass normal authentication or security checks (for example a special user name that always logs in, or code that checks for a particular input or date). Trap doors may be inserted into a **compiler** so that every program compiled contains the hole even though the source code looks clean. They are hard to detect because analysis requires reviewing all source code of all components. *Defences*: code review, source-code audits, and controlling and verifying the software build chain.

### 8.5.3 Stack and Buffer Overflow

The most common way for an attacker to gain unauthorised access from outside. It exploits a program that **does not check the length of input** before copying it into a fixed-size buffer (e.g., C functions *gets*, *strcpy*, *sprintf*). Excess data overflows the buffer and overwrites adjacent memory on the stack — including the **return address** of the function.

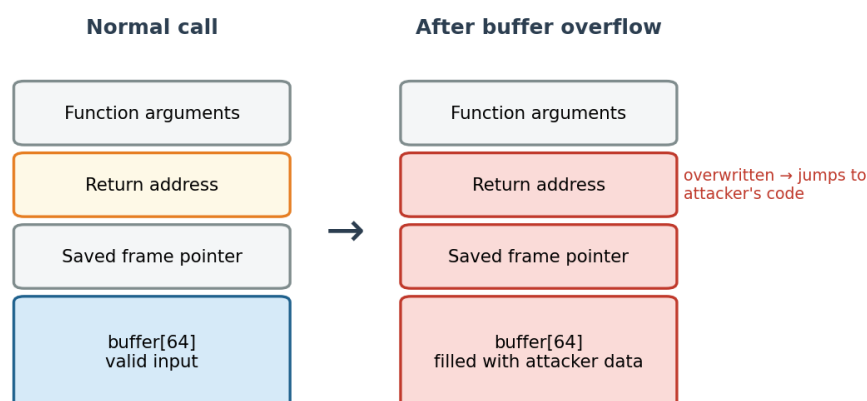


Figure 8.1 — Stack layout before and after a buffer overflow

```
void vulnerable(char *input) {
    char buffer[64];
    strcpy(buffer, input);    /* no bounds check! */
}                            /* input longer than 64 bytes overwrites
                             saved frame pointer and return address */
```

**Attack steps:** (1) find a program with an unchecked buffer; (2) craft input larger than the buffer that contains **malicious code (shellcode)** plus a new return address pointing to it; (3) when the function returns, control jumps to the attacker's code, which runs with the program's privileges (often a shell as root if the program is setuid-root).

**Defences:** use safe functions (*strncpy*, *fgets*, *snprintf*) and bounds checking; **stack canaries** (random value placed before the return address and verified on return); **non-executable stack** (NX/DEP bit); **ASLR** (address-space layout randomisation); safer languages (Java, Python, Rust); code review and fuzz testing.

## 8.6 System Threats

System and network threats involve the abuse of services and network connections. They spread across multiple machines using the operating system's own facilities.

### 8.6.1 Worms

A **worm** is a **self-replicating, standalone** program that spreads across networks **without a host file and without user action**, exploiting vulnerabilities in network services (mail, remote login, file sharing). Each copy reproduces itself on new machines, consuming network bandwidth and CPU and often carrying a payload. Historic example: the **Morris worm (1988)** exploited a buffer overflow in *fingerd*, the debug mode of *sendmail* and weak passwords (dictionary attack) to infect an estimated 6,000 UNIX machines; later: ILOVEYOU, Code Red, Slammer, Conficker, WannaCry (2017).

### 8.6.2 Viruses

A **virus** is a fragment of code embedded in a legitimate program (the **host**) which replicates itself by **infecting other programs** when the host is executed and spreads when infected files are shared (e-mail attachments, USB drives, downloads). A virus requires a **human action** (running the infected program) to spread.

| Virus type       | Description                                             |
|------------------|---------------------------------------------------------|
| File (parasitic) | Attaches itself to executable files (.exe, .com)        |
| Boot sector      | Infects the master boot record; runs at system start-up |

| Virus type                | Description                                                 |
|---------------------------|-------------------------------------------------------------|
| Macro                     | Written in macro languages of Word/Excel documents          |
| Polymorphic / Metamorphic | Changes its own code each time to avoid signature detection |
| Stealth                   | Hides its presence by intercepting system calls             |
| Memory-resident           | Stays in memory and infects every program executed          |

| Feature              | Virus                    | Worm                                  | Trojan horse                      |
|----------------------|--------------------------|---------------------------------------|-----------------------------------|
| Needs a host program | Yes                      | No — standalone                       | Disguised as a legitimate program |
| Replicates itself    | Yes (by infecting files) | Yes (across networks)                 | No                                |
| Needs user action    | Yes — run infected file  | No — exploits network vulnerabilities | Yes — user installs/runs it       |
| Main damage          | Corrupts/deletes files   | Consumes bandwidth, opens back doors  | Steals data, gives remote access  |

### 8.6.3 Denial of Service (DoS)

A **denial-of-service attack** does not try to steal or modify information but to **disrupt legitimate use** of a system or service by overloading its resources (CPU, memory, bandwidth, connections). Because it typically comes from a remote location, it is hard to stop. When the attack originates from thousands of compromised machines (a **botnet**) it is a **Distributed DoS (DDoS)**.

- **SYN flood** — sends a stream of TCP connection requests (SYN) and never completes the handshake, exhausting the server's connection table.
- **Ping of death / ICMP flood** — oversized or excessive ping packets.
- **Smurf attack** — spoofed ICMP requests to a broadcast address amplify the response toward the victim.
- **Resource exhaustion** — fork bombs, filling the disk, opening excessive connections.
- **Defences:** firewalls and rate limiting, ingress/egress filtering, SYN cookies, intrusion-prevention systems, load balancing and CDNs, and having sufficient spare capacity.

*Note: A **logic bomb** (code that triggers a malicious action when a condition — a date or the absence of an employee's name from payroll — is met), **spyware**, **ransomware** and **rootkits** are related threats.*

## 8.7 Defensive Measures — Overview

| Measure                                     | Purpose                                                |
|---------------------------------------------|--------------------------------------------------------|
| Authentication and strong passwords / MFA   | Verify identities                                      |
| Access control (ACL, RBAC, least privilege) | Limit what authenticated users can do                  |
| Encryption                                  | Protect confidentiality of stored and transmitted data |
| Firewalls and IDS/IPS                       | Block and detect malicious network traffic             |
| Antivirus / anti-malware                    | Detect and remove known malicious code                 |
| Patching and updates                        | Close known vulnerabilities (buffer overflows etc.)    |
| Auditing and logging                        | Record actions to detect and investigate intrusions    |
| Backups                                     | Recover from data loss, ransomware and disasters       |
| User education                              | Reduce phishing and social-engineering success         |

### Key Takeaways

- ✓ Security goals: **confidentiality, integrity, availability**; violations may be breach of confidentiality/integrity/availability, theft of service or DoS.
- ✓ Authentication: passwords (vulnerable to guessing, brute force, dictionary attacks, sniffing), **encrypted (salted hash) passwords, one-time passwords and biometrics**; use MFA.
- ✓ Authorization decides what an authenticated user may do (ACLs, roles, least privilege).
- ✓ Program threats: **Trojan horse** (disguised), **trap door** (hidden entry), **stack/buffer overflow** (unchecked input overwrites the return address).
- ✓ System threats: **worms** (self-propagating over networks), **viruses** (infect host files) and **DoS/DDoS** (overload a service).

— End of Unit 8 —