Secure Programming

Practices for Secure Programming_Part3

Dr. Fatma ElSayed

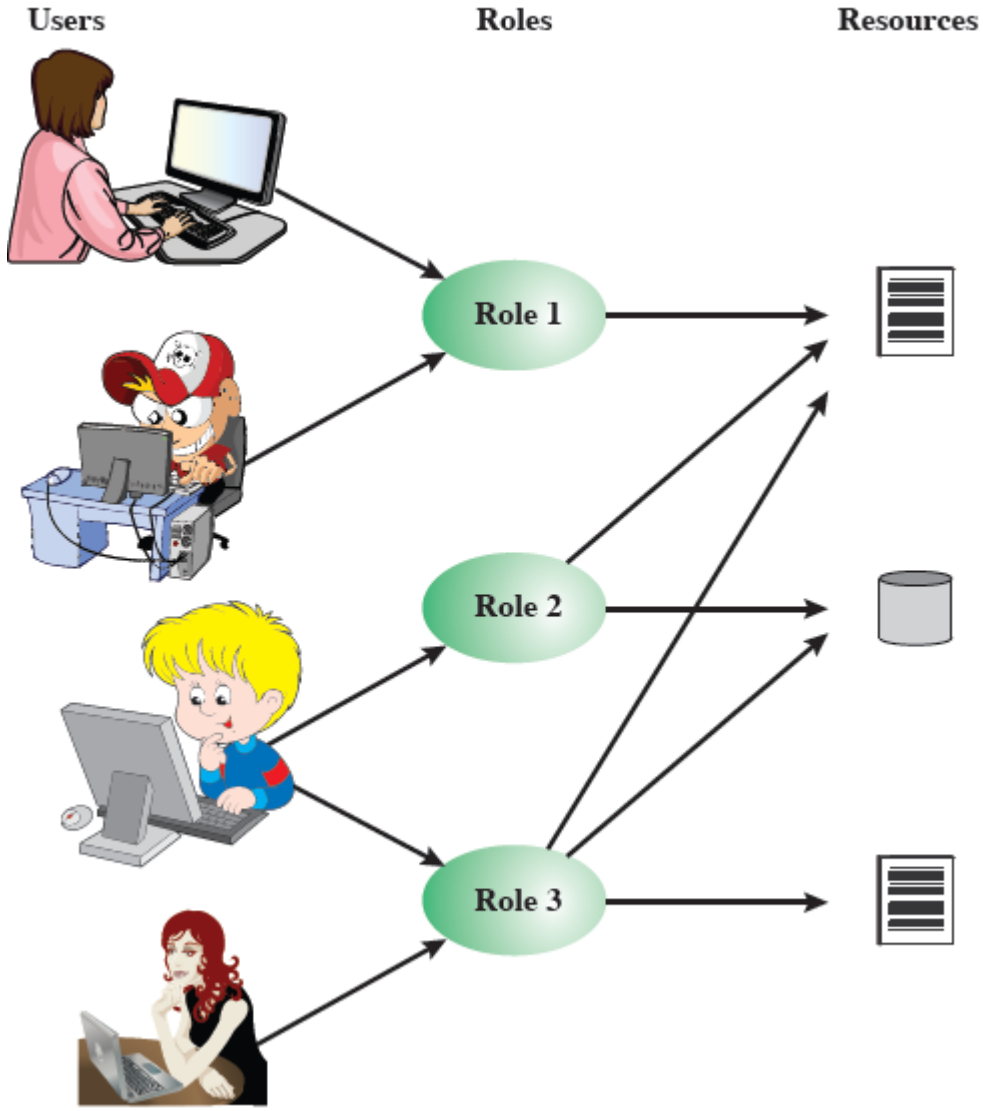
Computer Science Department
fatma.elsayed@fci.bu.edu.eg

Outline

- Access Control Policies
  - Discretionary
  - Mandatory
  - **Role-based**
- Memory Corruption

Role-Based Access Control (RBAC)

- Access based on **‘role’**, not identity
- **Many-to-many** relationship between users and roles
- Roles often **static**

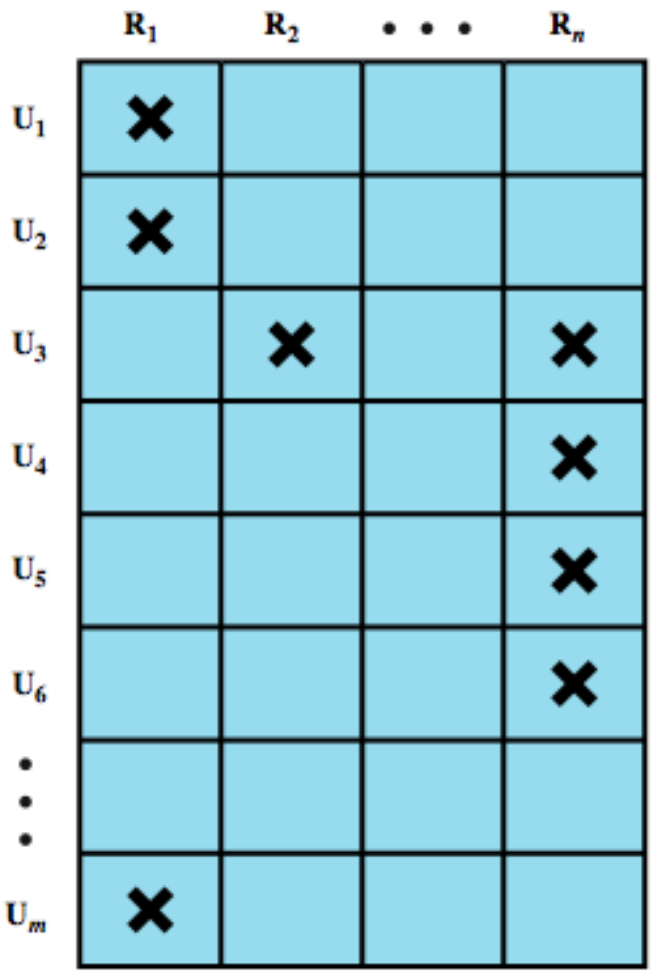


Role-Based Access Control (RBAC)

Access control matrix

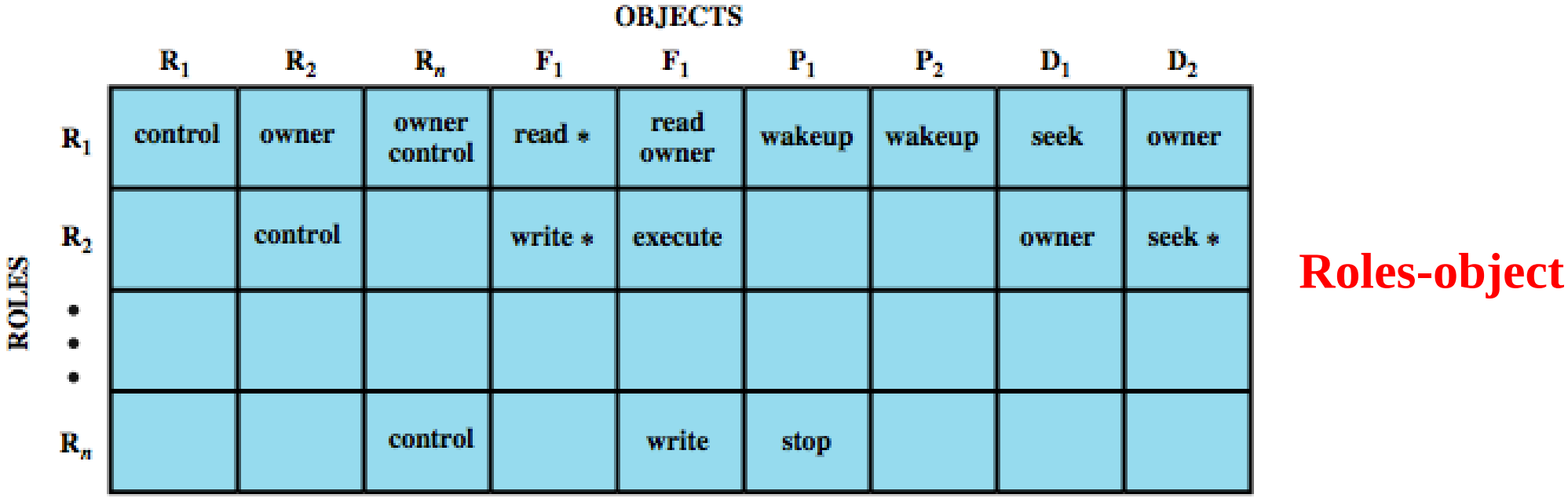
1. Role-users
2. Roles-object

- Rows represent **users** (U_1, U_2, ..., U_m).
- Columns represent **roles** (R_1, R_2, ..., R_n).
- Entries with ***(X)*** indicates a user is **assigned to a specific role.**



Role-users

Role-Based Access Control (RBAC)



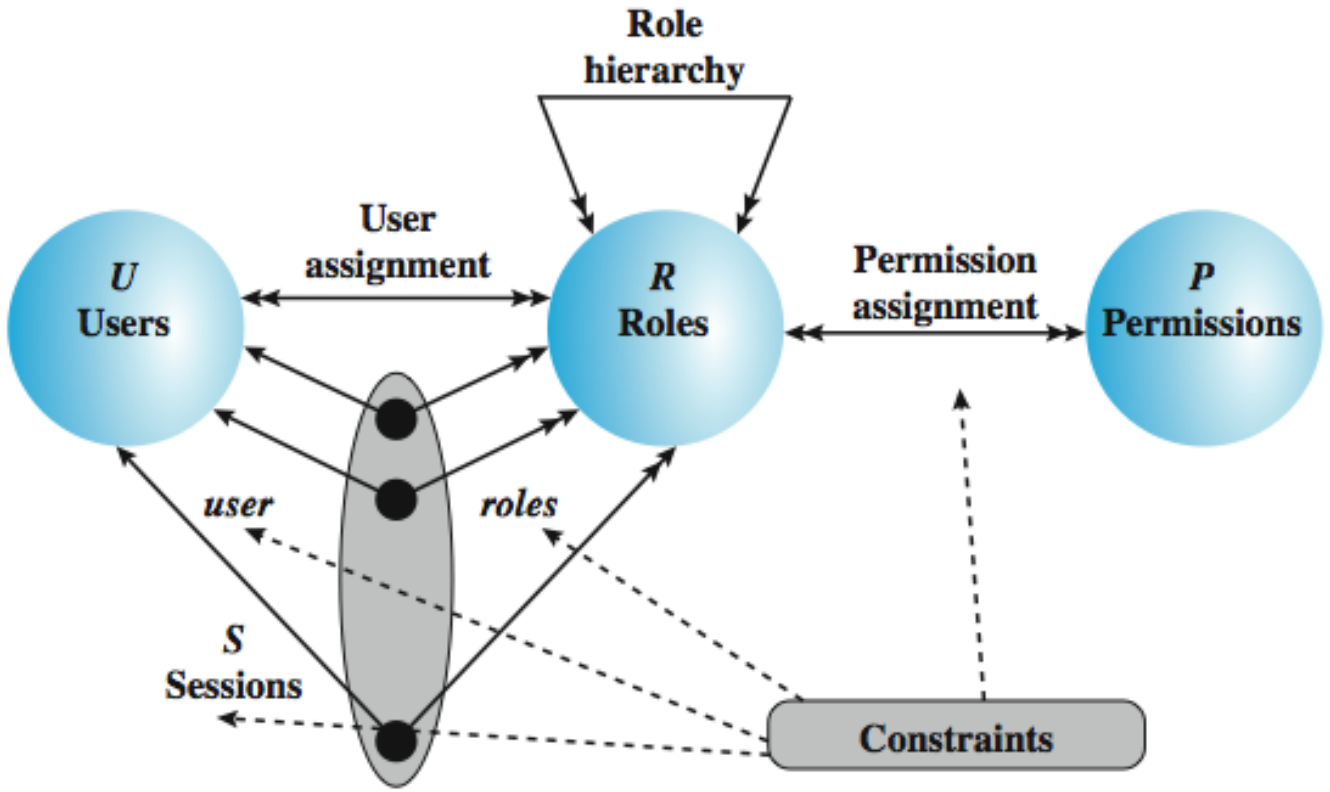
ROLES \ OBJECTS	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
• • •									
R_n			control		write	stop			

Roles-object

- Rows represent **roles** (R_1, R_2, ..., R_n).
- Columns represent **objects** (files, processes, devices) with specific **permissions**.
- Entries define the **access rights** (e.g., "control," "write," "read *," "execute," etc.) a role has on an object.

Role-Based Access Control (RBAC)

- Double arrow: *'many' relationship*
- Single arrow: *'one' relationship*
- The business function the user **performs** is a **role**
- A user can invoke **multiple** sessions
- In each session, a user can invoke any **subset of roles** that the user is a member of



(b) RBAC models

Constraints - RBAC

- Provide a means of adapting RBAC to the **specifics of administrative** and **security policies** of an organization.
- A condition **(restriction)** on a role or between roles.

Mutually Exclusive Roles *(Conflicting Roles)*

- A user can only be assigned to one role in the set *(during a session).*
- Any permission can be granted to only one role in the set.

Cardinality

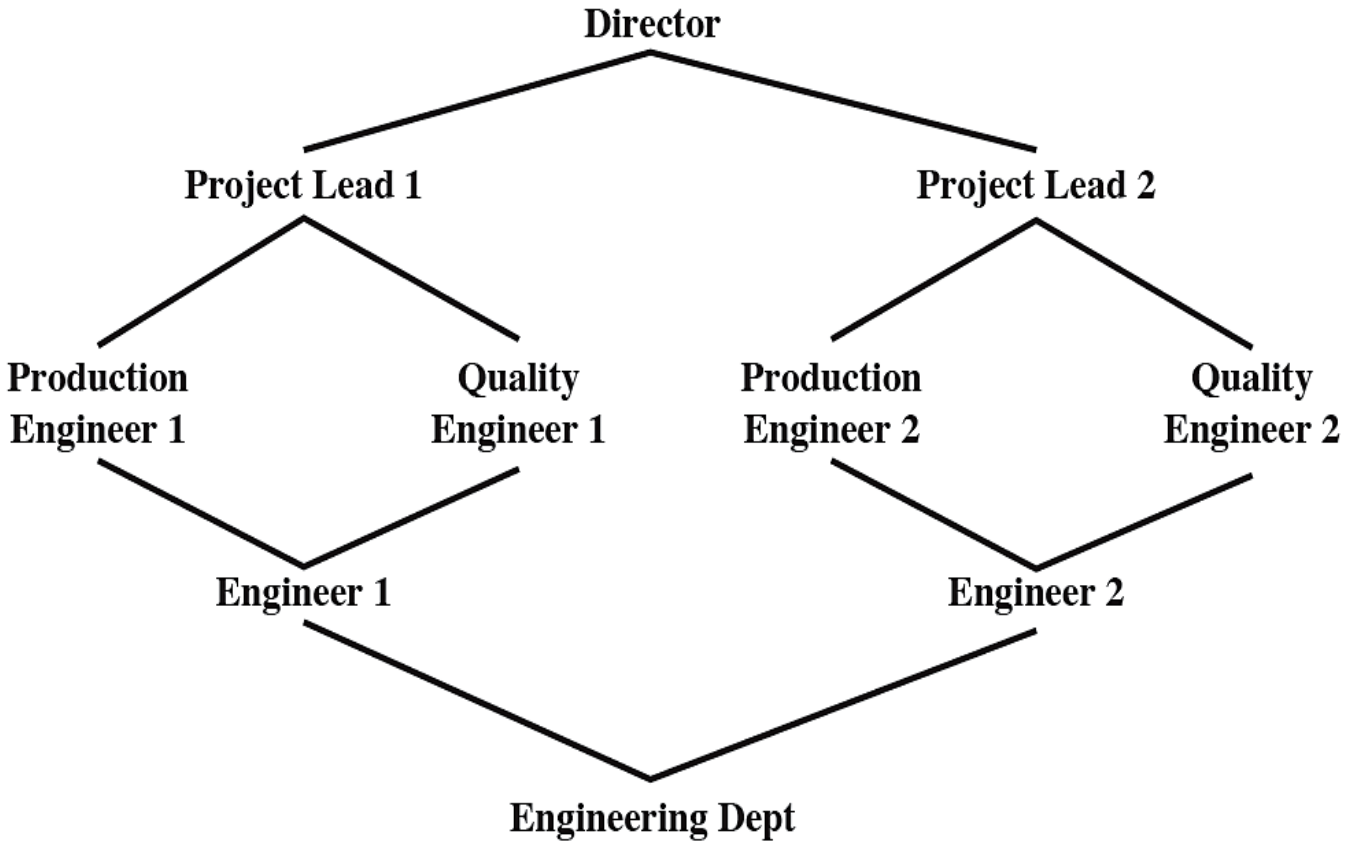
- Setting a maximum number with respect to roles.

Prerequisite Roles

- Dictates that a user can only be assigned to a particular role if it is already assigned to some other specified role.

Example of role hierarchy

- Director has **most** privileges
- Each role inherits all privileges from **lower** roles
- A role can inherit from **multiple** roles



RBAC System Functional Areas

Administrative Functions

Allow administrators to **create, delete, and maintain RBAC elements** (such as users, roles, and permissions) and their relationships.

- *Example:* Assigning a user to a role, removing permissions from a role, or defining role hierarchies.

Supporting System Functions

Help in **session management** and making **access control decisions** dynamically.

- *Example:* When a user logs in, the system determines which roles they can activate during that session.

Review Functions

Allow administrators to **query and analyze** RBAC elements and relationships.

- *Example:* Checking which users have access to a particular resource or reviewing role assignments.

Memory Corruption

Kinds of Vulnerability

- Software vulnerabilities fall into categories, for example:
  - Memory corruption errors
  - Injection
  - Broken authentication
  - Bad cryptography

Reasons for Memory Corruption

Memory corruption vulnerabilities arise from possible:

- **Buffer overflows**, in different places
  - stack overflows
  - heap overflows
- Other programming mistakes
  - Pointer arithmetic mistakes
  - Type confusion errors

Buffer Overflows

What is a Buffer?

- A buffer is an array *(area of memory storage)* used to **temporarily store** data.
- Used to **improve** the performance and **speed** of data access.

Buffer Overflow

A condition under which **more input can be placed into a buffer** than the capacity allocated, overwriting other information.

Consequences:

- **Corruption** of program data
- Unexpected **transfer** of control
- Memory access **violations**
- **Execution** of code chosen by attacker

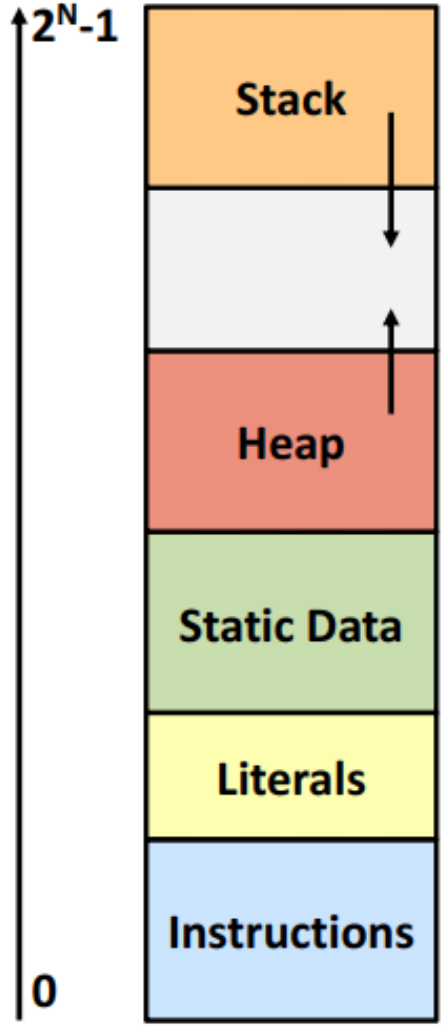
Review: General Memory Layout

Buffer could be located on the stack, in the heap, or in the data section of the process.

- **Stack**
  Local variables (procedure context)
- **Heap**

  Dynamically allocated as needed
- **Statically allocated Data**
  - Read/write: global variables (Static Data)
  - Read-only: string literals (Literals)
- **Code/Instructions**

  Executable machine instructions



Basic Buffer Overflow Example

```
#include <stdio.h>
void vulnerableFunction(char* input) {
char buffer[8];
strcpy(buffer, input);
printf("Buffer: %s\n", buffer);
}
int main() {
char maliciousInput[20] = "MaliciousInput";
vulnerableFunction(maliciousInput);
return 0;
}
```

Buffer Overflow Attacks

To exploit a buffer overflow an attacker needs:

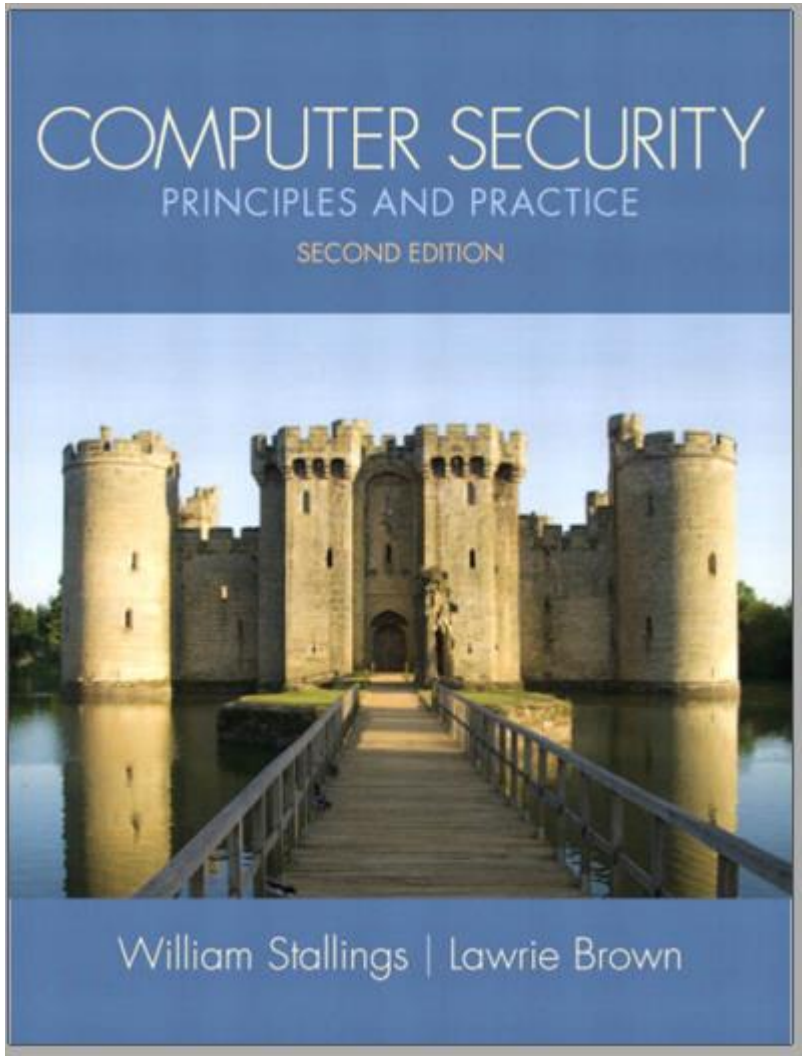
- To **identify a buffer overflow** vulnerability in some program .
- To understand **how that buffer is stored** in memory and determine potential for corruption.

Identifying vulnerable programs can be done by:

- **Inspect the program's code:** They look for unsafe functions (e.g., gets(), strcpy() in C).
- **Tracing the execution of programs:** They run the program with large inputs to see if it crashes or behaves abnormally.
- **Using tools such as fuzzing:** to automatically identify potentially vulnerable programs.

References

- Computer Security: Principles and Practice, William Stallings, 2nd Edition.
- ***Chapter 4: access control***



THANK YOU