Please check your attendance using Blackboard!

Lecture 3 – Memory Safety

[COSE451] Software Security

Instructor: Seunghoon Woo

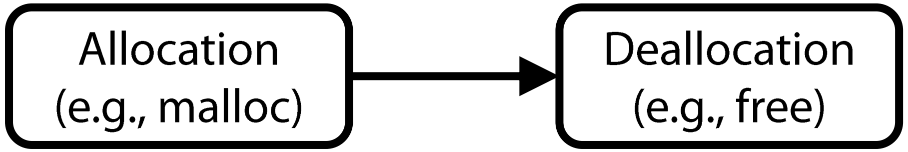
Spring 2024

Overview

- Vulnerabilities caused by improper memory management
- Access controls

Memory safety

- **Vulnerabilities caused by improper memory management**
  - When we dynamically allocate memory in C/C++, it is essential to deallocate it
    - i.e.,

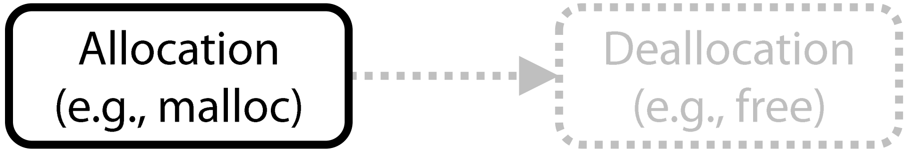


  - If this is not done correctly, vulnerabilities can occur
    (1) Memory Leak
    (2) Double Free
    (3) Use After Free

Memory safety

(1) Memory Leak

- Memory allocated but not deallocated (freed)




```c
/* Function with memory leak */
#include <stdlib.h>

void f()
{
    int* ptr = (int*)malloc(sizeof(int));

    /* Do some work */
    /* Return without freeing ptr*/
    return;
}
```

Memory safety

(2) Double Free

- Attempting to deallocate memory that has already been deallocated

```c
/* Function with double-free */
#include <stdio.h>
#include <stdlib.h>

void f()
{
    char* ptr = (char*)malloc (SIZE);
    /* Do Some work */

    if (abrt) {
        free(ptr);
        /* freeing ptr */
    }
    ...
    free(ptr);
    /* freeing ptr again: double-free */
}
```

Memory safety

(3) Use After Free

- Attempting to access deallocated memory

```c
/* Function with use after free */
#include <stdio.h>
#include <stdlib.h>

void f()
{
    char* ptr = (char*)malloc (SIZE);
    /* Do Some work */

    if (err) {
        abrt = 1;
        free(ptr);
        /* freeing ptr */
    }
    /* Do Some work */

    if (abrt) {
        logError("operation aborted before commit", ptr);
        /* access deallocated memory ptr: use after free */
    }
}
```

Memory safety

- **These vulnerabilities are related to memory management and are dangerous problems that threaten the stability and security**
  - Memory corruption
  - Data leak
  - Memory manipulation
  - Decrease system performance
  - Shell code injection

Memory safety

- **It seems that the issue can be easily resolved by simply mapping allocation (e.g., malloc) and free (e.g., free) statements correctly**

Memory safety

- **It seems that the issue can be easily resolved by simply mapping allocation (e.g., malloc) and free (e.g., free) statements correctly**
  - **FALSE**
  - Many developers are still fighting with this issue

Memory safety

- **Example: Linux kernel case**
  - Original code
    - Can you find where the problem is?

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Lee, Junhee; Hong, Seongjoon; Oh, Hakjoo. "MemFix: Static Analysis-Based Repair of Memory Deallocation Errors for C," *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018. p. 95-106.

Memory

- **Example: Linux kernel case**
  - First patch (2007. 09)
    - The existing problem was solved, but the root cause was not resolved

```c
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

```c
in = malloc(1);
out = malloc(1);
...
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL; // +
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL; // +
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

Memory

- **Example: Linux kernel case**
  - Second patch (2008. 06)
    - Can you find where the problem is?

```c
in = malloc(1);
out = malloc(1);
...
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL; // +
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL; // +
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

```c
in = malloc(1);
out = malloc(1);
...
// -
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out); // +
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

Memory

- **Example: Linux kernel case**
  - Third patch (2008. 07)
    - The existing problem was solved, but the code becomes even more redundant and confusable

```
in = malloc(1);
out = malloc(1);
...
// -
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out); // +
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

```
in = malloc(1);
out = malloc(1);
...
free(out); // +
free(in);
out = NULL; // +
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
// -
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

Memory

- **Example: Linux kernel case**
  - Third patch (2008. 07)
    - The existing problem was solved, but the code becomes even more redundant and confusable

```c
in = malloc(1);
out = malloc(1);
...
// -
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

```c
in = malloc(1);
out = malloc(1);
...
free(out); // +
free(in);
out = NULL; // +
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
...
err:
  free(in);
  free(out);
  return;
```

This is not an easy problem and must be managed carefully!

Lecture 4 – Access Controls

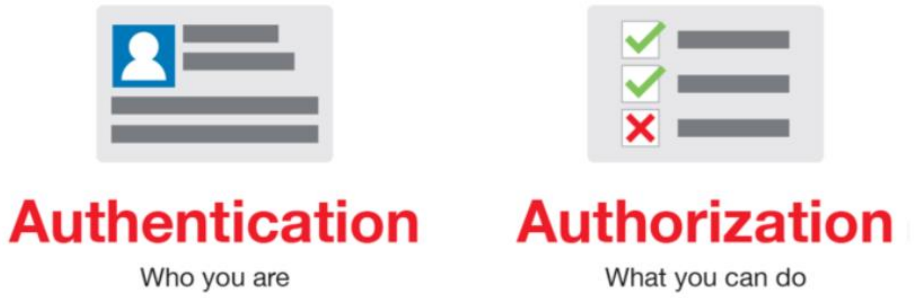
[COSE451] Software Security

Instructor: Seunghoon Woo

Spring 2024

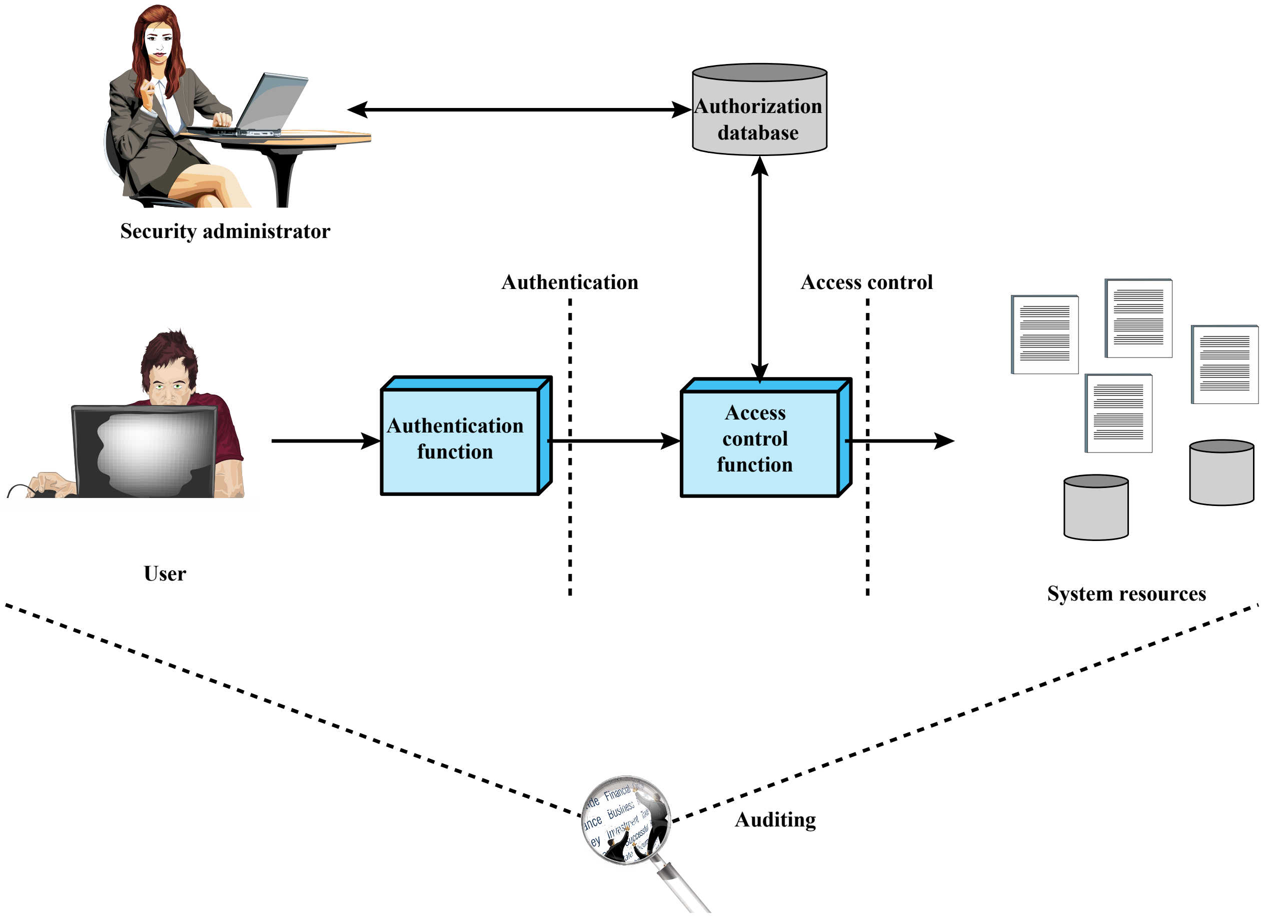
Access Controls

- **A process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities**
  - The process of determining whether a resource is available in a system



  - Whereas authorization policies define what an individual identity or group may access, access controls are the methods we use to enforce such policies

Access Controls

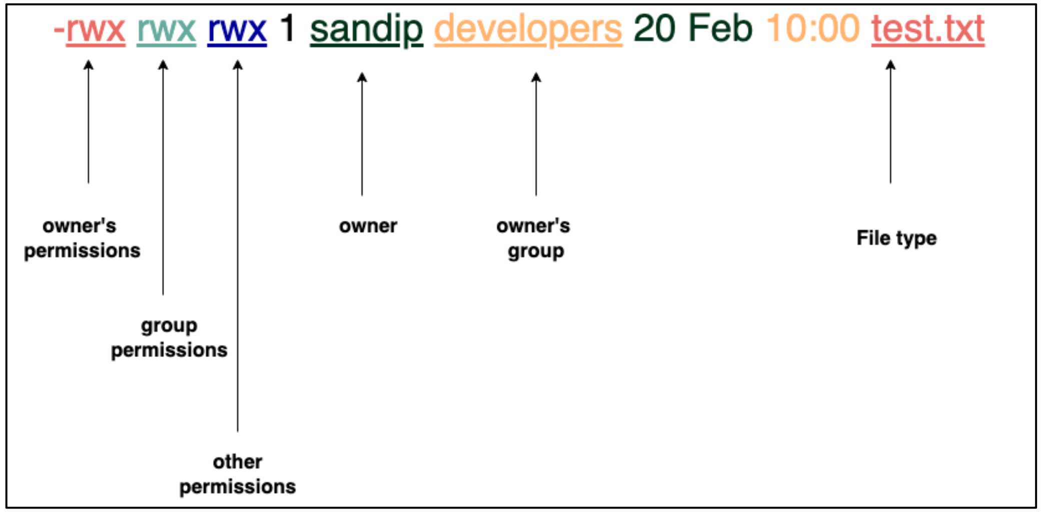


Access Controls

- **The central element of computer security**
  - To prevent unauthorized users from gaining access to resources,
  - To prevent legitimate users from accessing resources in an unauthorized manner
  - To enable legitimate users to access resources in an authorized manner

Access Controls

- **The central element of computer security**
  - To prevent unauthorized users from gaining access to resources,
  - To prevent legitimate users from accessing resources in an unauthorized manner
  - To enable legitimate users to access resources in an authorized manner



https://www.learn2torials.com/a/linux-access-control-list

Access Controls

- **Basic elements of access control**
  1. Subject
     - An entity capable of accessing objects
     - E.g., owner, group, world
  2. Object
     - A resource to which access is controlled
     - E.g., page, file, directory, message, program
  3. Access right
     - Describes the way in which a subject may access an object
     - E.g., read, write, execute, delete, create, search

Access Controls

- **Access controls in Unix file system**
  - File permissions with the user-group-others model
    - User
      - Indicating the userid (UID) of the file owner
    - Group
      - Indicating the groupid (GID) of the file
    - Others
      - Public

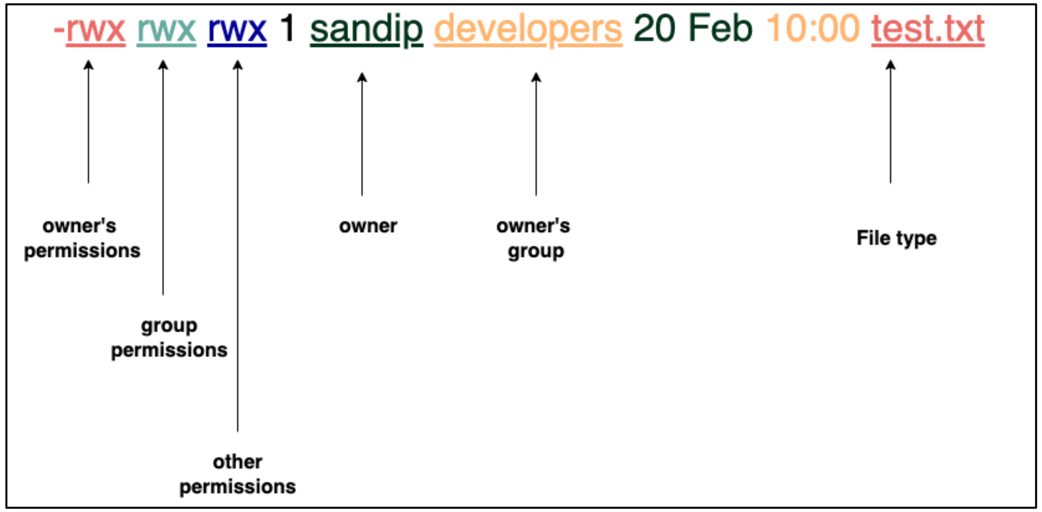
Access Controls

- **Access controls in Unix file system**
  - Three protection bits for each of user, group, and others
    1. Read (R)
       - The file contents can be read
    2. Write (W)
       - The file contents can be modified
    3. Execute (X)
       - A file can be run

Access Controls

- **Access controls in Unix file system**
  - Display for file permission: 10-char string

Type	User			Group			Others		
	R	W	X	R	W	X	R	W	X



https://en.wikipedia.org/wiki/File-system_permissions

Access Controls

- **Access controls in Unix file system**

Symbolic notation	Numeric notation	Description
----------	0000	no permissions
-rwx------	0700	read, write, & execute only for owner
-rwxrwx---	0770	read, write, & execute for owner and group
-rwxrwxrwx	0777	read, write, & execute for owner, group and others
---x--x--x	0111	execute
--w--w--w-	0222	write
--wx-wx-wx	0333	write & execute
-r--r--r--	0444	read
-r-xr-xr-x	0555	read & execute
-rw-rw-rw-	0666	read & write
-rwxr-----	0740	owner can read, write, & execute; group can only read; others have no permissions

https://en.wikipedia.org/wiki/File-system_permissions

Access Controls

- **Access control policies**
  1. Discretionary Access Control (DAC)
  2. Mandatory Access Control (MAC)
  3. Role-Based Access Control (RBAC)
  4. Attribute-Based Access Control (ABAC)

Access Controls

1. **Discretionary Access Control (DAC)**
   - Traditional method of implementing access control
   - Owner or administrator of resources grants access permissions to other users without the intervention of a security manager
   - Controls access based on the (1) identity of the requestor and on (2) access rules stating what requestors are (or are not) allowed to do
   - Easy to implement and simple to use, but not highly secure

Access Controls

1. Discretionary Access Control (DAC)

- Access matrix

OBJECTS

SUBJECTS		File 1	File 2	File 3	File 4
	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

Access Controls

2. **Mandatory Access Control (MAC)**
   - Enforcing restrictions on a low-security-level entity from accessing high-security-level objects
     - E.g., membership grading system in a cafe or community
       - Each member (subject) is assigned a security level
       - There are specified permission levels for accessing each bulletin board (object)
   - Even if one is the owner of an object, without being granted the security level to access that object, one cannot access it
   - Highly secure, but complex configuration
     - Access control cannot be applied differently for each subject
     - Permission levels must be set for all subjects and objects one by one

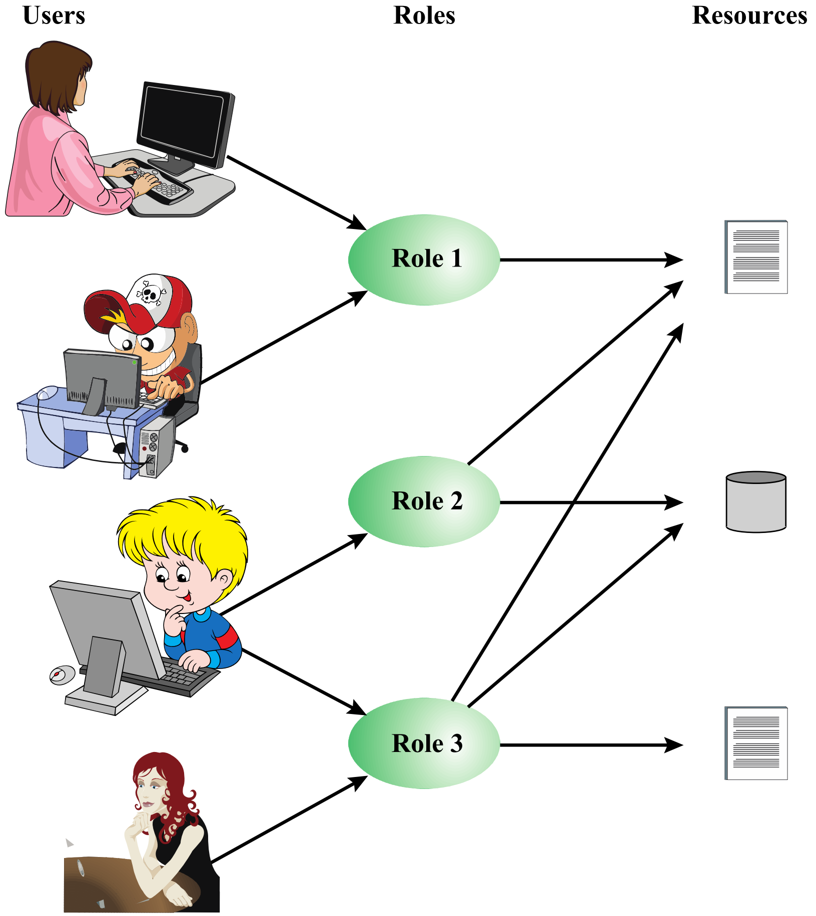
Access Controls

3. **Role-Based Access Control (RBAC)**
   - Granting permissions to role groups rather than individual users
   - Controlling access by assigning roles to users
   - Commonly used in services based on job roles

Access Controls

3. Role-Based Access Control (RBAC)

- Users assigned to different Roles according to their responsibilities
- Users-to-Roles are Many-to-Many
- Users may change frequently



Access Controls

4. **Attribute-Based Access Control (ABAC)**
   - Controlling access by describing conditions based on the attributes of objects and subjects
     - E.g., To access "File 1", users must have the "admin" tag attached to their type attribute
   - Attributes
     - E.g., Subject name, resource types, and current time
   - Typically used in conjunction with RBAC to manage permissions more finely

Access Controls

4. Attribute-Based Access Control (ABAC)

- Example


```
{
"bindings": [{
"role": "roles/testRole",
"members": [
"user:developer@s-core.co.kr"
],
"condition": {
"title": "DateTime Expires",
"description": "Expires at noon on 2021-12-31 UTC",
"expression": " request.time < timestamp(' 2021-12-31T12:00:00Z ')"
}
}]
```


https://www.samsungsds.com/kr/insights/cloud_platform_manage.html

Access Controls and Software Security

- **Improper access control can lead to software vulnerabilities**
  - CWE-22: Improper Limitation of a Pathname to a Restricted Directory (8)
  - CWE-264: Permissions, Privileges, and Access Controls
  - CWE-269: Improper Privilege Management (22)
  - CWE-284: Improper Access Control
  - CWE-285: Improper Authorization
  - CWE-862: Missing Authorization (11)
  - CWE-863: Incorrect Authorization (24)

* Highlighted numbers: the rankings of the top 25 most dangerous Common Weakness Enumeration (CWE) entries in 2023

Access Controls and Software Security

- **Improper access control can lead to software vulnerabilities**
  - Example: CVE-2023-4696

```
5 server/jwt.go
@@ -111,6 +111,11 @@ func JWTMiddleware(server *Server, next echo.HandlerFunc, secret string) echo.Ha
 				}
 				return nil, errors.Errorf("unexpected access token kid=%v", t.Header["kid"])
 			})
+
+			if !accessToken.Valid {
+				return echo.NewHTTPError(http.StatusUnauthorized, "Invalid access token.")
+			}
+
 			if !audienceContains(claims.Audience, auth.AccessTokenAudienceName) {
 				return echo.NewHTTPError(http.StatusUnauthorized, fmt.Sprintf("Invalid access to
 			}
```

Access Controls and Software Security

- **Privilege escalation**
  - Gaining unauthorized permissions within a system, network, or application
    - E.g., gain root privileges
  - This can be achieved by exploiting vulnerabilities to bypass security measures that prevent the user from accessing certain types of information

Access Controls and Software Security

- **Privilege escalation**
  - Vertical
    - An attempt to access the highest level account from the lowest level privileged account in a multi-level privilege structure
  - Horizontal
    - An attempt to elevate privileges and moves laterally to access the functions or data of another user at the same level

Access Controls and Software Security

- **Privilege escalation**
  - Demo: Windows privilege escalation attack
    - Exploiting CVE-2017-0213 vulnerability
    - https://www.youtube.com/watch?v=f6x0hBerObM

Next Lecture

- **Software vulnerabilities**