Foundations of Cybersecurity

Fall 2024

Final Exam

Due Date: 12/03/2024 Points: 100

Student Registration # 035 Student Name: Christian Carrion

NOTE:

- Please carefully read the instructions after each question.
- Before executing **`CLI/Terminal/Bash`** command(s), change the prompt with your First Name and registration Number (**`bashrhc PS1="First_Name_Reg#")`**.

Q.1. How Access Control Matrix (ACM) represent the Access Control List (ACL) and *Capability List*? Explain the answer using the following ACM. **[15]**

User/Group	XYZ1.txt	XYZ2.txt	/usr/ABC
Owner (VMASC)	rwx	r--	rwx
Group (ODU)	r--	rw-	r-x
Others	r--	r--	r-x

Access Control Matrix is a 2D table where the row represents users or users' group. The columns represent system files. Access Control is a list of attached to each resource which specifies which users or groups have access to, and the type of access they have. A capability list is a list associated with each group that shows the specific resources they can access. Also, it shows the type of access they have. Each group has a list of their own permission.

ACM to ACL:

XYZ1.txt

1. **Owner(VMASC): Read, Write, Execute**
2. **Group(ODU): Read Only**
3. **Others: Read only**

XYZ2.txt

1. **Owner(VMASC): Read only**
2. **Group(ODU): Read and Write**
3. **Others: Read only**

/usr/ABC

1. **Owner(VMASC): Read, Write, Execute**
2. **Group(ODU): Read and Execute**
3. **Others: Read and Execute**

ACM as Capability List:

VMASC user:

1. **XYZ1.txt: Read, Write, Execute**
2. **WXY2.txt: Read only**
3. **/usr/ABC: Read, write, Execute**

Group (ODU):

1. **XYZ1.txt: Read only**
2. **XYZ2.txt: Read and Write**
3. **/usr/ABC: Read and Execute**

Others:

1. **XYZ1.txt: Read only**
2. **XYZ2.txt: Read only**
3. **/usr/ABC: Read and Execute**

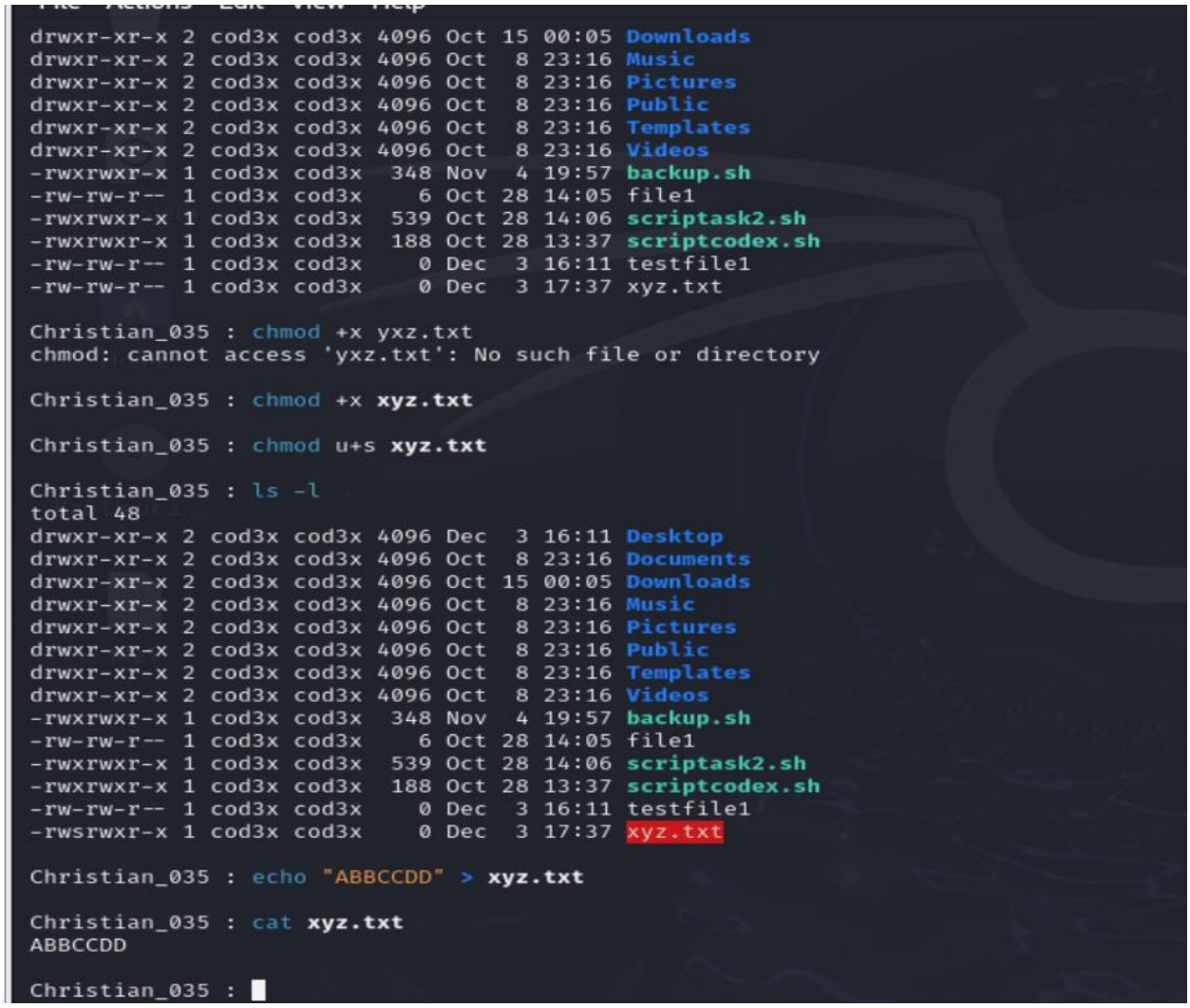
Q.2. What is the privilege program and what are the vulnerabilities that it poses during the execution? Create a file "**`xyz.txt`**" using the following command and change its set-UID bit to HIGH. Show the "**`xyz.txt`**" file attributes using "**`ls -l`**" command before and after the set-UID enabling. **[15]**

A privilege program is a program or process that runs with elevated privileges. They have more capabilities than regular privileged programs. Privilege programs can raise potential vulnerabilities such:

1. **Race Conditions**
2. **Buffer Overflows**
3. **Privilege Escalation**

[Include Screenshots of the Terminal / CLI]

`echo "ABBCCDDD" > xyz.txt`



Q.3. What type of vulnerabilities are presented by the following code? **[10]**

```
-----------------------------------------------
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int main() {
    char* v[2];
    int fd = open("/etc/zzz", O_RDWR | O_APPEND);
    if (fd== -1) {
        printf("Cannot Open /etc/zzz \n");
        exit(0);
    }
    setuid(getuid());

    v[0]= "/bin/sh"; v[1] = 0;
    execve(v[0],v,0);
    return 0;
}
```

There are two main vulnerabilities that I see in this code.

1. **Insecure file Access, which is potential for race condition:**
   - This code opens the file /etc/zzz for read and write (which is 0_RDWR) with the flag 0_APPEND. The vulnerability is that the file path /etc/zzz is accessed without validation or security measures. This is where race conditions can take place. If an attacker could manipulate the contents of the file of /etc/zzz between the time it's checked and time it is opened, it can give the attacker an opportunity to cause abnormal behavior. This is known as Time-of-check to time-of-use vulernability.

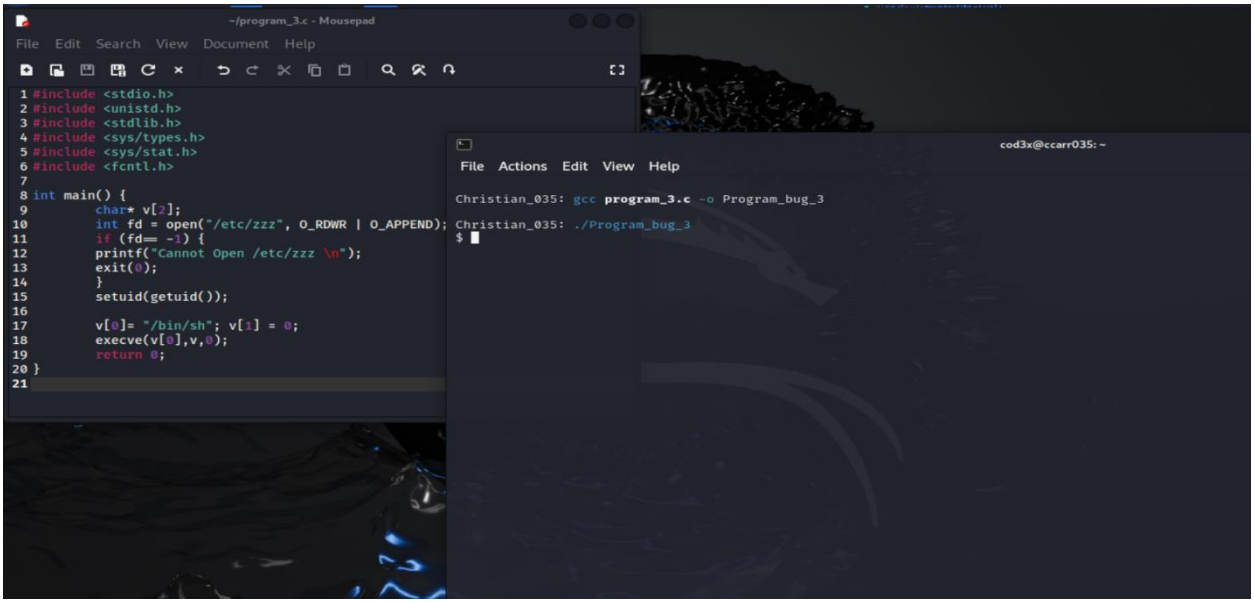
2. Command Injection

- The line of code execve(v[0], v,0);executes the /bin/shell. This has the potential to introduce command injections. For example, of an attacker could manipulate the content of v[0] , they could potentially force the program to execute arbitrary shell commands. This means this could lead to remote code execution.

Q.4. What is the format string vulnerability and how it can be exploited? Explain your answer with example code and the screenshot of the code execution output in the terminal?**[15]**

A format string vulnerability occurs when an attacker can manipulate format string used in function like printf(). This this case, the coding in Q.3 has the function printf(). The attacks can control the format specifiers like %s or %d, which can lead to code execution, stack corruption or memory leaks.

After I executed the code file, it will tell me I can not open /etc/zzz. However, for some unknown reasons, when I execute the file nothing happens.

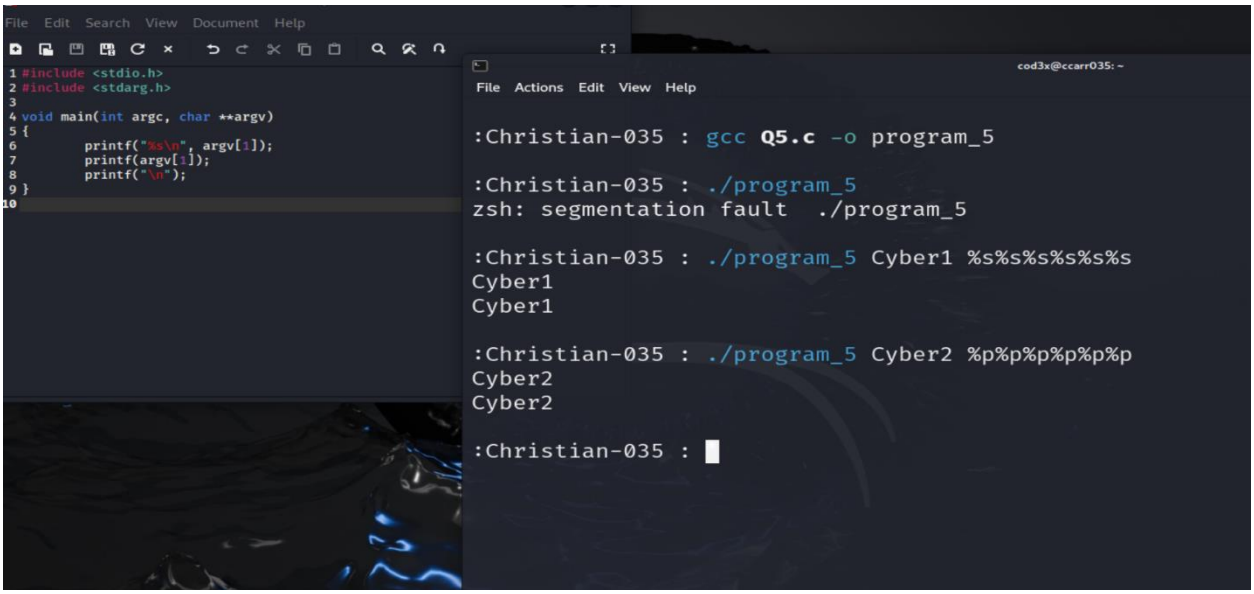


Q.5. What vulnerabilities the following code represents and what will be its output when it is passed with the command line arguments of "**Cyber1 %s%s%s%s%s%s**" and "**Cyber2 %p %p %p %p %p %p**"? **[15]**
[Include Screenshots of the Terminal Window / CLI]

```
-------------------------------------------------
#include <stdio.h>
#include <stdarg.h>

void main(int argc, char **argv)
{
    printf("%s\n", argv[1]);
    printf(argv[1]);
    printf("\n");
}
```

Vulnerabilities found in code are buffer Overflow and Stack Corruption.



Q.6. What vulnerabilities the following code represents and what will be its output when it is passed with the command line arguments of "**Cyber3 %p %p %p %p %p %p**" and "**Cyber4 %x %x %x %x %x %x**" ? **[15]**

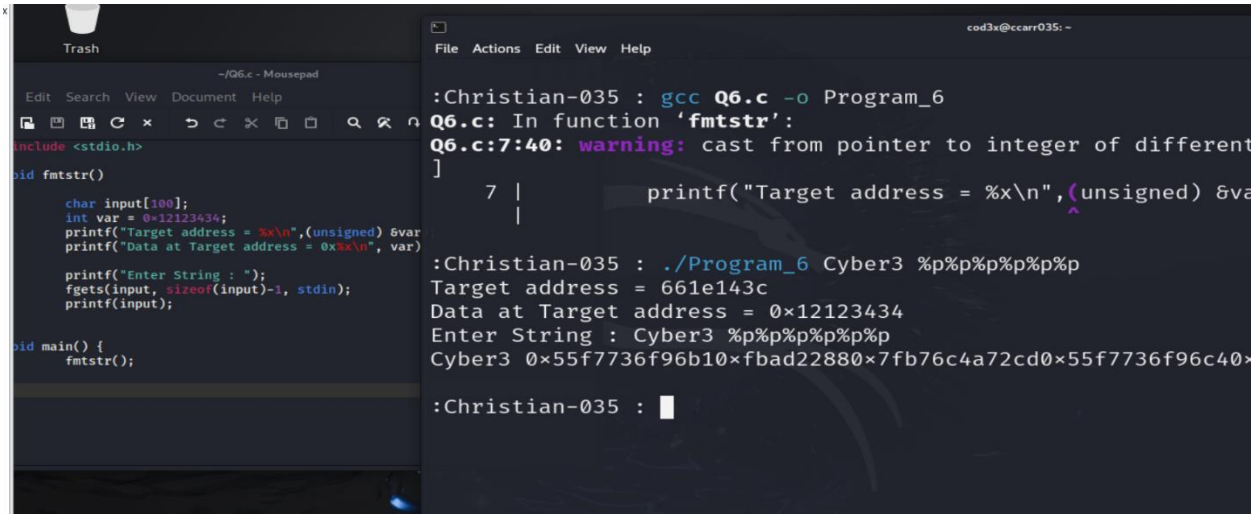
[Include Screenshots of the Terminal Window / CLI]

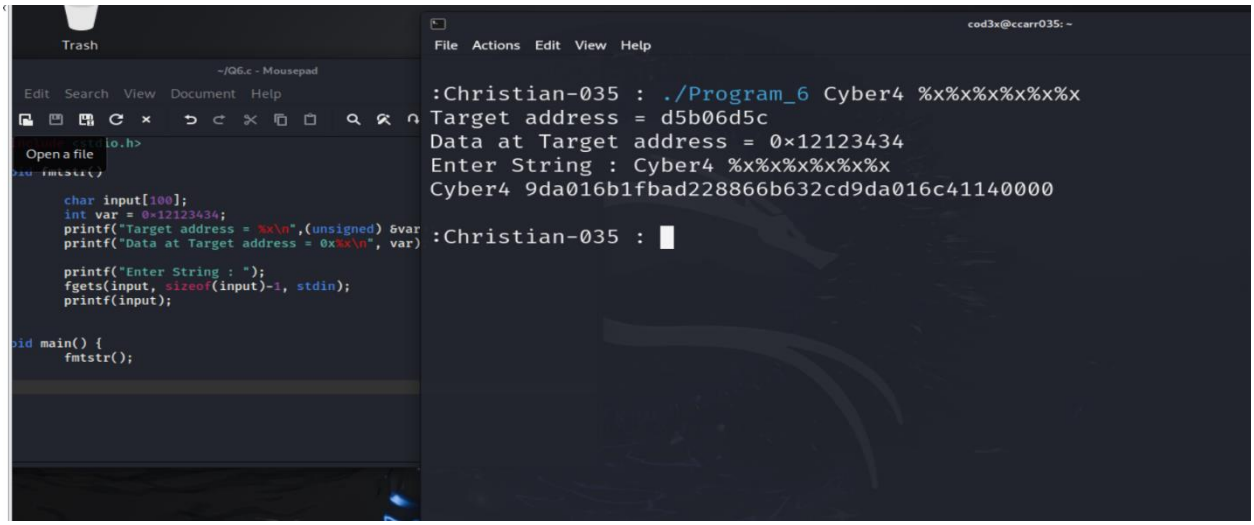
```
-------------------------------------------------
#include <stdio.h>

void fmtstr()
{
    char input[100];
    int var = 0x12123434;
    printf("Target address = %x\n",(unsigned) &var);
    printf("Data at Target address = 0x%x\n", var);

    printf("Enter String : ");
    fgets(input, sizeof(input)-1, stdin);
    printf(input);
}

void main() {
    fmtstr();
}
-------------------------------------------------
```





The vulnerabilities that can be found in this code are Format string vulnerability, memory corruption, and lack of proper input validation.

Q.7. Consider a scenario where two processes, A and B, need to update a shared file in a Unix system. Process A writes to the file first, followed by Process B. Describe a potential race condition that could occur in this scenario. Explain how you would use Unix system calls to prevent this race condition, ensuring that the file updates are performed correctly and without interference. **[15]**

In this scenario, if both A and B process simultaneously attempted to access or modify the file, this could lead to a potential race condition. Here is an example in how a race condition could occur. Lets say that process A started writing data to the file. In this case it is trying to write "Hello". At the same time Process B also starts to write data to the file. In this case let's say that process B is writing "world" before Process A completed its operation. If this was the case the final contents of the file could become corrupt. This is because A and B processes overlap overtime. With

no proper mechanisms, there is a chance A process and B process and interfere with one another.

Have preventive measure that could solve this issue is by using the flock() system call. This system call is used to acquire a lock on a file. There are types of locks:

1. **Shared lock (LOCK_SH)**
2. **Exclusive Lock (LOCK_EH)**

An example of how flock() works is LOCK_EX would lock the file exclusively. This means that no other file can read or write until the lock as been released. If process A has completed writing data into the file, It would then call flock system call to release the lock. Than Process B can acquire the lock, and could then proceed to write into the file.