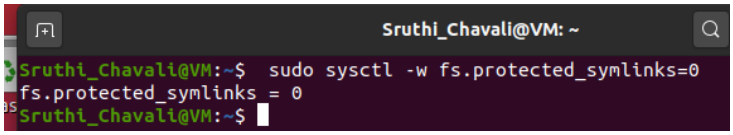
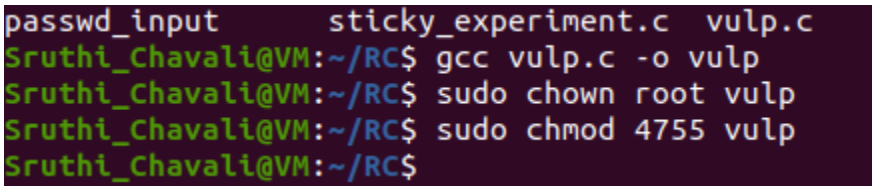
Task 0

For the set up of this, we need to turn off the symlink protection. This protection is in ubuntu 10.0 and later and it works by symlinks in world-writable sticky directories (e.g. /tmp) cannot be followed if the follower and directory owner do not match the symlink owner.

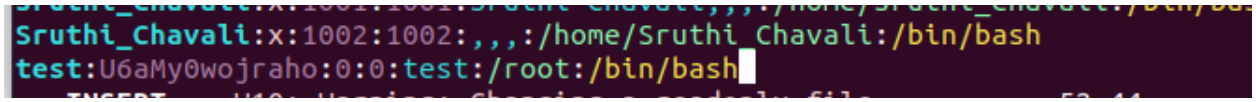


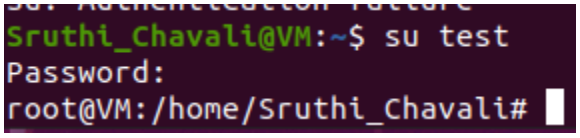
The next thing we need to do is compile the vulnerable program, vulp.c. We then change the owner to root and then change the permissions to 4755 so that the user can read, write, and execute.

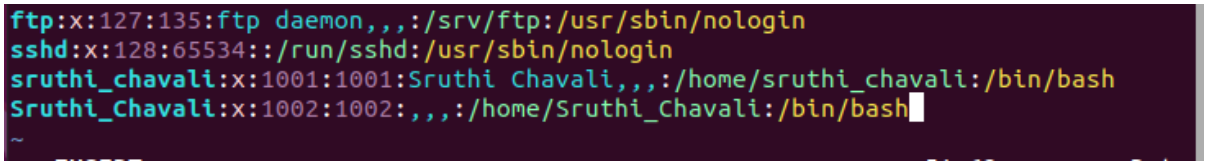


Task 1

In this task, we add the magic value which is dees, the password to the test user to the /etc/passwd file as a root user. I then logged in as the test user to see if this worked. Lastly, I removed the magic value from the /etc/passwd file.



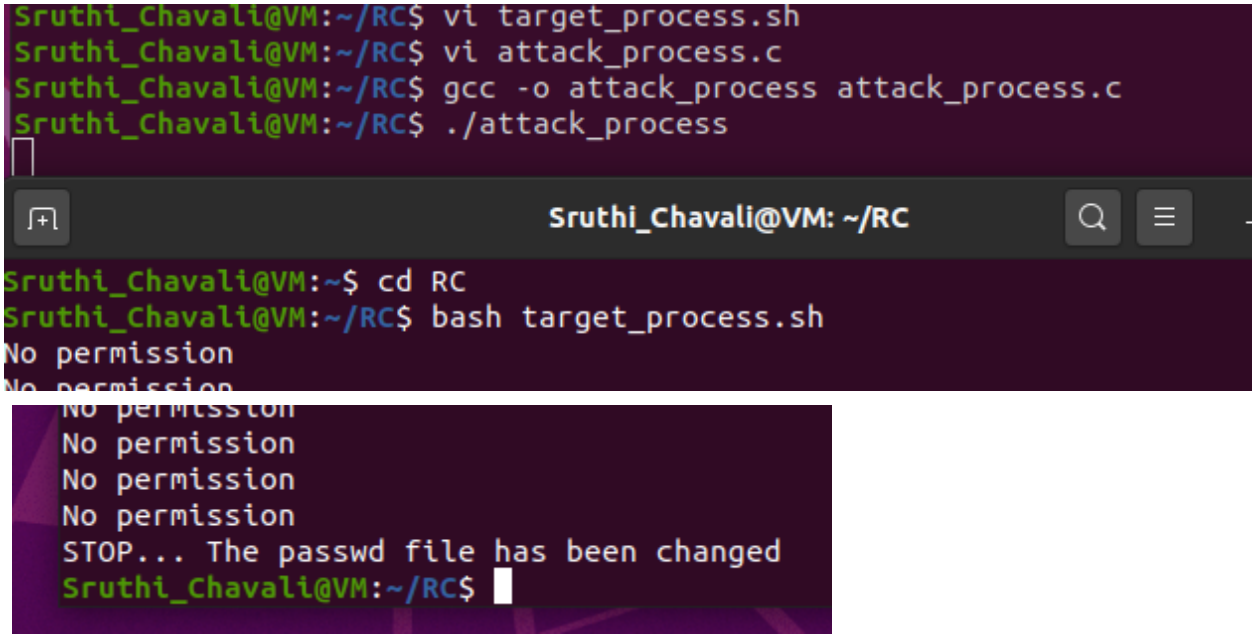




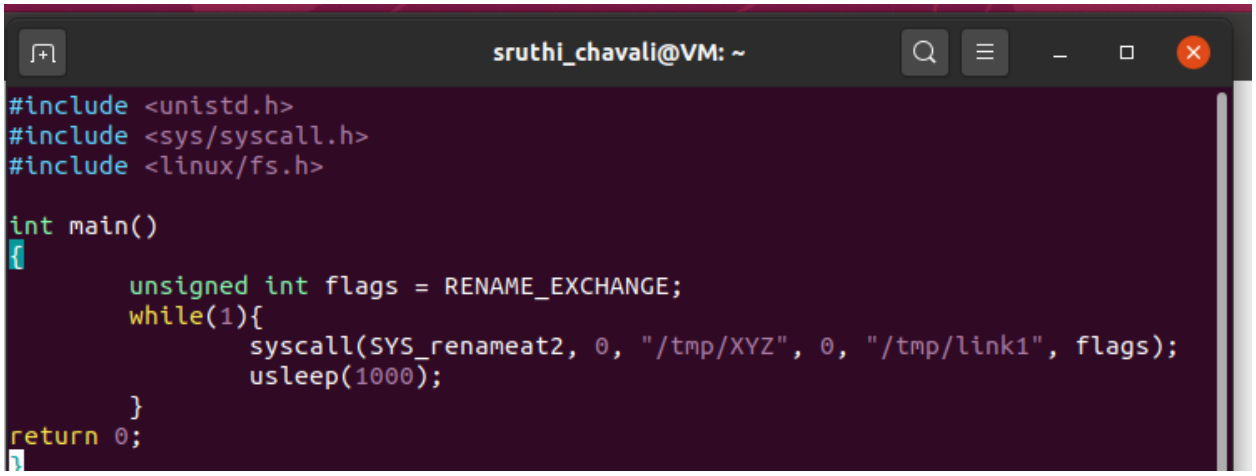
Task 2

In this task, you compile and run the attack program. In another terminal window, run the target_process.sh using bash. After getting No Permission on a loop, if the attack is successful, it

will end with Stop….the passwd file has been changed.



Next, we attempt to redo the attack without using the help of root as done in the original attack_process.c. The /tmp folder has a sticky bit which means only the owner of the file can delete it even though it is world writable. The unlink() and symlink() in the original are unatomic. In order to counteract this problem, we rewrite the attack program so that the calls are atomic.



After doing this we run the attack the same way we did it above. This results in

```
Sruthi_Chavali@VM:~/RC$ rm attack_process
Sruthi_Chavali@VM:~/RC$ gcc -o attack_process attack_process.c
Sruthi_Chavali@VM:~/RC$ ./attack_process
```

Sruthi_Chavali@VM: ~/RC

```
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
STOP... The passwd file has been changed
Sruthi_Chavali@VM:~/RC$
```

```
No permission
No permission
STOP... The passwd file has been changed
Sruthi_Chavali@VM:~/RC$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
Sruthi_Chavali@VM:~/RC$ id
uid=1002(Sruthi_Chavali) gid=1002(Sruthi_Chavali) groups=1002(Sruthi_Chavali),27
(sudo)
Sruthi_Chavali@VM:~/RC$ su test
Password:
root@VM:/home/Sruthi_Chavali/RC# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/Sruthi_Chavali/RC#
```

showing that the attack was successful.

Task 3

The purpose of this attack is to include the principle of least privilege. The programer tries to control permissions to the user using attack() but instead we have to make sure that if a user does not need a permission, it does not get that permission. This is the following update.

```c
#include <unistd.h>
#include <stdio.h>
#include <string.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);
    setuid(realUID);
   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");

   return 0;
-- INSERT --                                                    1
```

After running this attack program,

```
No permission
No permission
target_process.sh: line 10: 14250 Segmentation fault      ./vulp < passwd_input
No permission
target_process.sh: line 10: 14256 Segmentation fault      ./vulp < passwd_input
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
target_process.sh: line 10: 14290 Segmentation fault      ./vulp < passwd_input
```

we realize that the attack was not successful. This is because making this update makes the vulnerable program not vulnerable anymore.

Task 4

In this task, we turn on the symlinks protection that we disabled in task 0.

```
Sruthi_Chavali@VM:~/RC$ vi vulp.c
Sruthi_Chavali@VM:~/RC$ rm vulp
rm: remove write-protected regular file 'vulp'? y
Sruthi_Chavali@VM:~/RC$ gcc vulp.c -o vulp
Sruthi_Chavali@VM:~/RC$ sudo chown root vulp
Sruthi_Chavali@VM:~/RC$ sudo chmod 4755 vulp
Sruthi_Chavali@VM:~/RC$ sudo sysctl -w fs.protected_symlinks = 1
sysctl: "fs.protected_symlinks" must be of the form name=value
sysctl: malformed setting "="
sysctl: "1" must be of the form name=value
Sruthi_Chavali@VM:~/RC$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
```

```
Sruthi_Chavali@VM: ~/RC
No permission
No permission
No permission
target_process.sh: line 10: 17149 Segmentation fault      ./vulp < passwd_input
target_process.sh: line 10: 17151 Segmentation fault      ./vulp < passwd_input
target_process.sh: line 10: 17153 Segmentation fault      ./vulp < passwd_input
No permission
target_process.sh: line 10: 17157 Segmentation fault      ./vulp < passwd_input
target_process.sh: line 10: 17159 Segmentation fault      ./vulp < passwd_input
target_process.sh: line 10: 17161 Segmentation fault      ./vulp < passwd_input
No permission
No permission
No permission
```

As shown, this does not result in a successful attack.

How does this protection scheme work?

When the sticky symlink prevention is enabled, symbolic links inside a sticky world-writable directory can only be followed when the owner of the symlink matches the follower or the owner. Since the program has root privilege and the /tmp directory is also owned by the root, the program will not be allowed to follow any symbolic link that is not created by the root. The symbolic link was created by the attacker or the seed. Which causes this link to not be followed, and that leads to an unsuccessful attack.

What are the limitations of this scheme?

This protection scheme does not stop the race condition from happening, but controls the damages. Also, this protection applies only to word-writable sticky directories such as /tmp and nothing else.