

## **Lab2 - Öryggi Tölvukerfa**

Rafnar Ólafsson

**Task 1- Shell Exploration**

**1. Following are common Set-UID programs in Unix, *passwd*, *chsh*, *su*, and *sudo*. Will these programs still be Set-UID programs when you copy these programs to your own directory? Please perform the copy operations and report your Observations**

Copying *passwd* to */tmp/*, it lost root's privileges. As for *chsh*, *su* and *sudo*, they are the same.

Meaning, we can use this tmp *passwd* we can change our password, however any other program will not function, having lost their Set-UID bit.

**2. If you copy the file */bin/zsh* to */tmp/zsh*, report the permissions of the file */tmp/zsh*? Report who is the owner of the file.**

The user root is the owner of *zsh* before moving the file, but after copying the file the owner is now seed. Additionally, all users have all accesses to the *zsh* file in its original *bin* folder, but members and others lose write access after copying the program.

3. Login as root, copy /bin/zsh to /tmp, and make it a set-root-uid program with permission 4755. Then login as a normal user, and run /tmp/zsh. Will you get root privilege? Please describe your observation. Please use proper Unix commands to find who you are, your UID, EUID to report your observation.

```
root@VM: /tmp
[03/18/19]seed@VM:~$ sudo su
[sudo] password for seed:
root@VM:/home/seed# cp /usr/bin/zsh /tmp/
root@VM:/home/seed# chmod u+s zsh
chmod: cannot access 'zsh': No such file or directory
root@VM:/home/seed# cd /tmp/
root@VM:/tmp# chmod u+s zsh
root@VM:/tmp# ls
config-err-fs2wjs
config-err-W8ReJ4
systemd-private-3ece830e3e0e456a853e414f0a62a524-color
systemd-private-3ece830e3e0e456a853e414f0a62a524-rtkit
unity_support_test.1
zsh
root@VM:/tmp# ls -al zsh
-rwsr-xr-x 1 seed seed 756476 Mar 18 16:44 zsh
root@VM:/tmp# whoami
root
root@VM:/tmp# id -u root
0
root@VM:/tmp# id -u
0
root@VM:/tmp# id -u -r
0

root@VM:/tmp# exit
exit
[03/18/19]seed@VM:~$ cd tmp
bash: cd: tmp: No such file or directory
[03/18/19]seed@VM:~$ cd /tmp/
[03/18/19]seed@VM:/tmp$ ./zsh
VM% id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
VM% 
```

Normal users now get new access.

**4. Repeat above procedure with /bin/bash instead of /bin/zsh. Copy /bin/bash to /tmp, make it a set-root-uid program. Run /tmp/bash as a normal user. Will you get root privilege? Please describe and explain your observation.**

We end up with the same response. We do not gain root privileges.

**Task 2 – System Command Vulnerabilities**

**1. Type the below code in systemCall.c**

**// Lots of code**

**2. Compile and run it using the following command**

**gcc -o systemCall systemCall.c**

**./systemCall**

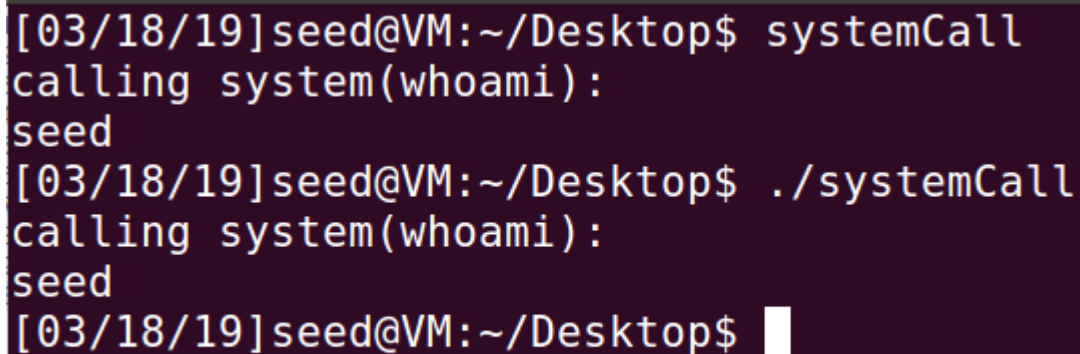
**Report your observations**

Here we call our User. Nothing seems out of the ordinary.

**3. Run the systemCall as follows**

**systemCall Report your observations and explain the differences between this step and the previous step.**

Screenshot:



```
[03/18/19]seed@VM:~/Desktop$ systemCall
calling system(whoami):
seed
[03/18/19]seed@VM:~/Desktop$ ./systemCall
calling system(whoami):
seed
[03/18/19]seed@VM:~/Desktop$
```

Current directory has been included in the PATH, as such I cannot create an error here. Otherwise, an error would've occurred.

**Task 3 – Exploiting PATH Environment Variable**

We will be using the shell zsh for this part of the task. You need to create symbolic link for /bin/sh to point to /bin/zsh and change the default shell to zsh. Lets assume that below is a Set-UID program. The program is supposed to execute /bin/ls command. However the programmer used the relative path.

```
int main()
{
    system("ls"); return 0;
}
```

**1. Demonstrate if you can allow the above Set-UID program owned by the root to run your code instead of /bin/ls. If you are successful, is your code running with root privileges. Explain your observation. In order to do this task, you need to perform the following actions:**

**a) Create above program as Set-UID program and test to see if the results match the intent of the program.**

My lsPhony file now has the Set-UID privileges.

**b) Create your own program called ls in the current directory and test to make sure the program ls does not have root permissions. Obviously, your program ls should have an operation different than the /bin/ls**

I create the phony ls program which will print out the string 'PATH is compromised!'

**c) Make the necessary changes to the PATH variable such that your program ls is called when you run the above Set-UID program. Report your observations.**

When we run the ls program we use my malicious program rather than the normal ls program. We print out the string 'PATH is compromised!'

**d) Report if your code is running with root privileges or not.**

My code is not running on root privileges.

**2. Now, change /bin/sh so it points back to /bin/bash, and repeat the steps. Can you still get the root privilege? Explain your observations.**

Repeating the steps for /bin/bash ends up with the same result, as bash runs on Set-UID permission as well. My code is still not running on root privileges.

**Task 4 - Exploiting Set-UID program**

Please make sure /bin/sh is pointing back to /bin/zsh to conduct the task Here.

**(a) Set q = 0 in the program. This way, the program will use system() to invoke the command.**

**Is this program safe? If you were Bob, can you compromise the integrity**

**of the system? For example, can you remove any file that is not writable to you? (Hint: remember that `system()` actually invokes `/bin/sh`, and then runs the command within the shell environment. We have tried the environment variable in the previous task; here let us try a different attack. Please pay attention to the special characters used in a normal shell environment).**

With this program, we are able to invoke the Set-UID bit which allows us to run the program as root, the program is not safe and Bob can compromise the system by removing files not writable by him. I was able to remove a file I only had read access to using this program. This is because of the fact that `System()` invokes `bin/sh` which has root privileges.

**(b) Set `q = 1` in the program. This way, the program will use `execve()` to invoke the command. Do your attacks in task (a) still work? Please describe and explain your observations.**

When we modify `q`, `execve()` executes a folder via filename, with no special privileges as it is not called from `/bin/sh` like in our previous part. This ends up not making sense, and has no effect.

**Task 5 – LD PRELOAD environment variable.**

To make sure Set-UID programs are safe from the manipulation of the LD PRELOAD environment variable, the runtime linker (ld.so) will ignore this environment variable if the program is a Set-UID root program, except for some conditions. We will figure out what these conditions are in this task.

Please run myprog under the following conditions, and observe what happens. Based on your observations, tell us when the runtime linker will ignore the LD PRELOAD

environment variable, and explain why.

- Make myprog a regular program, and run it as a normal user.

When we run the program as a normal user, mylib PATH overrides the sleep function.

- Make myprog a Set-UID root program, and run it as a normal user.

When we run the program as a normal user after giving it the Set-UID bit, we ignore the environment PATH and as such, *sleep()* is not overridden.

- Make myprog a Set-UID root program, and run it in the root account.

In this situation, root will use the PATH env. Variable and override the *sleep()* function.

- Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), and run it as a different user(not-root user).

Here we are unable to compile the program, as the user does not have permission to do so.

```
[03/19/19]seed@VM:~/Desktop$ sudo su
root@VM:/home/seed/Desktop# useradd -d /usr/user1 -m user1
root@VM:/home/seed/Desktop# su user1
user1@VM:/home/seed/Desktop$ export LD PRELOAD=./libmylib.so.1.0.1
user1@VM:/home/seed/Desktop$ gcc -o myprog myprog.c
myprog.c: In function 'main':
myprog.c:4:1: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
  sleep(1);
  ^
/usr/bin/ld: cannot open output file myprog: Permission denied
collect2: error: ld returned 1 exit status
user1@VM:/home/seed/Desktop$ chmod u+s myprog
chmod: changing permissions of 'myprog': Operation not permitted
```



**Task 6 – Capability Leak**

**In the class, we looked at a example of capability leak. Here is another example of the same issue. Compile the following program, and make the program a SETUID root program. Run it in a normal user account, and describe what you have observed. Will the file /etc/zoo be modified? Please explain your observation**

When we open the file we mimic the tasks conducted by the program. We then remove the root privileges and create two processes. We don't need the parent process so we close the file in that process. Meanwhile, we attack the file and modify it before closing it. We have now compromised the file without the needed privileges.