**Lab 7**

**Kaleb Alstott**

**Lab-Setup**

A screenshot of a computer

Description automatically generated

**DNS Setup:** Similar to Lab5 and Lab6 we had to first map the names of the web server to the correct IP address. The server container’s IP address is 10.9.0.5. The hostname of the server is called [www.seed-server.com](http://www.seed-server.com) and is mapped to 10.9.0.80 .

A screenshot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

The first line here is setting up our DNS that is described above.

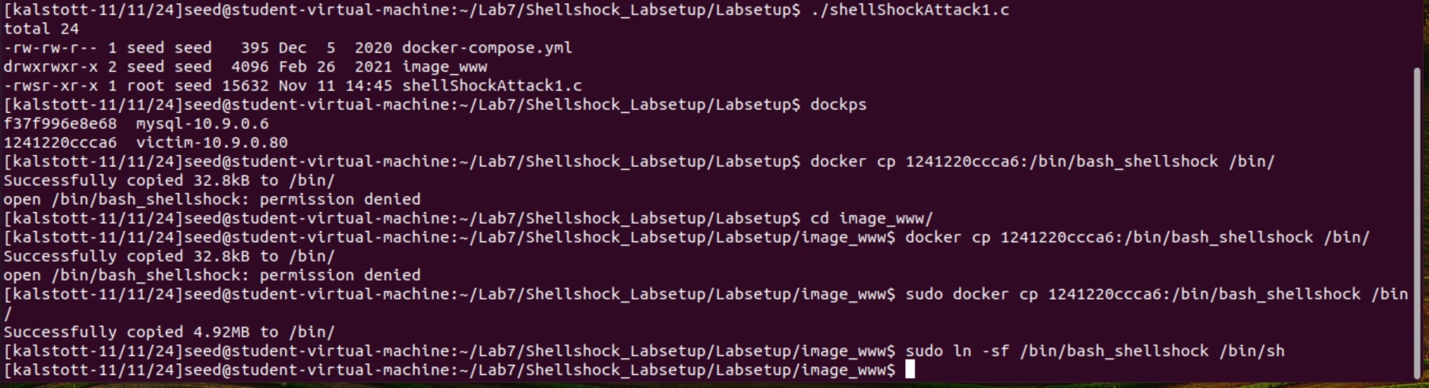
In order to start the lab, we must create our container using Docker Compose commands. We use a shortcut for docker-compose build, dcbuild, which builds the container images specified in the docker-compose.yml file. After this step is done, we want to use the shortcut for docker-compose up, dcup, which starts up all the containers specified in the docker-compose.yml

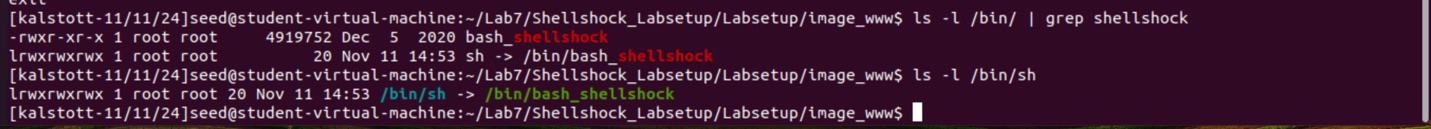
// Aliases for the Compose commands above  
$ dcbuild # Alias for: docker-compose build  
$ dcup # Alias for: docker-compose up

A screenshot of a computer

Description automatically generated

Showing that the web server and CGI working like described in the lab. This is to show that our web server is running and working correctly. The CGI program being used here is our vul.cgi program.





We need to be able to use the vulnerable bash shell (bash shellshock) in this lab. To do so we have to copy the file /bin/bash\_shellshock /bin/ from the container to the host machine, and link it to /bin/sh. This is done so by the following commands in the screenshot above with the cp and sudo ln -sf.

Once this is copied over, we are ready to rock and roll on this assignment!

A black background with white text

Description automatically generated

**NOTE:** I had to change files to ensure that I could finish the setup here. I had incorrectly created the wrong file from earlier screenshots. My mistake was with my c file initially and how I gcc the file. With this done correctly I had finished the setup correctly and was able to proceed and finish the tasks that were assigned. I just wanted to show and take note in case of any questions or concerns with tasks later on in this lab.

**TASK 2** **Passing Data to Bash via Environment Variable**

Our goal here in this task is exploit a Shellshock vulnerability in a bash-based CGI program. To do so we need to pass the attackers data to the vulnerable bash program, and the data need to be passed via an environment variable. We already have a preset CGI program (getenv.cgi) on the server to help us identify what user data can get into the environment variables of a CGI program. We will take a deeper in depth look below first with using a web browser and second by using the “curl” command.

**Using browser:**

**A screenshot of a computer

Description automatically generated**

**My environment variables on the web browser**

**A screenshot of a computer

Description automatically generated**

**Lab document environment variables**

As described for this first part in task 2, we have toidentify what user data can get into the environment variables of a CGI program via web server. This part was super easy and all we had to do was make sure we were running our HTTP live header extension that we downloaded from Lab5 and plug in our URL in correlation to our web server and program that is helping us do this task.

As you can see in our screenshot, we were successful in matching the lab document environment variables via web browser. We can see that the host, user-agent, and accept are all environment variables that we were able to retrieve from this web browser.

**Using curl first command – worked**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

The command-line tool called curl allows users to control most fields in an HTTP request. This will be helpful in this task because we will want to set the environment variable data to arbitrary values, so we control and exploit environment variables leading us to our shellshock vulnerability.

In our first command “curl -v [www.seed-server.com/cgi-bin/getenv.cgi](http://www.seed-server.com/cgi-bin/getenv.cgi)”, shows us the default environment variables set by the browser. In our case we have a lot of similarity from what we saw in the web browser version of this task. Such environment variables that are similar are host, user-agent, and accept. We get more of a peek at the environment variables that are set initially by the web browser such as the server address, port, server name, and a lot more.

**NOTE:** “Here are some of the userful options: (1) the -v field can print out the header of the HTTP request; (2) the -A, -e, and -H options can set some fields in the header request, and you need to figure out what fileds are set by each of them.”

-A: Modifies the HTTP\_USER\_AGENT environment variable.

-e: Modifies the HTTP\_REFERER environment variable.

-H: Adds custom headers, which are converted into new environment variables

**Using second command - worked**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

In our second command “curl -A "my data" -v www.seed-server.com/cgi-bin/getenv.cgi”, shows us the default environment variables set by the browser as well as sets the HTTP\_USER\_AGENT variable with what we have specified in the quotations. In our case -A is used to modify the HTTP\_USER\_AGENT environment variable, in which we have successfully modified this environment variable with our data “my data”. We can confirm that this has been done correctly by finding “my data” set to the environment variable of HTTP\_USER\_AGENT, which is the second line of the “environment variables”.

**Using third command - worked**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

In our third command “curl -e "my data" -v www.seed-server.com/cgi-bin/getenv.cgi”, shows us the default environment variables set by the browser as well as sets the HTTP\_REFERER variable with what we have specified in the quotations. In our case -e is used to modify the HTTP\_REFERER environment variable, in which we have successfully modified this environment variable with our data “my data”. We can confirm that this has been done correctly by finding “my data” set to the environment variable of HTTP\_REFERER, which is the fourth line of the “environment variables”.

**Using fourth command -worked**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

In our fourth command “curl -H "AAAAAA: BBBBBB" -v www.seed-server.com/cgi-bin/getenv.cgi”, shows us the default environment variables set by the browser as well as creates a new environment variable with what we have specified in the quotations. In our case -H is used to add custom headers, which are converted into new environment variables. We have successfully added our custom header which is converted into a new environment variable with what we had specified in quotations, which for us is “OOOOOOOO: KKKKKKK”. We can confirm that this has been done correctly by finding “OOOOOOO: KKKKKKKK” set as an environment variable, which is the fourth line of the “environment variables”.

**Based on this experiment, please describe what options of curl can be used to inject data into the environment variables of the target CGI program.**

The options -A, -e, and -H in curl can be used to inject arbitrary data into environment variables of a CGI program.

**Task 3: Launching the Shellshock Attack**

Our goal for this task is to launch our shellshock attack. Our job is to launch the attack through the URL http://www.seed-server.com/cgi-bin/vul.cgi, so we can get the server to run arbitrary commands. As specified, we will use three different approaches (i.e., three different HTTP header fields) to launch the Shellshock attack against the target CGI program. The approaches you will see are as follows:

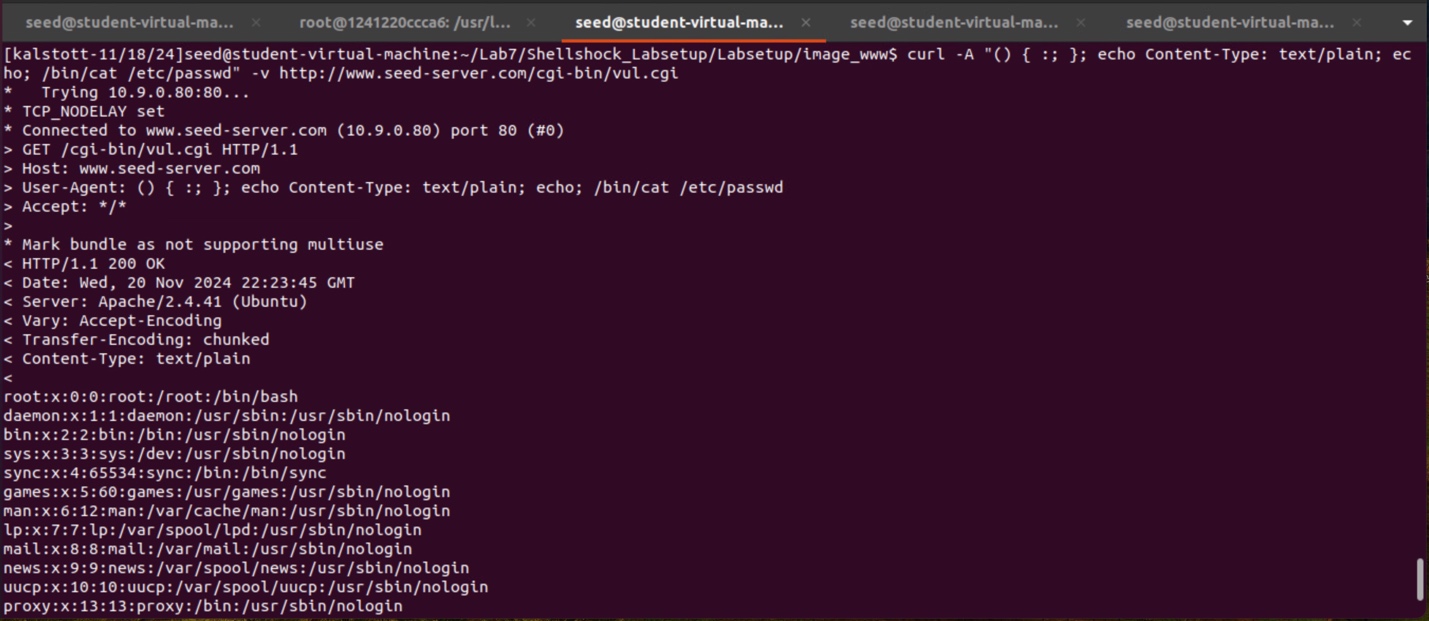
-A: Modifies the HTTP\_USER\_AGENT environment variable.

-e: Modifies the HTTP\_REFERER environment variable.

-H: Adds custom headers, which are converted into new environment variables

We will use the specific curl options of -A, -e, -H to inject data into different HTTP headers because these headers get translated into environment variables when processed by the CGI script on the server.

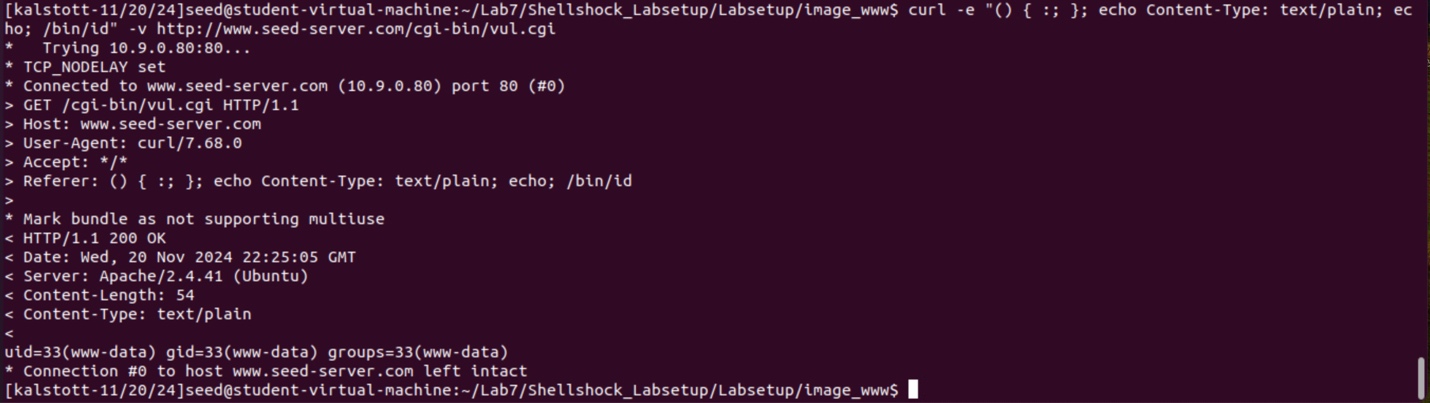
**Task 3.A: Get the server to send back the content of the /etc/passwd file.**

****

****

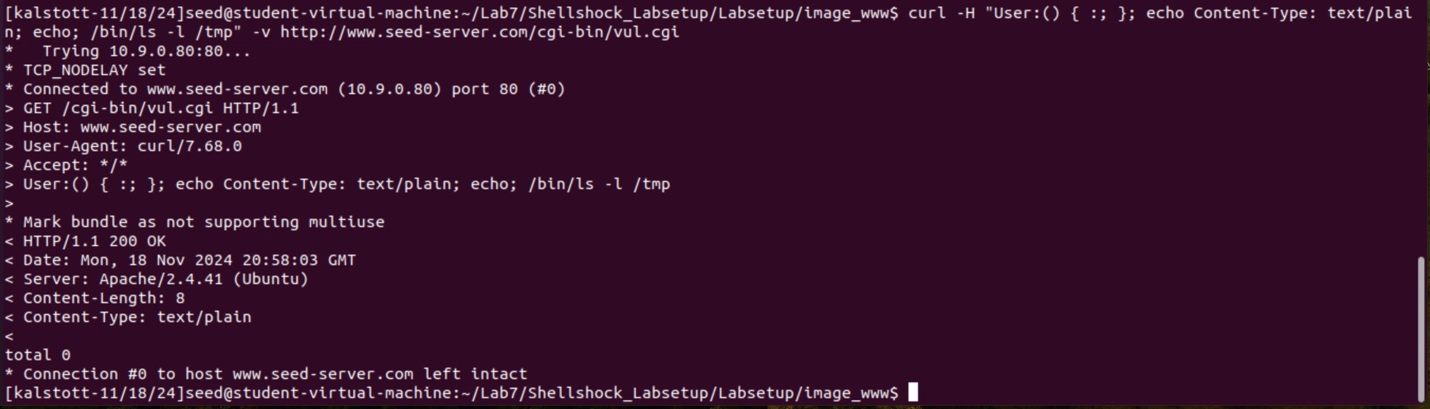
In the first part of task 3 we are challenged to get the server to send back the content of the /etc/passwd file. In this task I had chosen to use the curl -A command due to wanting to use the user-agent variable that is often logged or used for analytics. Since we are trying to have the server send back the content of the /etc/passwd file I felt like -A was the best option here. By constructing the command seen in the screenshot above we can see that the server did indeed send back the content of the /etc/passwd file. We have correctly launched 1 of the 4 shellshock attacks.

**Task 3.B: Get the server to tell you its process’ user ID. You can use the /bin/id command to print out the ID information.**

****

In the second part of task 3 we are challenged to get the server to tell you it’s process’ user ID. We ended up using the /bin/id command to print out the ID information. In this task I had chosen to use the curl -e command due to wanting to modify and use the referer header variable which is another common header that web servers process and trust allowing us another vector for this attack (we are challenged with using all 3 commands, so this is also why I used -e here). By constructing the command seen in the screenshot above we can see that the server did indeed send back and tell us tell its process’ user ID. We have correctly launched 2 of the 4 shellshock attacks.

**Task 3.C: Get the server to create a file inside the /tmp folder. You need to get into the container to see whether the file is created or not, or use another Shellshock attack to list the /tmp folder.**

****

**Showing that there is no files in the /tmp folder**

**A computer screen with white text

Description automatically generated**

**Creation of the file**

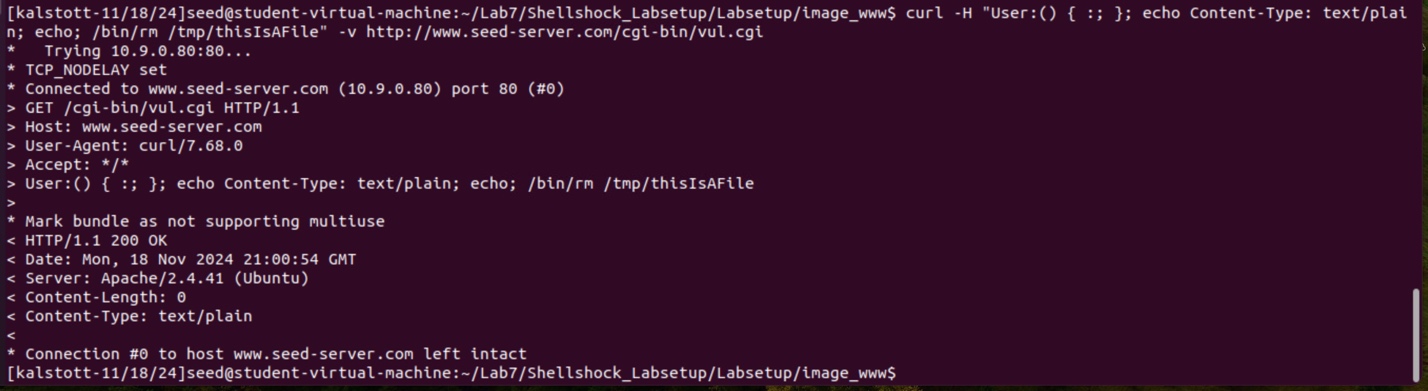
**A screenshot of a computer

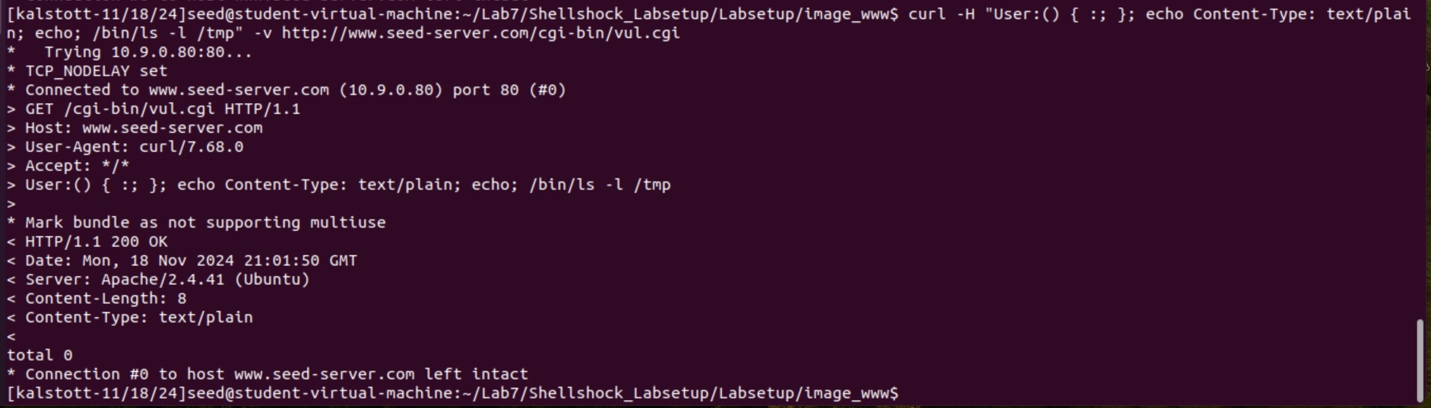
Description automatically generated**

**Showing that there is now the file we created in the /tmp folder.**

In the third part of task 3 we are challenged to get the server to create a file inside the /tmp folder. We had to be sure to check this before we created the file which you can see that we have done within the first screenshot here. In this task I had chosen to use the curl -H command due to wanting to modify and use my own custom header variable, this is used because it provides flexibility if other common headers are filtered or sanitized by the server.Thecustom header used here is “User:”. By constructing the command seen in the screenshot above we can see that the server did indeed create a file inside the /tmp folder named “thisIsAFile”. We have correctly launched 3 of the 4 shellshock attacks.

**Task 3.D: Get the server to delete the file that you just created inside the /tmp folder.**

****

****

In the final part of task 3 we are challenged now to get the server to delete the file that we just created inside the /tmp folder (thisIsAFile). In this task, once again, I had chosen to use the curl -H command due to wanting to modify and use my own custom header variable, this is used because it provides flexibility if other common headers are filtered or sanitized by the server. Thecustom header used here is “User:”. By constructing the command seen in the screenshot above we can see that the server did indeed delete the file inside the /tmp folder named “thisIsAFile”. We have correctly launched all 4 of the 4 shellshock attacks.

**• Question 1: Will you be able to steal the content of the shadow file /etc/shadow from the server? Why or why not? The information obtained in Task 3.B should give you a clue.**

No, we will not be able to steal the content of the shadow file /etc/shadow from the server. The reason as to why we are not able to is because typically the shadow file located in /etc/shadow requires a root or elevated permission to read. As we saw in our task 3B CGI scripts are usually run with limited permissions, in our case it looks like with permissions of www-data. As confirmed in task 3B the server process does not have root privileges allowing us to fail when we try to read the shadow file.

**• Question 2: HTTP GET requests typically attach data in the URL, after the ? mark. This could be another approach that we can use to launch the attack. In the following example, we attach some data in the URL, and we found that the data are used to set the following environment variable:**

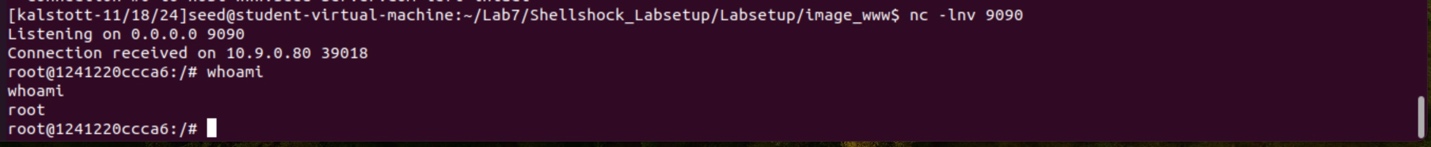
**$ curl "http://www.seed-server.com/cgi-bin/getenv.cgi?AAAAA"  
...  
QUERY\_STRING=AAAAA  
...**

**Can we use this method to launch the Shellshock attack? Please conduct your experiment and derive your conclusions based on your experiment results.**

When conducting this experiment, I was unsuccessful with launching the Shellshock attack with this method. I do not think we are able to lunch the Shellshock attack from this method. What I had found was that the data after the ? in the URL actually gets assigned to the QUERY\_STRING environment variable. It appears to me that using this approach will not work because the Bash shell does not interpret the QUERY\_STRING the same way as other environment variables. It looks like QUERY\_STRING is treated like a string and does not pass to Bash.

**Task 4: Getting a Reverse Shell via Shellshock Attack**

In this task our goal is to exploit the Shellshock vulnerability to create a reverse shell.

****

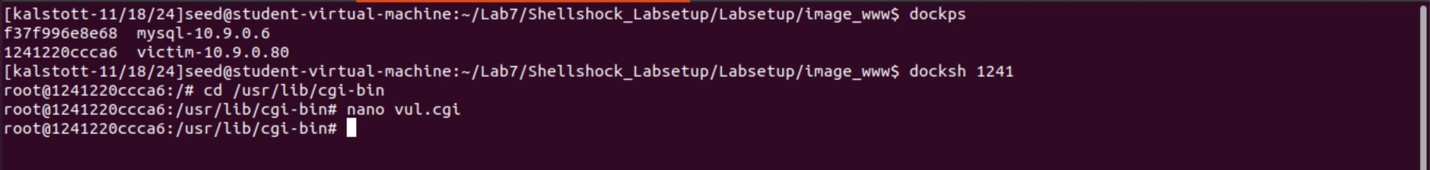
**A screenshot of a computer

Description automatically generated**

Our first step in this task was to start out Netcat or nc command. This will allow us to listen to the specified port that we have listed (9090, taken from the textbook). Now we directly run the bash program on the server’s machine (bash program specified in textbook pg. 331). This will emulate what an attacker would run after compromising the server. This bash program used from the textbook simply will start a bash shell in the server’s machine with both input and output coming from the TCP connection we had specified. As you can see, we were successful with launching the bash shell showing that our attack was successful.

**Task 5: Using the Patched Bash**

In this task our goal is to see what would happen when we tested a patched version of Bash. This would allow us to see how it affects the ability to exploit the Shellshock vulnerability.

****

We accessed the victim container where the vulnerable CGI program (vul.cgi) is located.

**A screenshot of a computer

Description automatically generated**

**Changed file.** We replaced the first line of the vul.cgi script with the patched version of Bash (#!/bin/bash).

**A screenshot of a computer

Description automatically generated**

We reattempted the Shellshock attack like we did in task 3 to see if the vulnerability could still be exploited with the patched Bash. As we can see with this patched version of bash our attack was unsuccessful and we did not print out the process’ user ID but our text of “Hello World”.

**Note:** We did not perform all 4 tasks but only one to show that all 4 would still fail and only print what is in our script which in this case is the text of “Hello World”.