CS445 Lab Report: Shellshock Attack

Austin Decker

November 12 2024

# Environment Setup:

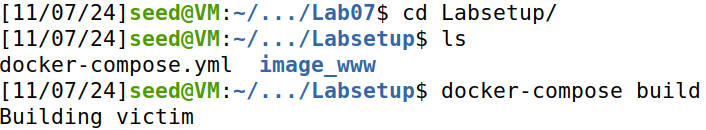
## DNS Setting:

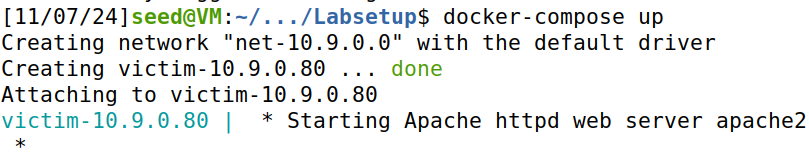
Adding mapping [10.9.0.80 [www.seedlab-shellshock.com](http://www.seedlab-shellshock.com/)] to //etc/hosts.



## Container Setup:

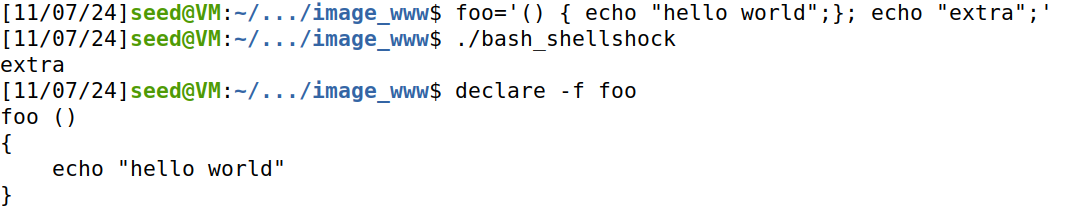
**Building container (output too long):**

  
**Running Container (output too long):**



# Task 1: Experimenting with Bash Functions

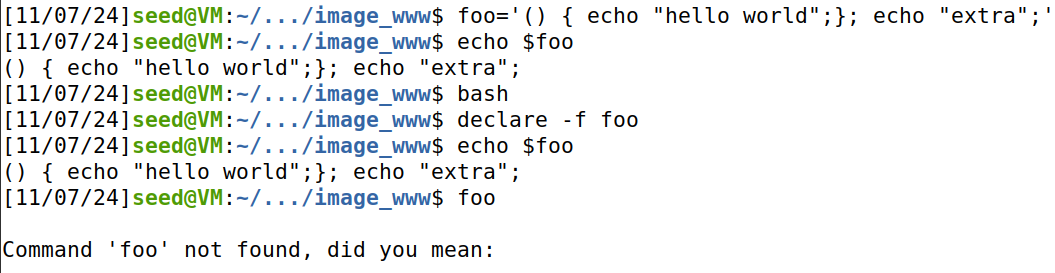
**Exporting a shell variable and running bash\_shellshock:**



**Observations:**

I defined a shell variable called foo and exported it before running the vulnerable bash version. As the output shows, the vulnerability worked as it ran the extra command and exported the shell variable as a function.

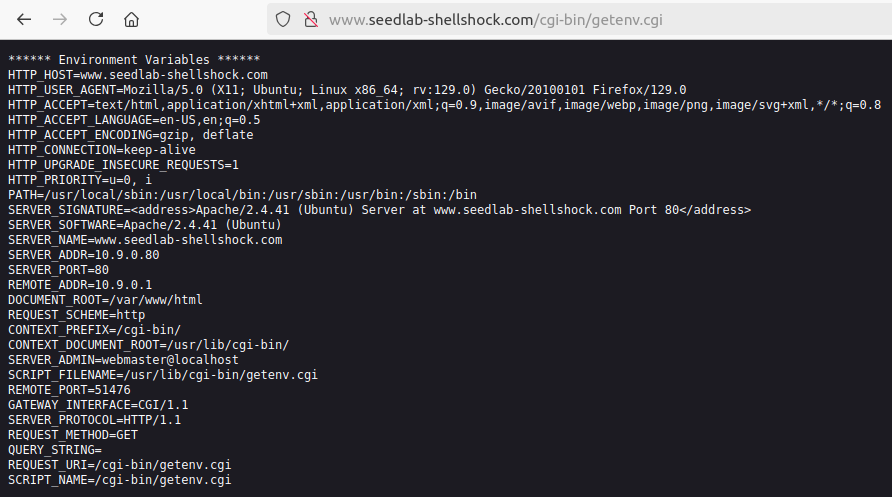
**Exporting a shell variable and running the updated bash:**

**Observations:**

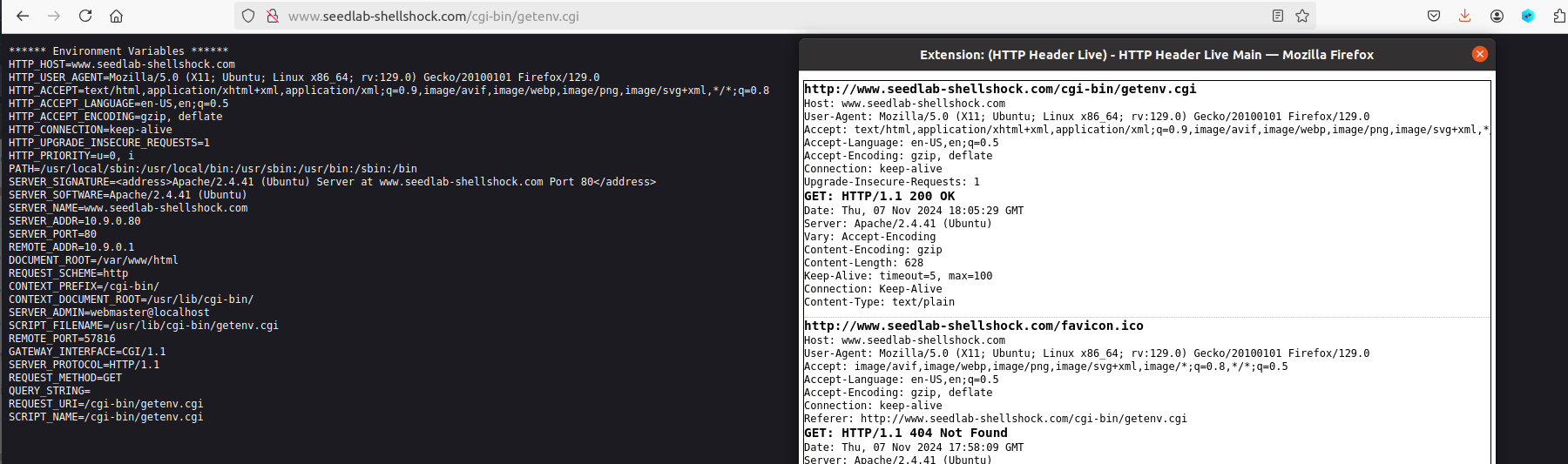
I defined a shell variable called foo just as I did in the first experiment. This time I ran the patched version of bash with the exploit patched. As you can see from the screenshot, it did not run the extra command and it was also not turned into a bash function.

# Task 2: Passing Data to Bash Via Environ Variable

**Running getenv.cgi in browser:**

****

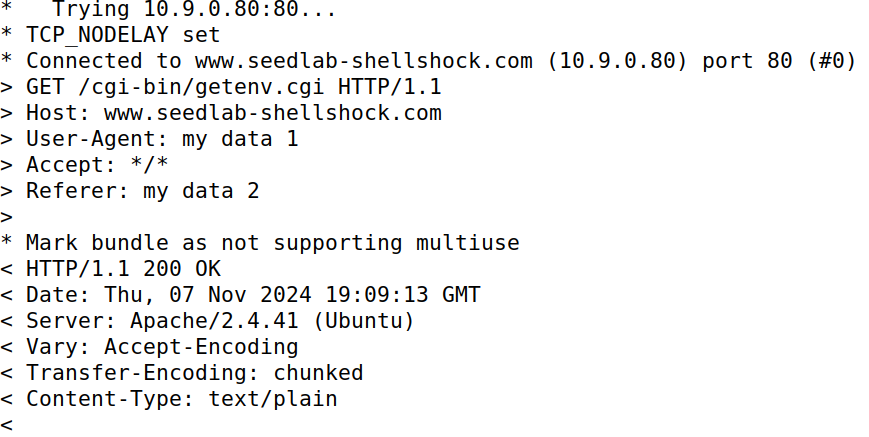
## Task 2-A: Using Browser

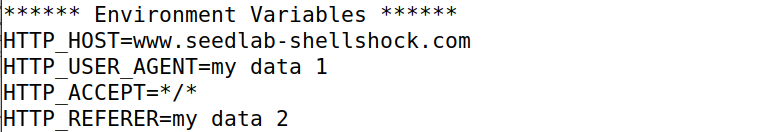


**Observations:**

This task was a little unclear. I ran the getenv.cgi program on the browser and compared the output from the server and the HTTP header from the extension. Based on what I can tell, the HTTP\_HOST, HTTP\_USER\_AGENT, HTTP\_ACCEPT, HTTP\_ACCEPT\_LANGUAGE, HTTP\_ACCEPT\_ENCODING are all from the browser. The screenshot above shows an overlap between the browser HTTP REQUEST headers and what some of the server env variables are stored and named.

## Task 2-B: Using curl



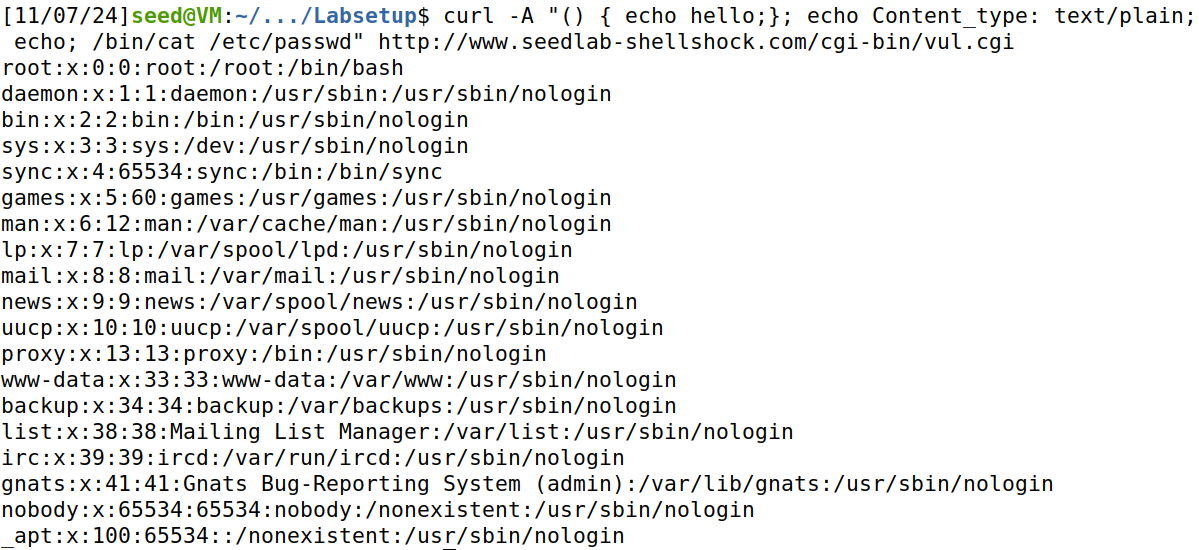


**Observations:**

I found that the -A and -e flags for curl represent the HTTP\_USER\_AGENT and HTTP\_REFERRER respectively. The -v flag prints out the header of the HTTP Request. We can probably use one of these flags to insert our environment variable into the CGI program.

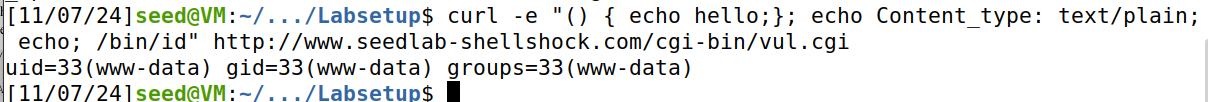
# Task 3: Launching the Shellshock Attack

## Task 3-A: Get //etc/passwd



Above is the results of the successful shellshock attack. As the screenshot shows, I got the server to print out the contents of the //etc/passwd file.

## Task 3-B: Get server process user id

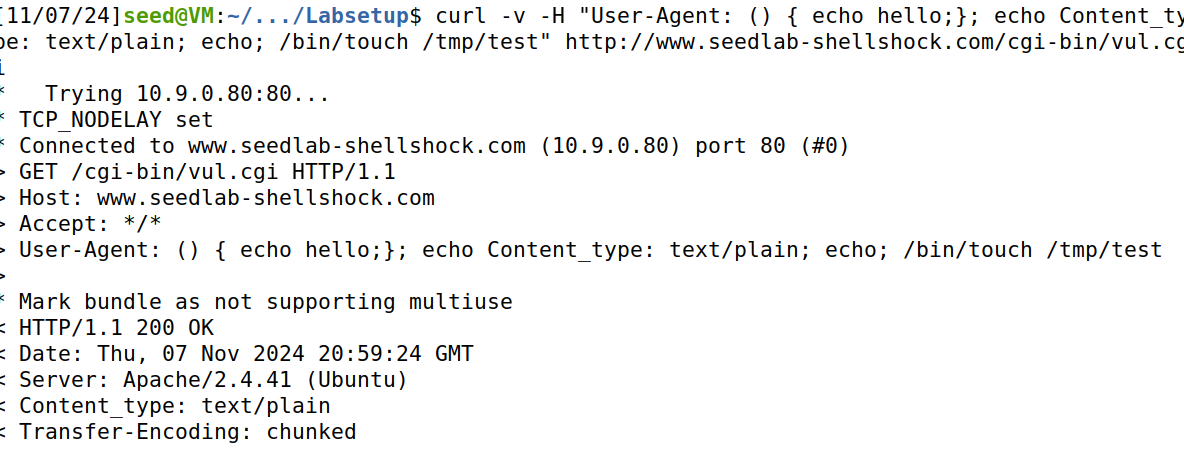


The output above shows the successful shellshock attack. The uid of the running process on the server is 33 (www-data).

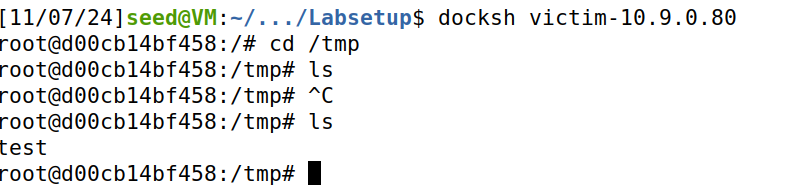
Note that I also used a different flag -e instead of -A. I could have used either, but I must use all the flags from Task 2 at least once.

## Task 3-C: Creating file in //tmp

**Creating the file:**

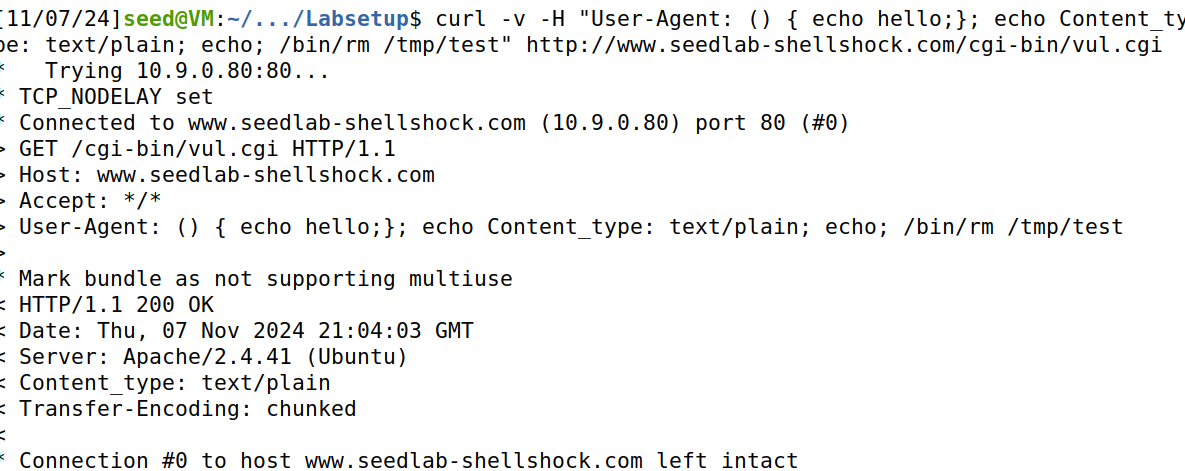
****

**Verifying file has been created in tmp folder:**

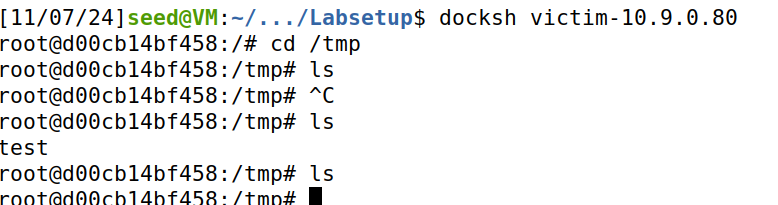
****

## Task 3-D: Removing created file from tmp

**Removing the file:**

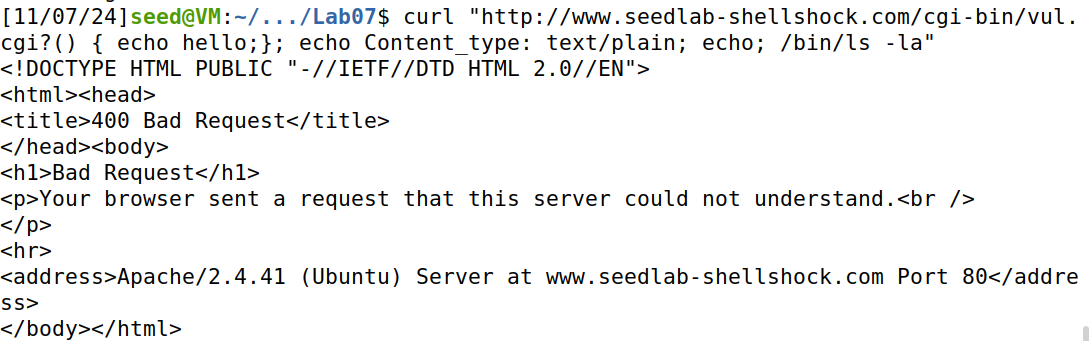
****

**Verifying file deletion:**

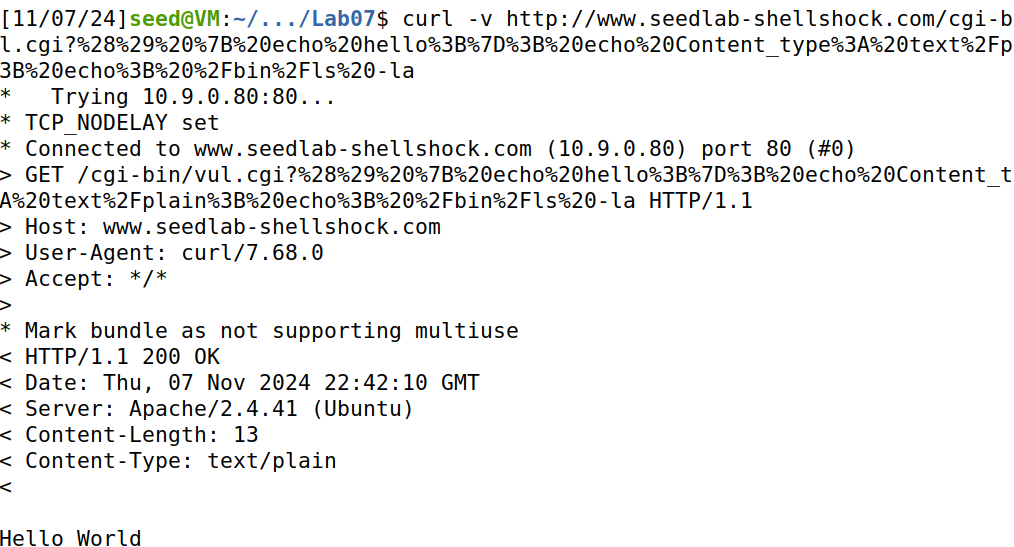
****

## Questions:

1. No, you would not be able to steal the content of the //etc/shadow from the server. This is because the shadow file typically can only be accessed via root. The server is running as a user with much less authority and access control than root.
2. I believe it would not be possible to do a shellshock attack via this method. I attempted the attack, but the shellshock exploit seemed to have failed.



I also tried encoding the query, and while this stopped the bad request, it still did not work:



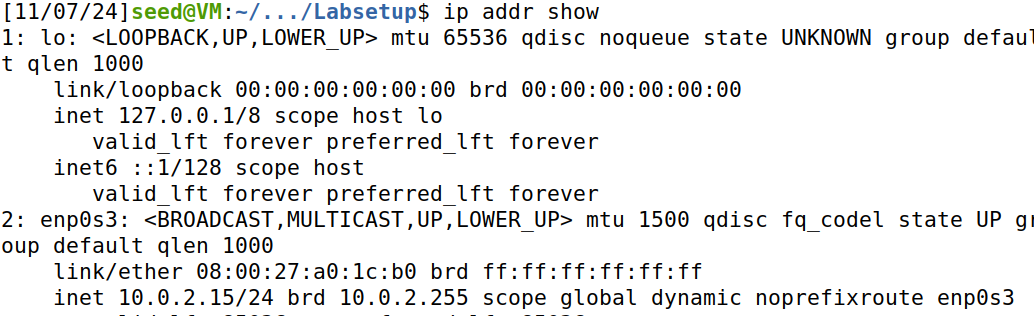
# Task 4: Getting a Reverse Shell Via Shellshock Attack

**Setting up a tcp server listening for connections:**

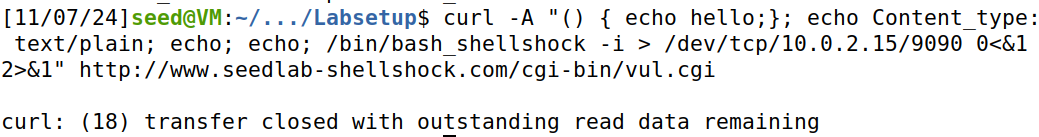
****

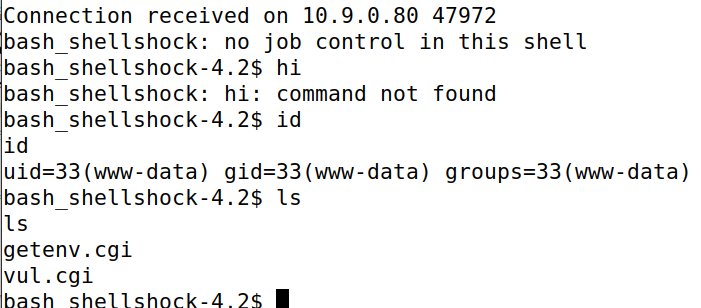
**Setting up reverse shell:**

First I had to find the correct IP to connect to:



The correct IP I needed the server to connect to was 10.0.2.15. Now I can properly set up the shellshock reverse shell.





The output above shows the successful reverse shell via the shellshock exploit.