CSE: 5382-001: SECURE PROGRAMMING

ASSIGNMENT 2

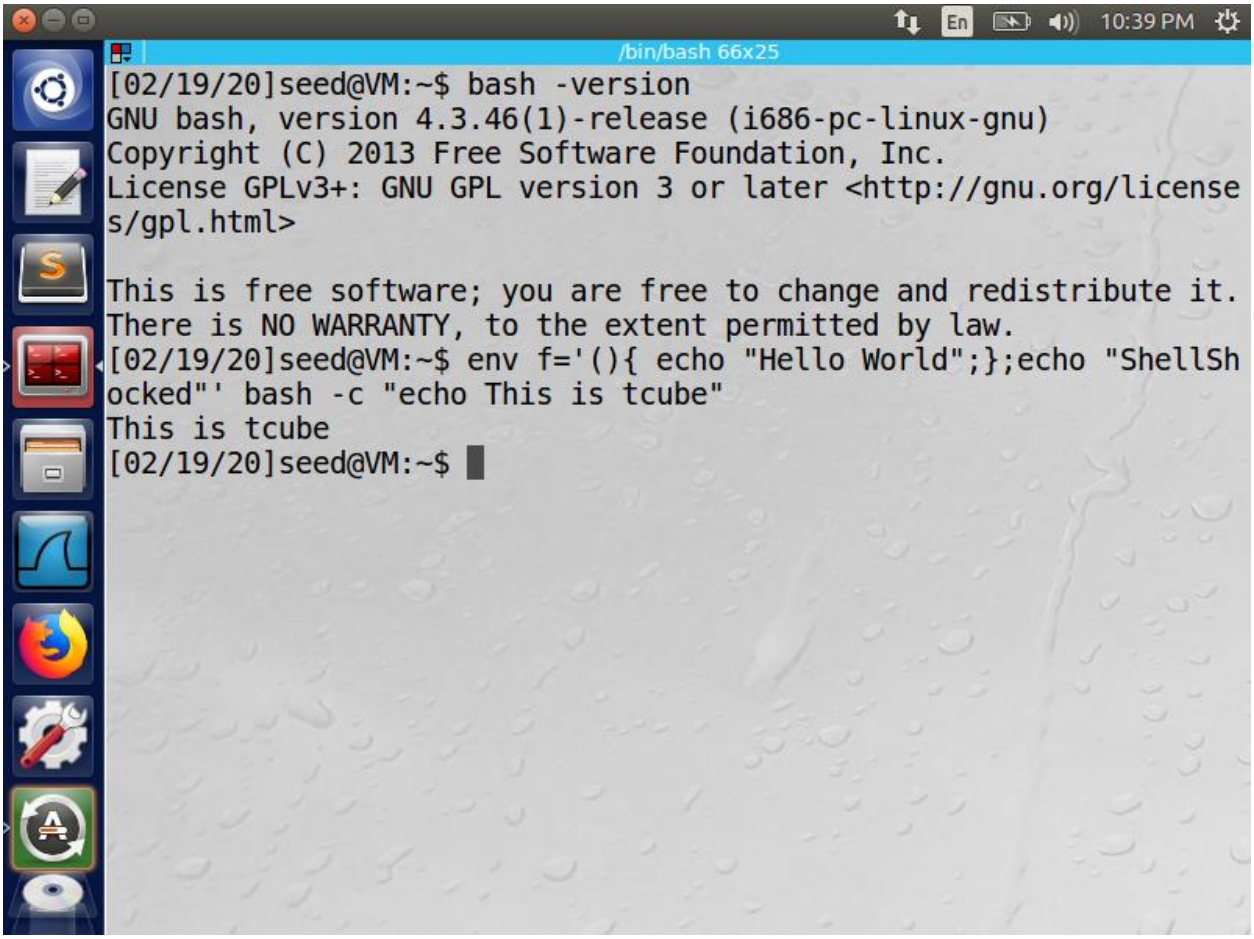
Tharoon T Thiagarajan

1001704601

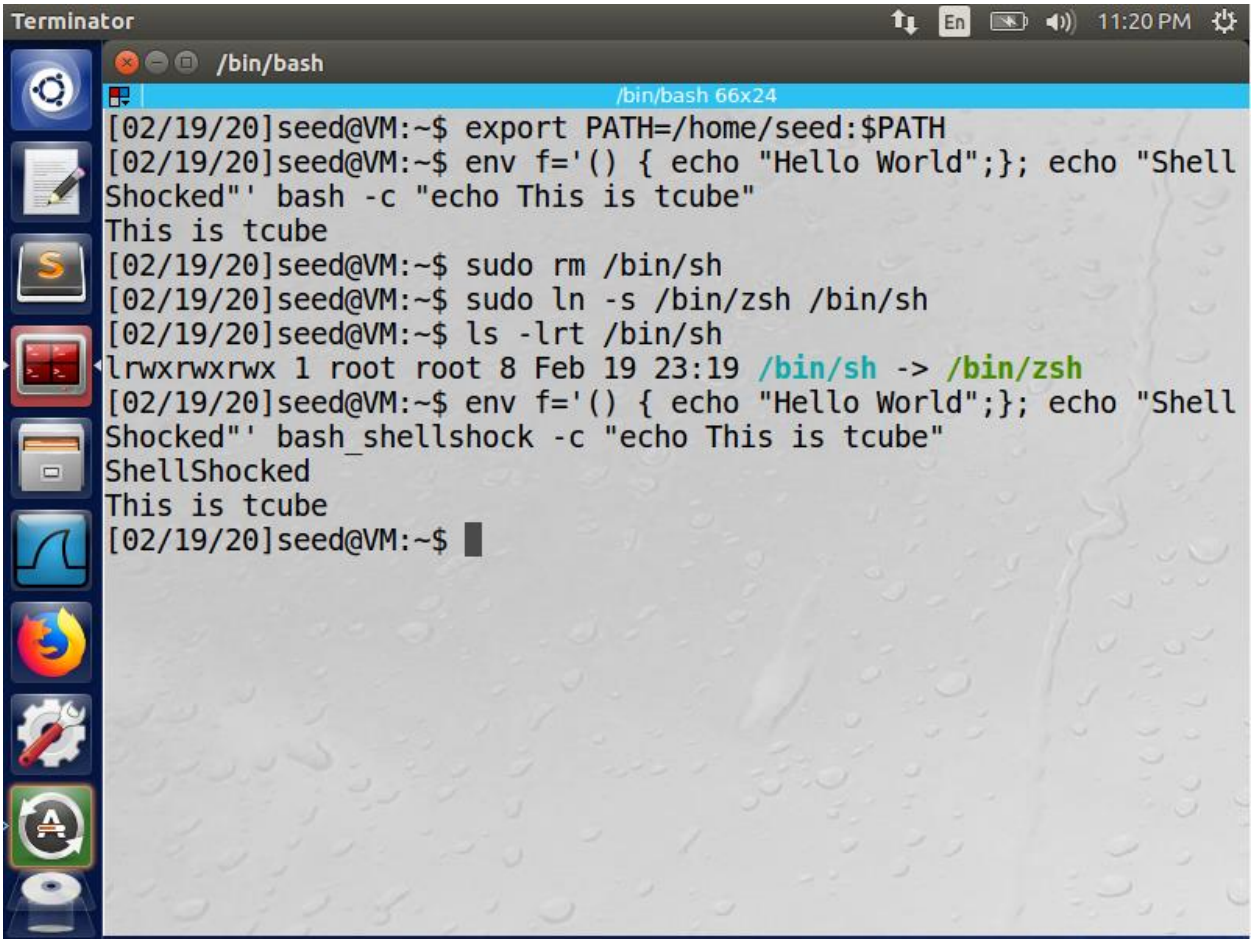
2.1 Task 1: Experimenting with Bash Function

Output:

First, I check the version of the bash to ensure that it is not shellshock vulnerable. Then from the seed user I invoke the function definition of the shell variable and I run the function definition using the bash that is not shellshock vulnerable and to check if the bash is not shellshock vulnerable, I have used the echo statement after compiling the function definition. I am able to see that the statements inside the function is not invoked and the statement 'ShellShocked' is not getting printed. Only the echo statement after compilation is getting printed. This is because the Ubuntu 16.04 is already patched and not vulnerable to shellshock attack.



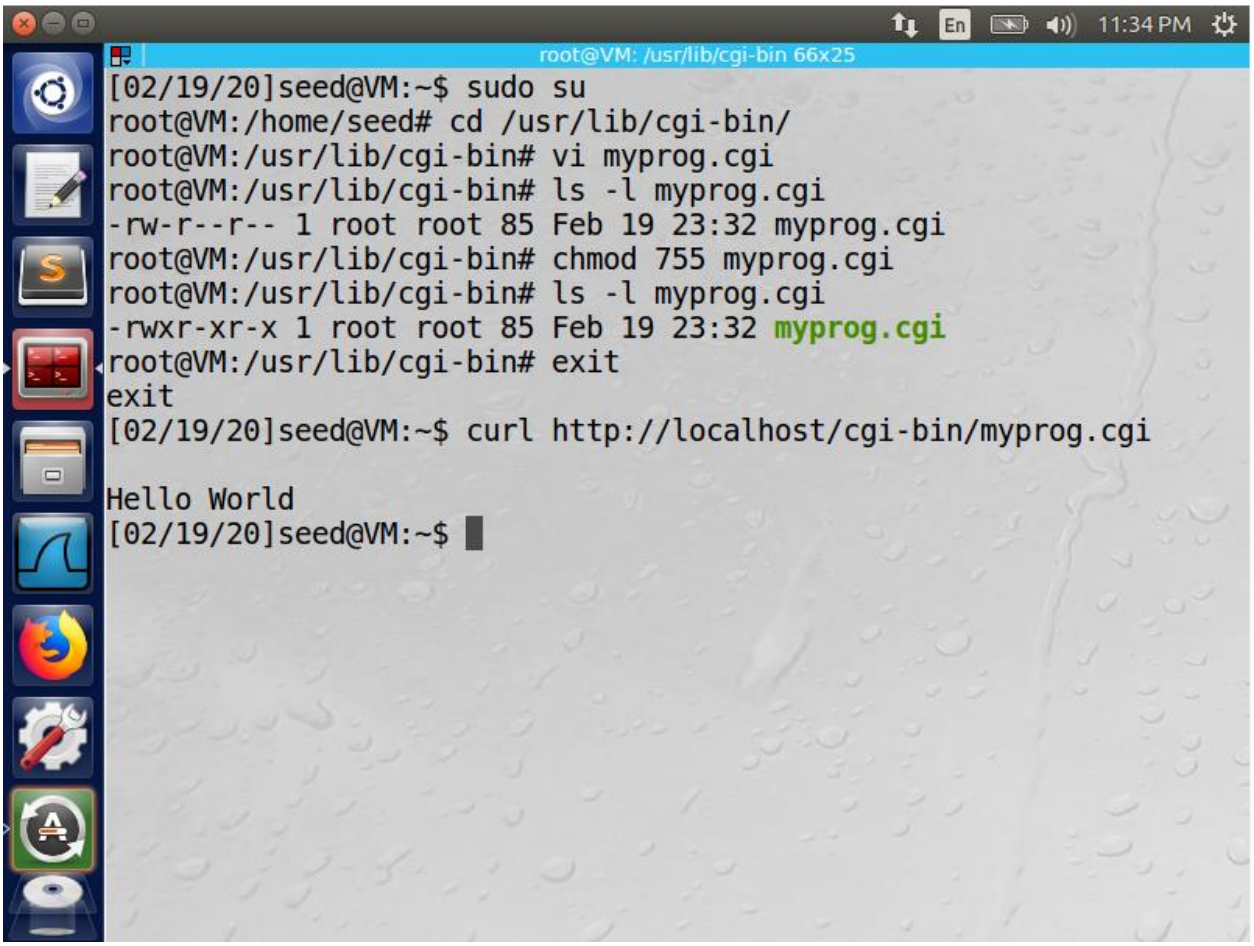
Next, I export the PATH environment to the environment variable. Then I remove the /bin/sh which is not vulnerable to shellshock attack, and I created a symbolic link pointing to the vulnerable shell /bin/zh. Now, when I ran the function definition and compiled using the bash_shellshock, I am able to see the statement 'ShellShocked' getting printed. This is because of the shellshock vulnerability present in bash_shellshock, which allows execution of the echo statement after the function definiton.



2.2 Task 2: Setting up CGI programs

Output:

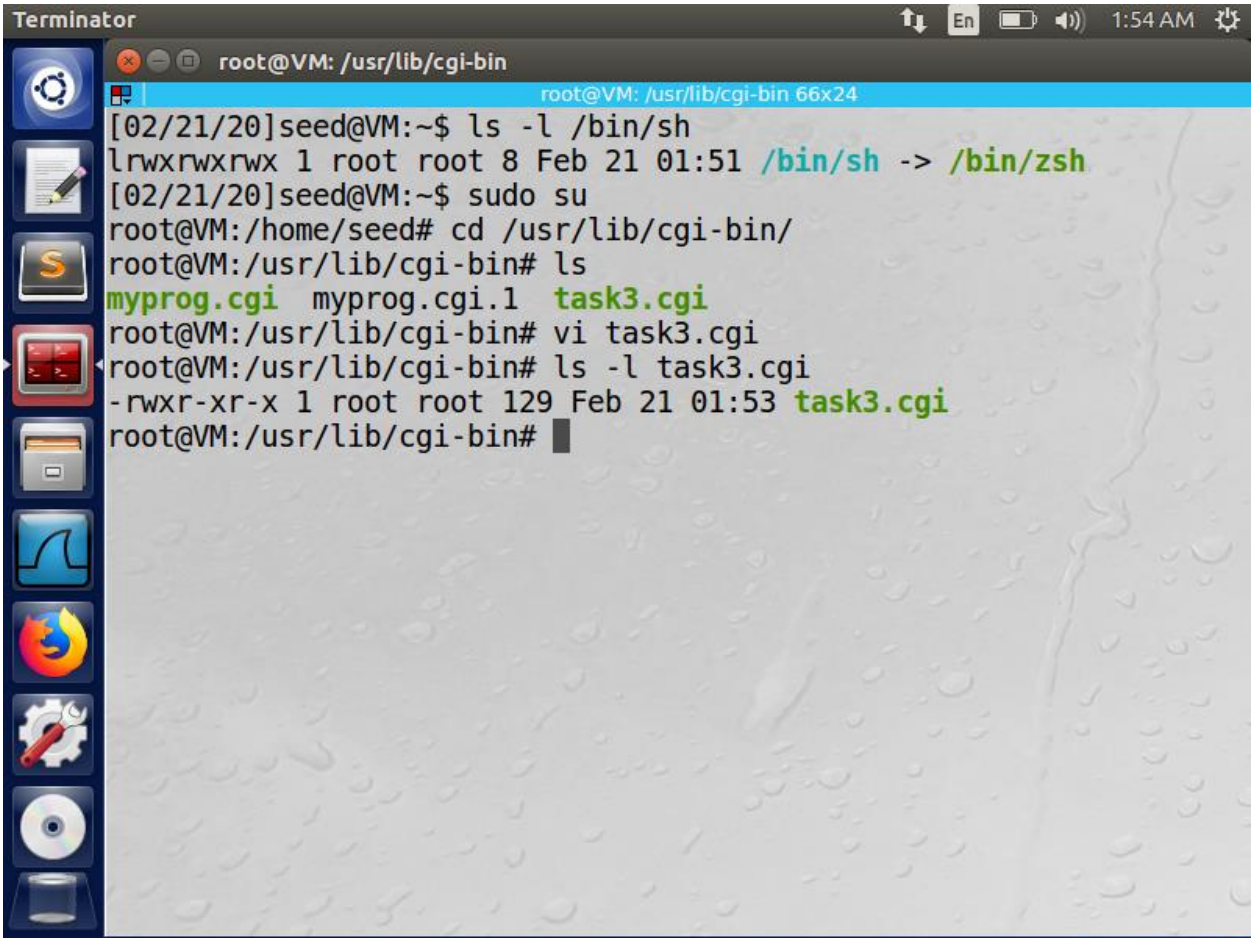
Using the sudo su command I gained access into the root account. Then I navigated to the /usr/lib/cgi-bin/ directory using the cd command. Then I created the myprog.cgi from the given shell script and saved it in the cgi-bin directory. After creating the file, I changed its permission to 755 and made it as executable. Then I ran the curl http://localhost/cgi-bin/myprog.cgi command from the seed user, I was able to see the statement getting printed that was in myprog.cgi. This is because the default CGI directory for the web server is the cgi-bin. So, when invoked using the curl command, our file myprog.cgi gets executed which is present inside the cgi-bin directory.



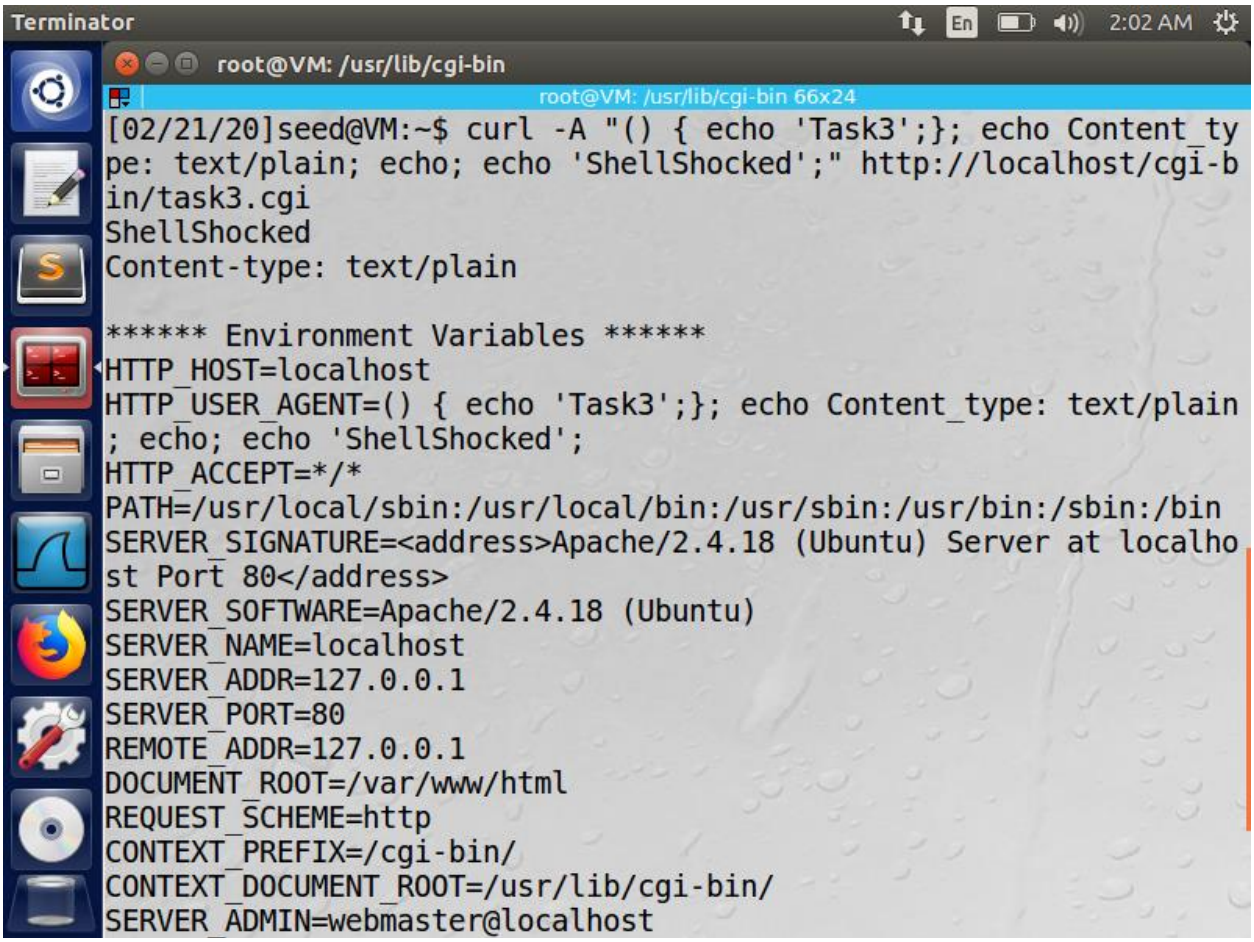
2.3 Task 3: Passing Data to Bash via Environment Variable

Output:

I have checked whether the bash sh is pointing to the vulnerable bash zsh using the ls command. Then using the sudo command I went to the root login, and navigated to the /usr/lib/cgi-bin directory using the cd command. The I created a new cgi file called task3 and I wrote the given program in that file and saved it in the root user. Then I changed the privileges to 755 using chmod and made it as executable inside the cgi-bin directory. I also checked the file permission to recheck the privileges. After all the process in root user I exit the root user and came to the seed user.



Now, I am in seed user and I ran the curl command along with appending the echo statement 'ShellShocked' in the function definition of the shell variable. Also gave the localhost link of the cgi-bin directory and the task3.cgi file which we created. Now when I ran the curl command, I am able to see the statement 'ShellShocked' getting printed along with the environment variables. Since our file task3.cgi had the command to list all the environment variables of the current process using the strings /proc/$$/environ command.

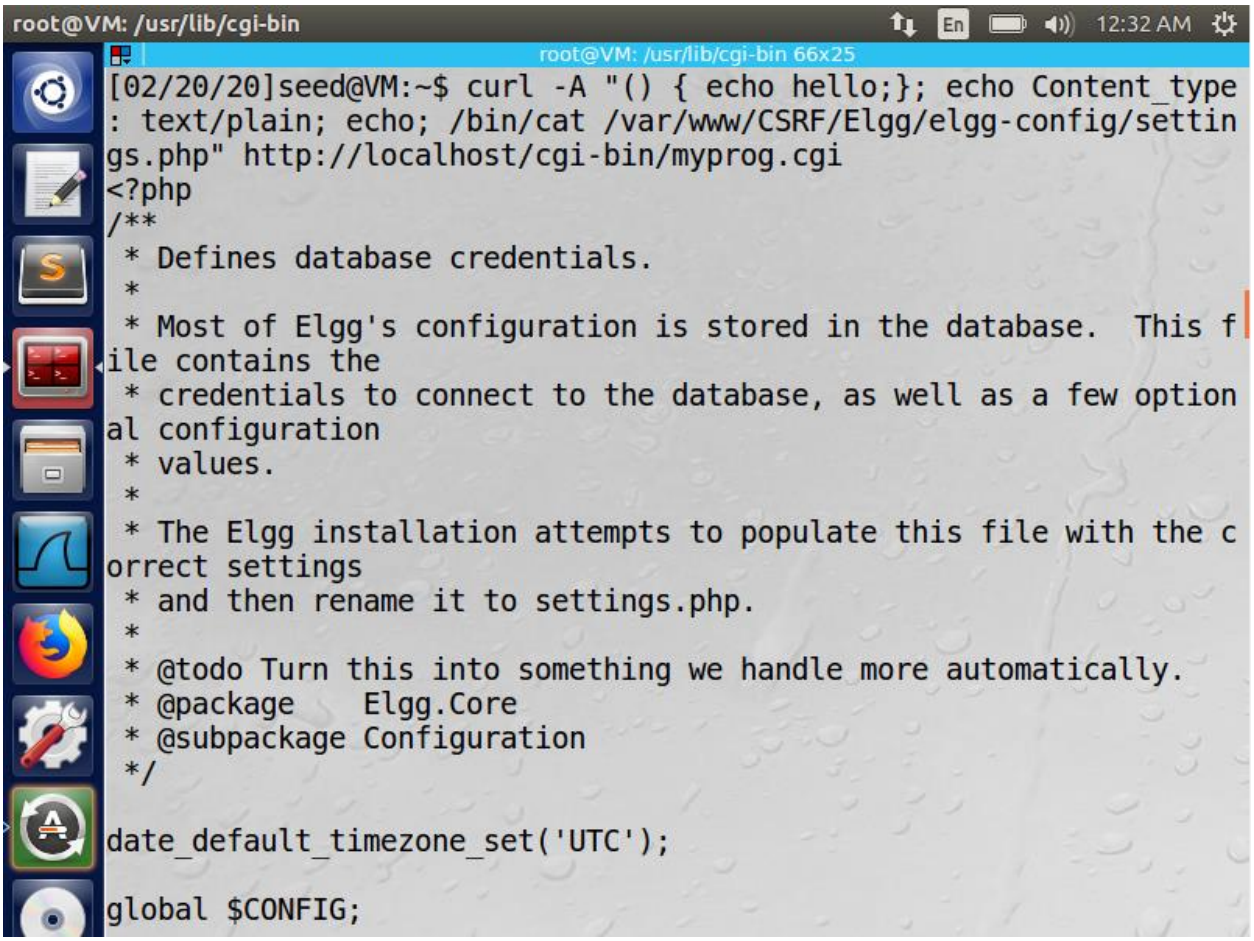


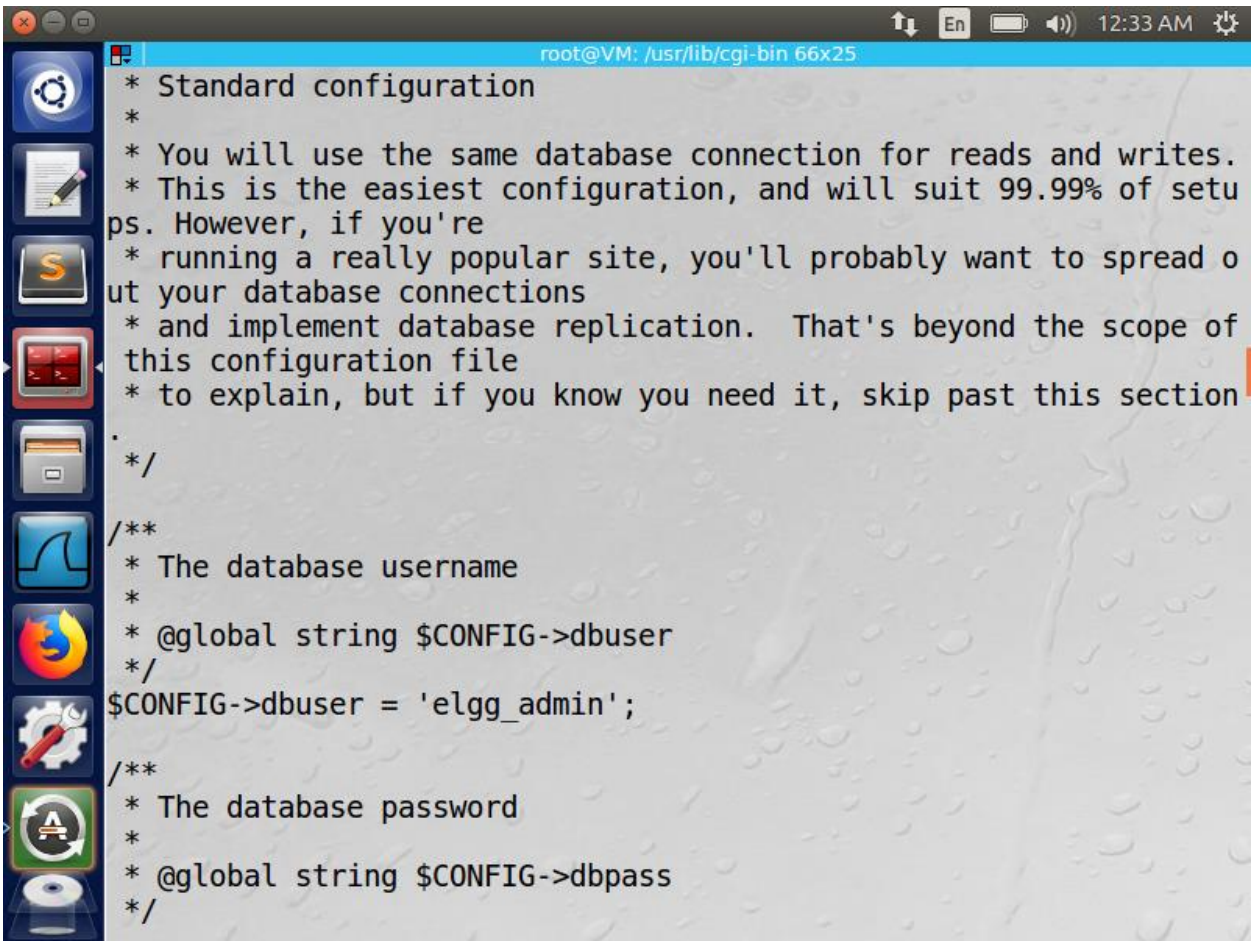
The reason that the statement 'ShellShocked' from a remote user can get into the environment variables when running the curl command is due the vulnerable shell zsh, in which the parse_and_execute function does not work properly. The vulnerable shell zsh is not patched and this the reason that the data can be passed via the environment variables.

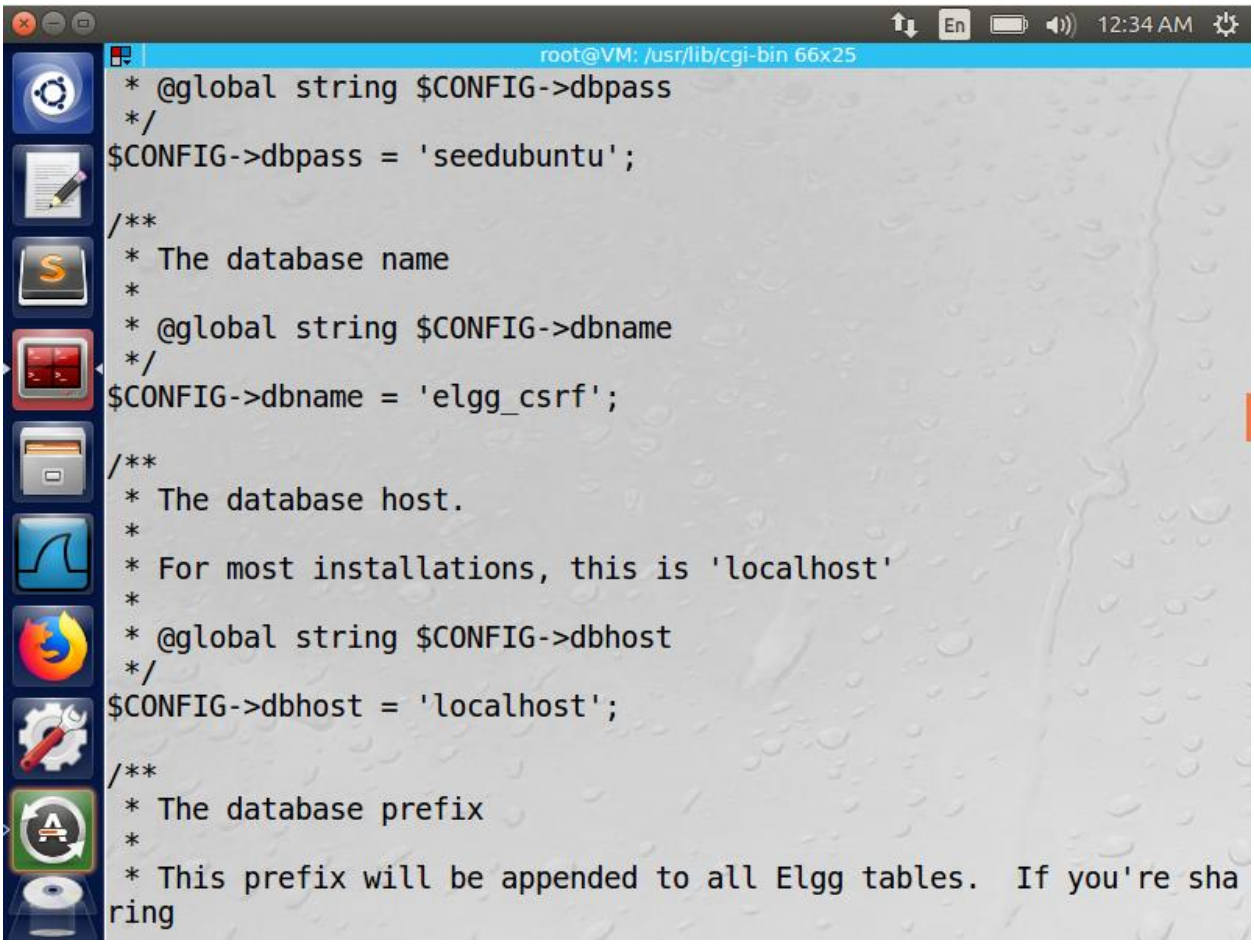
2.4 Task 4: Launching the Shellshock Attack

Output:

I try to view the configuration details of the database in the server using the curl command. In my curl command under the function definition of the shell variable I gave the file location where the username and password of the database resides on the server. I used /bin/cat command to view the contents of the file directly while running the curl command. I was able to see the username and password of the database in the server using the curl command. The statements after the end of the function definition gets executed because, the parse_and_execute() function can parse other shell commands also, which eventually leads to shellshock vulnerability.

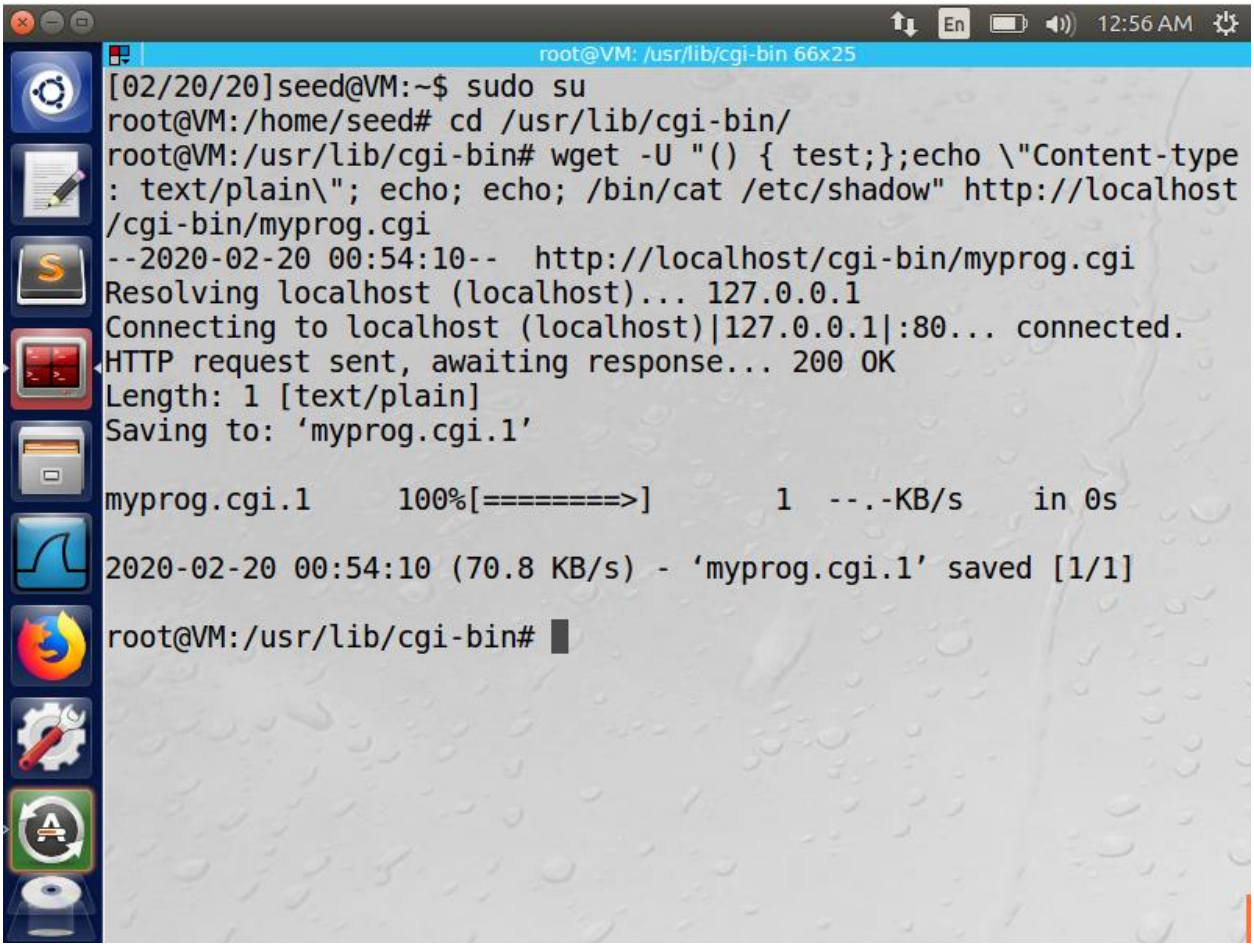



12:33 AM
root@VM: /usr/lib/cgi-bin 66x25
* Standard configuration
*
* You will use the same database connection for reads and writes.
* This is the easiest configuration, and will suit 99.99% of setu
ps. However, if you're
* running a really popular site, you'll probably want to spread o
ut your database connections
* and implement database replication. That's beyond the scope of
this configuration file
* to explain, but if you know you need it, skip past this section
.
*/
/**
* The database username
*
* @global string $CONFIG->dbuser
*/
$CONFIG->dbuser = 'elgg_admin';
/**
* The database password
*
* @global string $CONFIG->dbpass
*/



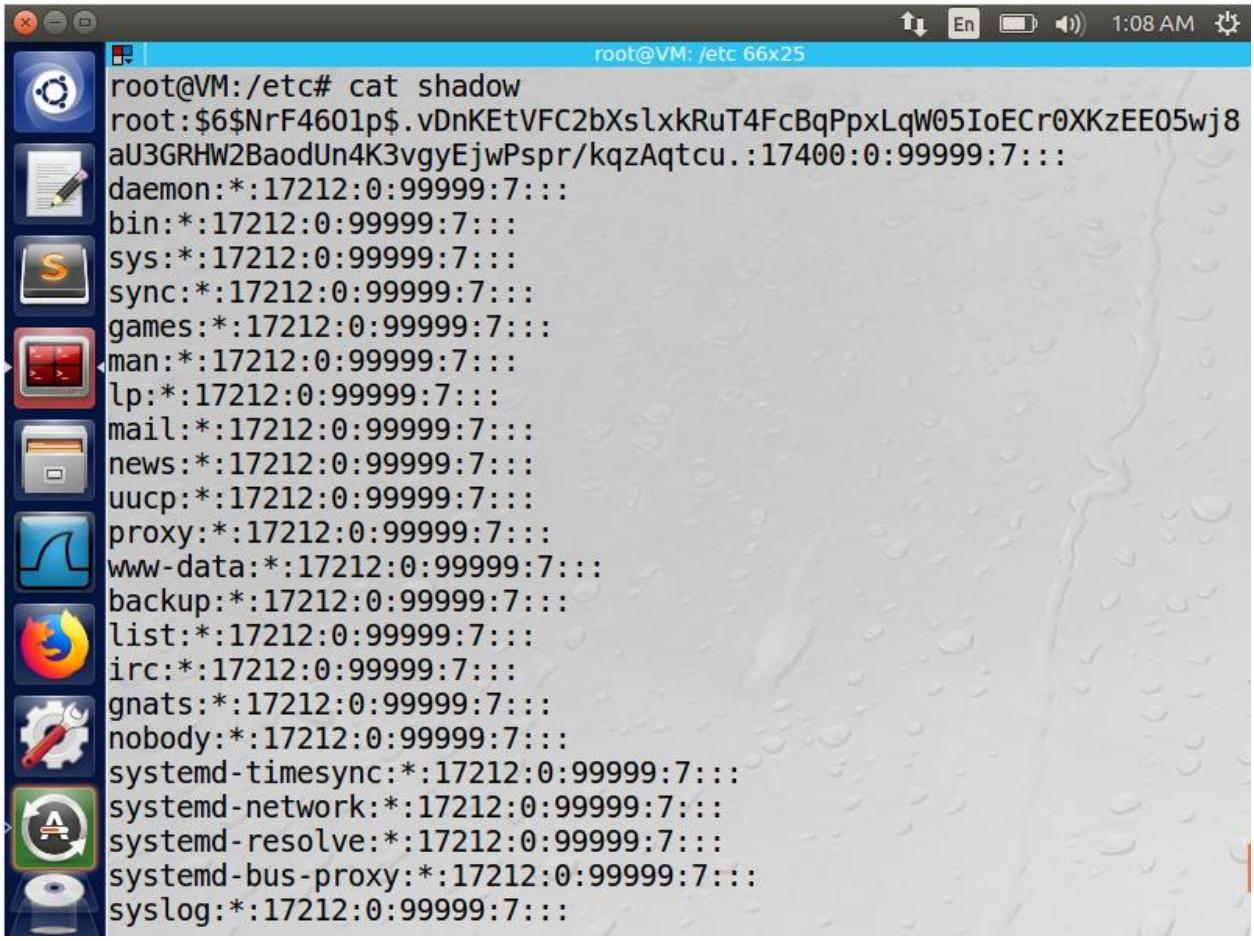
```
 * @global string $CONFIG->dbpass
 */
$CONFIG->dbpass = 'seedubuntu';

/**
 * The database name
 *
 * @global string $CONFIG->dbname
 */
$CONFIG->dbname = 'elgg_csrf';

/**
 * The database host.
 *
 * For most installations, this is 'localhost'
 *
 * @global string $CONFIG->dbhost
 */
$CONFIG->dbhost = 'localhost';

/**
 * The database prefix
 *
 * This prefix will be appended to all Elgg tables.  If you're sha
ring
```

Using the sudo command I get into the root user. I then navigate to the cgi-bin directory using the cd command. Then in order to steal the contents of the /etc/shadow file, we use the wget command to get the contents of the /etc/shadow file and store it a file called myprog.cgi.1. When I try to open the file myprog.cgi.1 which is suppose to have the contents of the file /etc/shadow, is empty. This is because the contents of /etc/shadow is not writable and thus the file myprog.cgi.1 is empty. Hence, we cannot steal the contents of the file /etc/shadow.




En
12:58 AM
root@VM: /usr/lib/cgi-bin 66x25
"myprog.cgi.1" 1L, 1C
1,0-1
All

Now when I try to view the contents of the shadow file using the cat command, I am able to view the contents of the shadow file. Previously I tried to steal the contents of the /etc/shadow file and store it in an another file and I was not able to do it. This is because the file is write protected and it is only accessible for the root user. No other user can view the contents of the /etc/shadow file. Since it is root owned file.

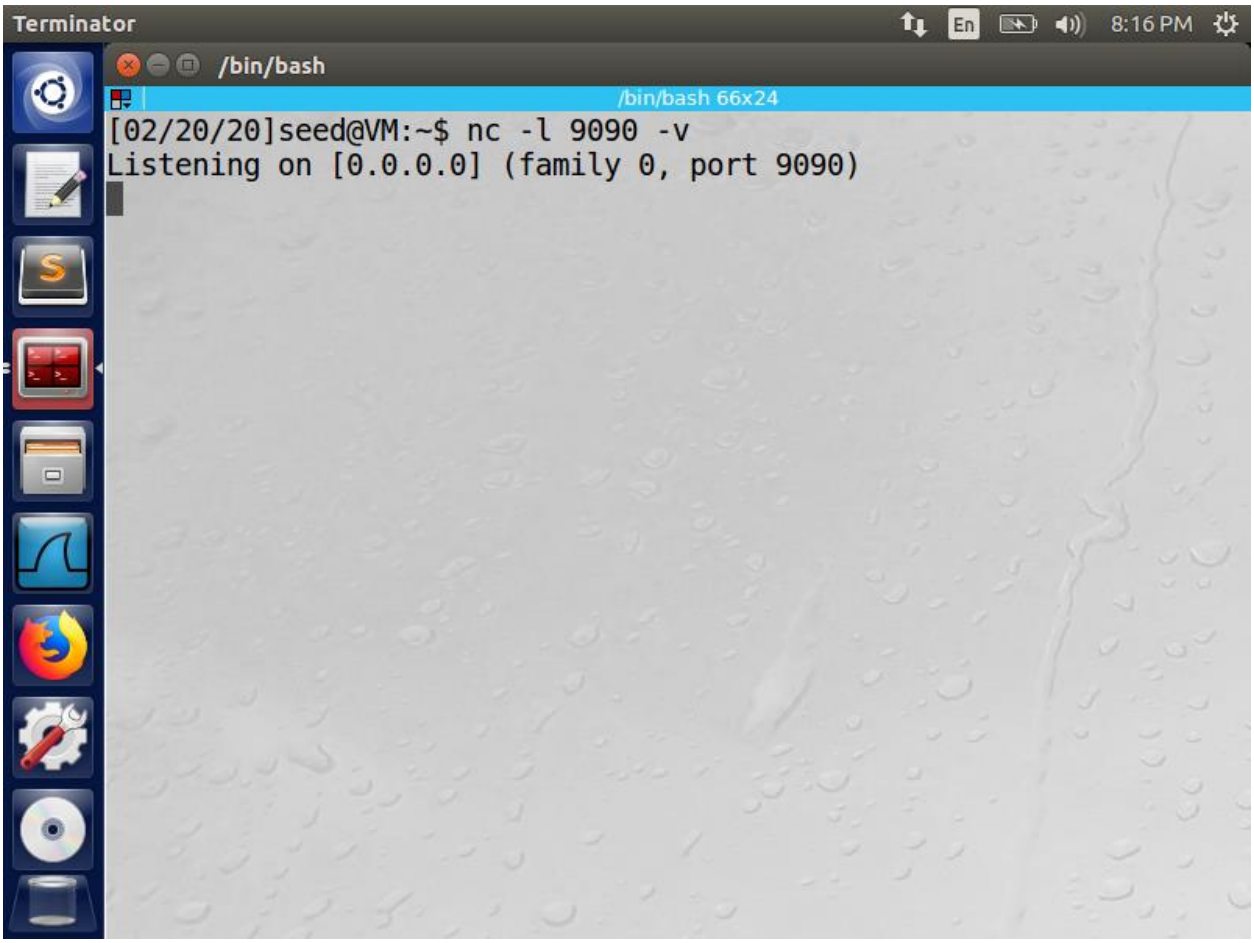


```
root@VM: /etc                                                    1:10 AM
root@VM: /etc 66x25
lightdm:*:17212:0:99999:7:::
whoopsie:*:17212:0:99999:7:::
avahi-autoipd:*:17212:0:99999:7:::
avahi:*:17212:0:99999:7:::
dnsmasq:*:17212:0:99999:7:::
colord:*:17212:0:99999:7:::
speech-dispatcher:!:17212:0:99999:7:::
hplip:*:17212:0:99999:7:::
kernoops:*:17212:0:99999:7:::
pulse:*:17212:0:99999:7:::
rtkit:*:17212:0:99999:7:::
saned:*:17212:0:99999:7:::
usbmux:*:17212:0:99999:7:::
seed:$6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN
/sfYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/:17372:0:99999:7:::
vboxadd:!:17372::::::
telnetd:*:17372:0:99999:7:::
sshd:*:17372:0:99999:7:::
ftp:*:17372:0:99999:7:::
bind:*:17372:0:99999:7:::
mysql:!:17372:0:99999:7:::
tharoon:$6$x/s10Dr/$NdNsvR.G1wXPoo/UwyLJAk7s40YKKCRk6ZznFJUGoYRgcJ
PJxm6gegbvT3qUXDF2Jzw0oo9bMCu2WlSy95b9K0:18306:0:99999:7:::
root@VM:/etc#
Display all 2702 possibilities? (y or n)
```

2.5 Task 5: Getting a Reverse Shell via Shellshock Attack

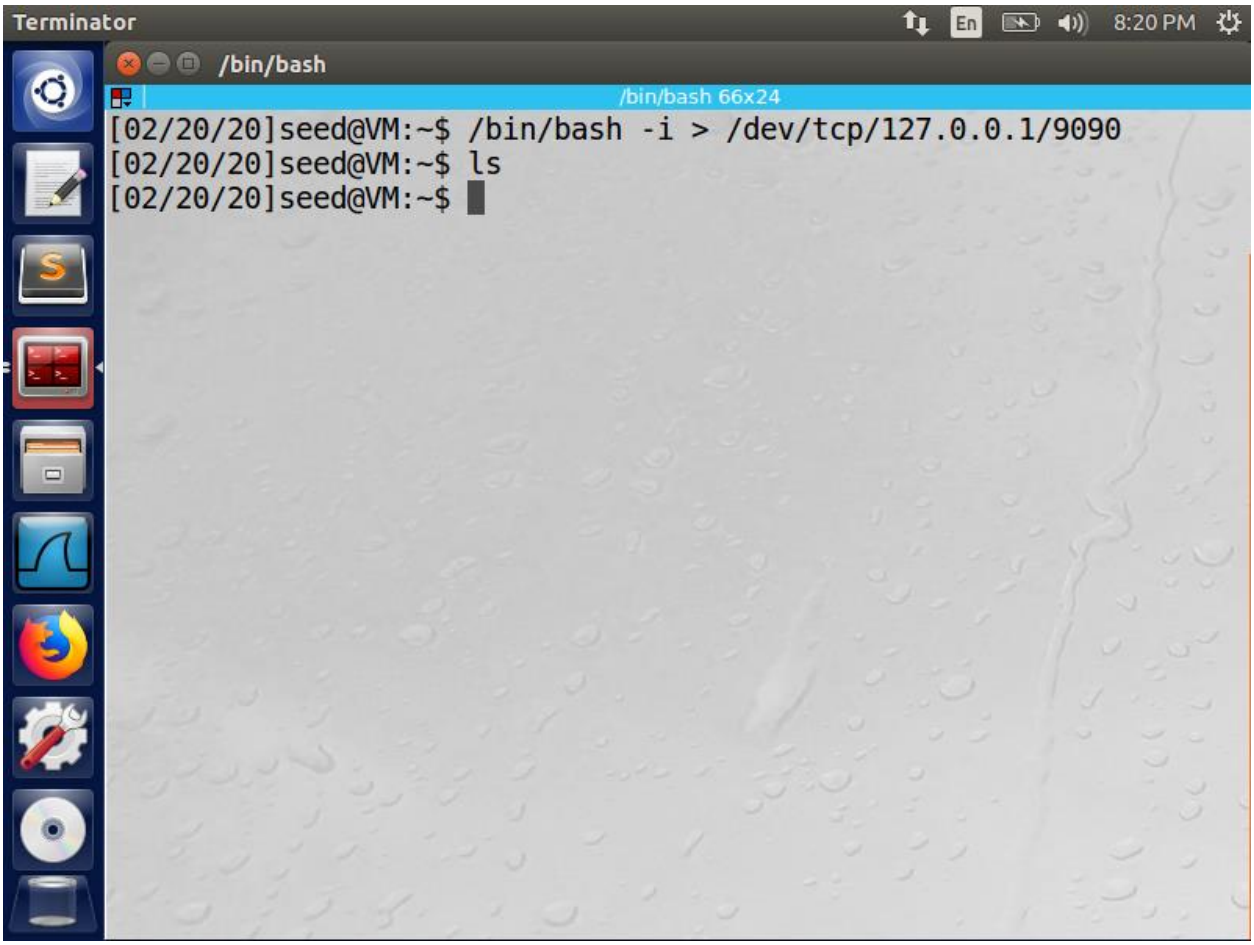
Output:

To create a reverse shell I use the netcat (nc) command to listen to the victim system. From the attacker terminal I have started to listen using the nc -l 9090 -v command.

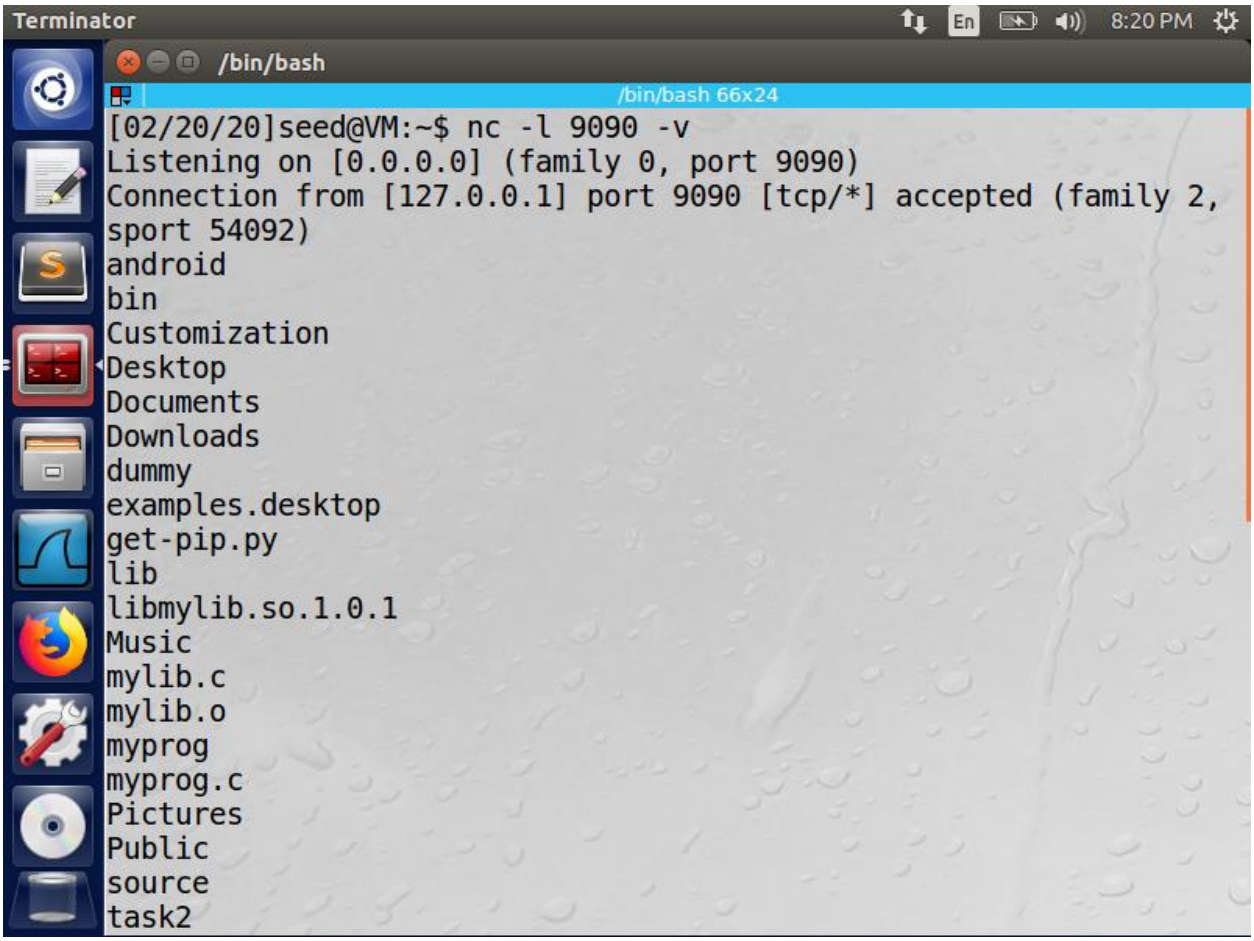


This is the victim terminal where I am redirecting the victims bash to the attacker's machine using the looping address. Now the connection has been established in the attacker's machine.

Now when I give the ls command from the victims machine all the files will be listed in the attacker's machine.

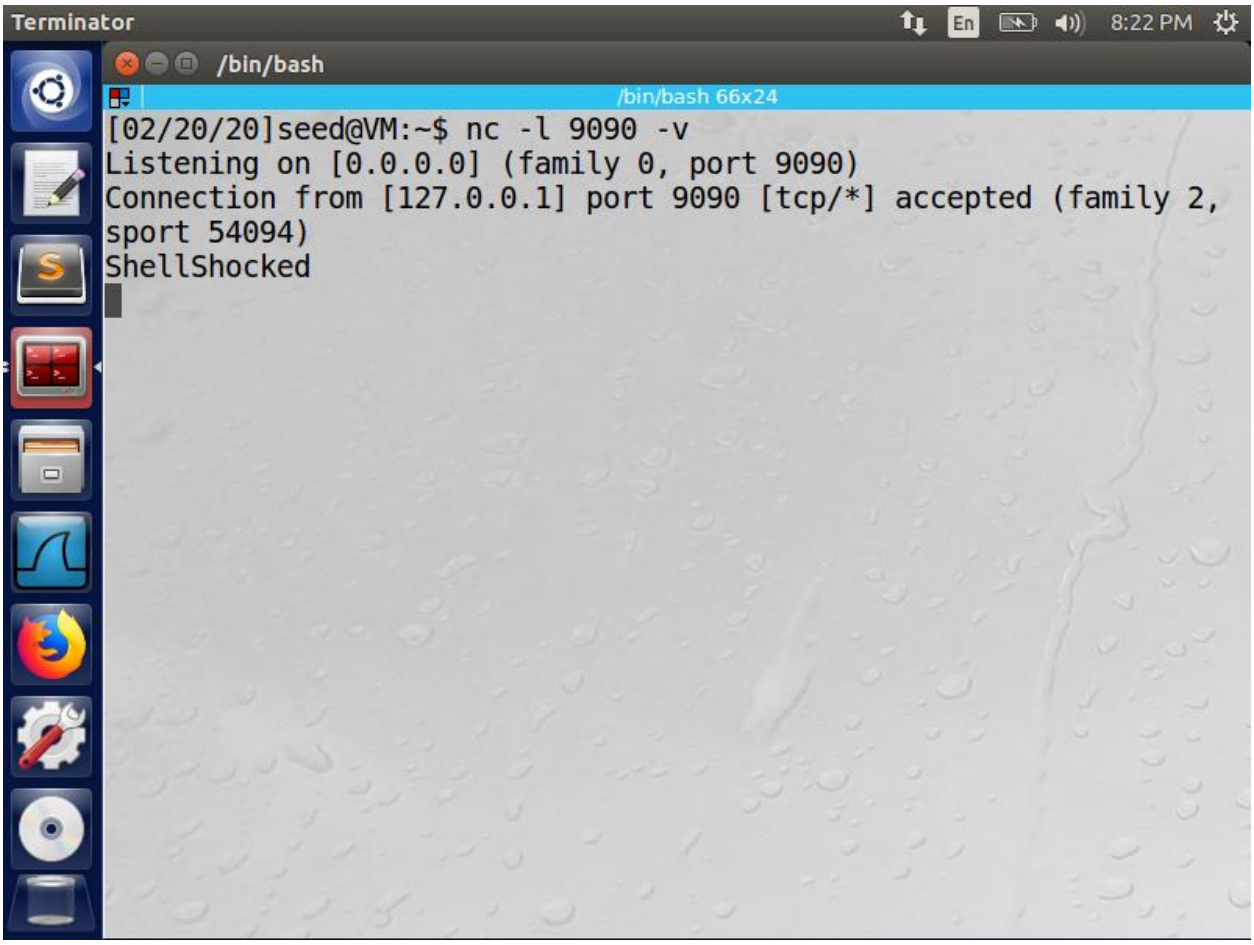


This is the attacker's machine. When I gave the ls command in the victims machine I am able to view the files of the victim in the attackers machine. Hence reverse shell has been created on the attacker's machine.

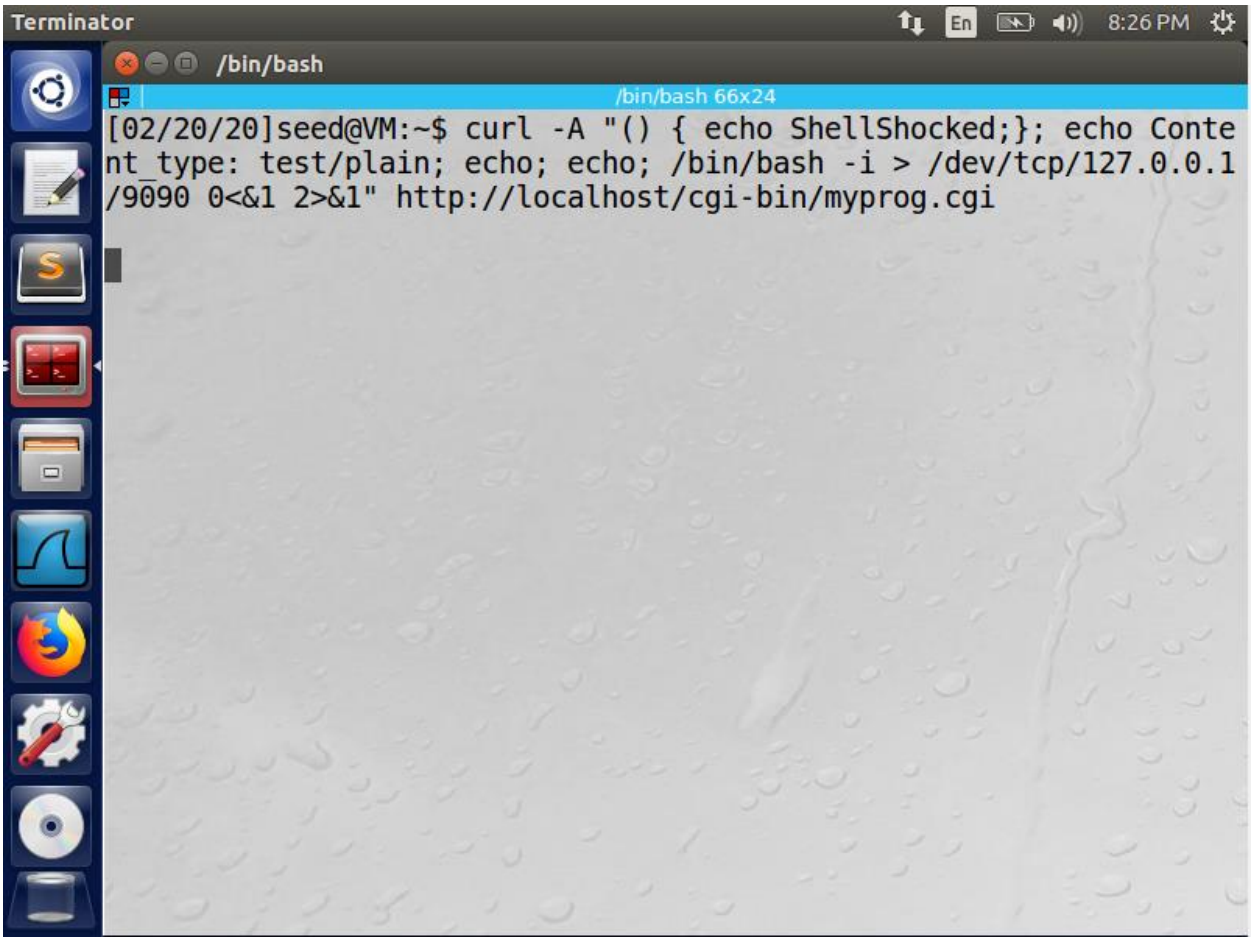


Now when I give the echo statement in the victims machine I will be able to view the statement used in the echo command in the attackers machine. This is due to the invocation of the reverse shell.

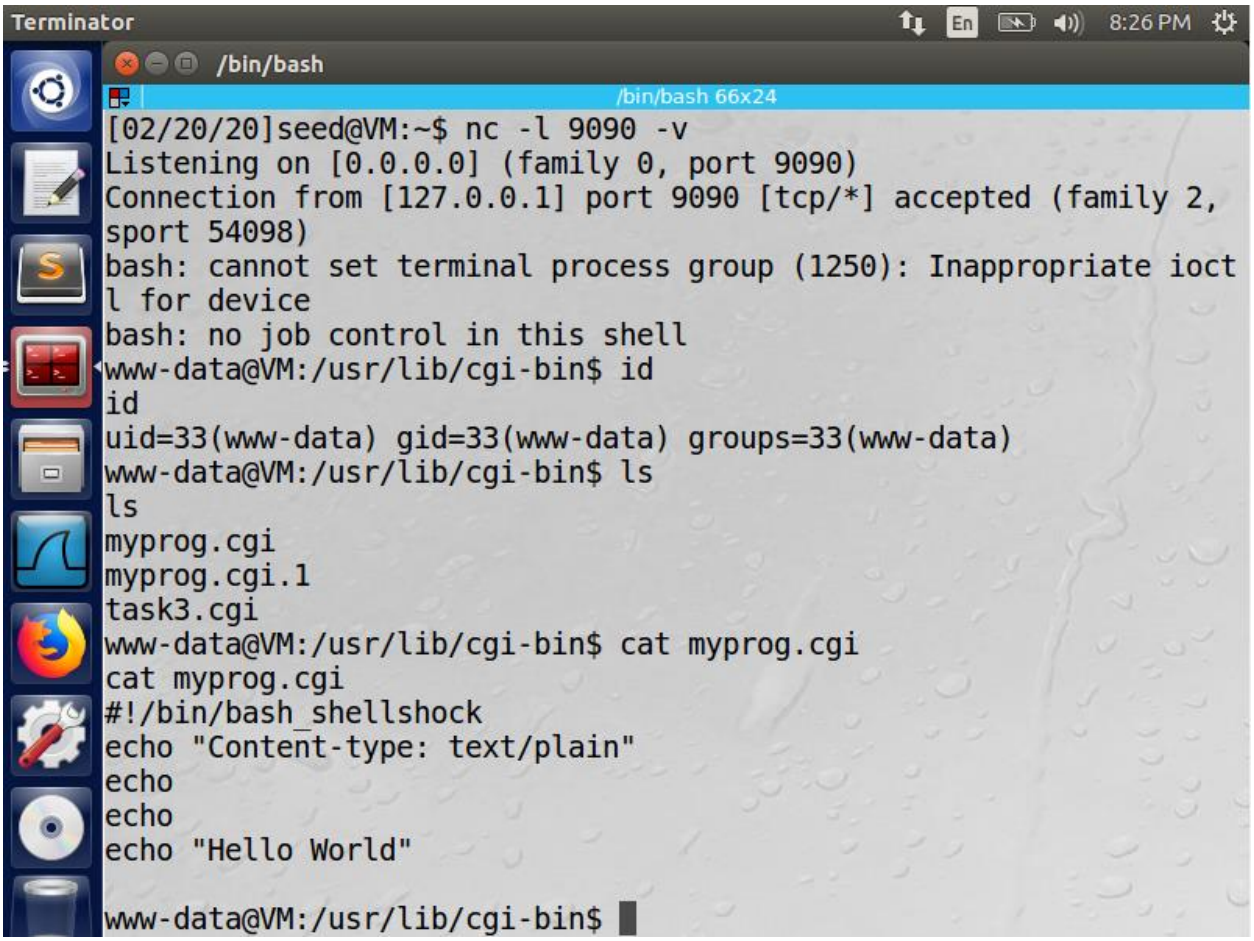
This is the attackers machine. When I gave the echo statement in the victims machine, I am able to see the statement 'ShellShocked' getting printed in the terminal of the attackers machine.



Now I use the curl command to invoke the reverse shell. I made the reverse shell to be interactive, using the -i command and make the shell to be redirected to the TCP connection. I also make the standard output and the standard input redirect to the standard input of the attackers machine. Now when I run the command on the victims machine I can see that the attacker can get into the root of the victims system.



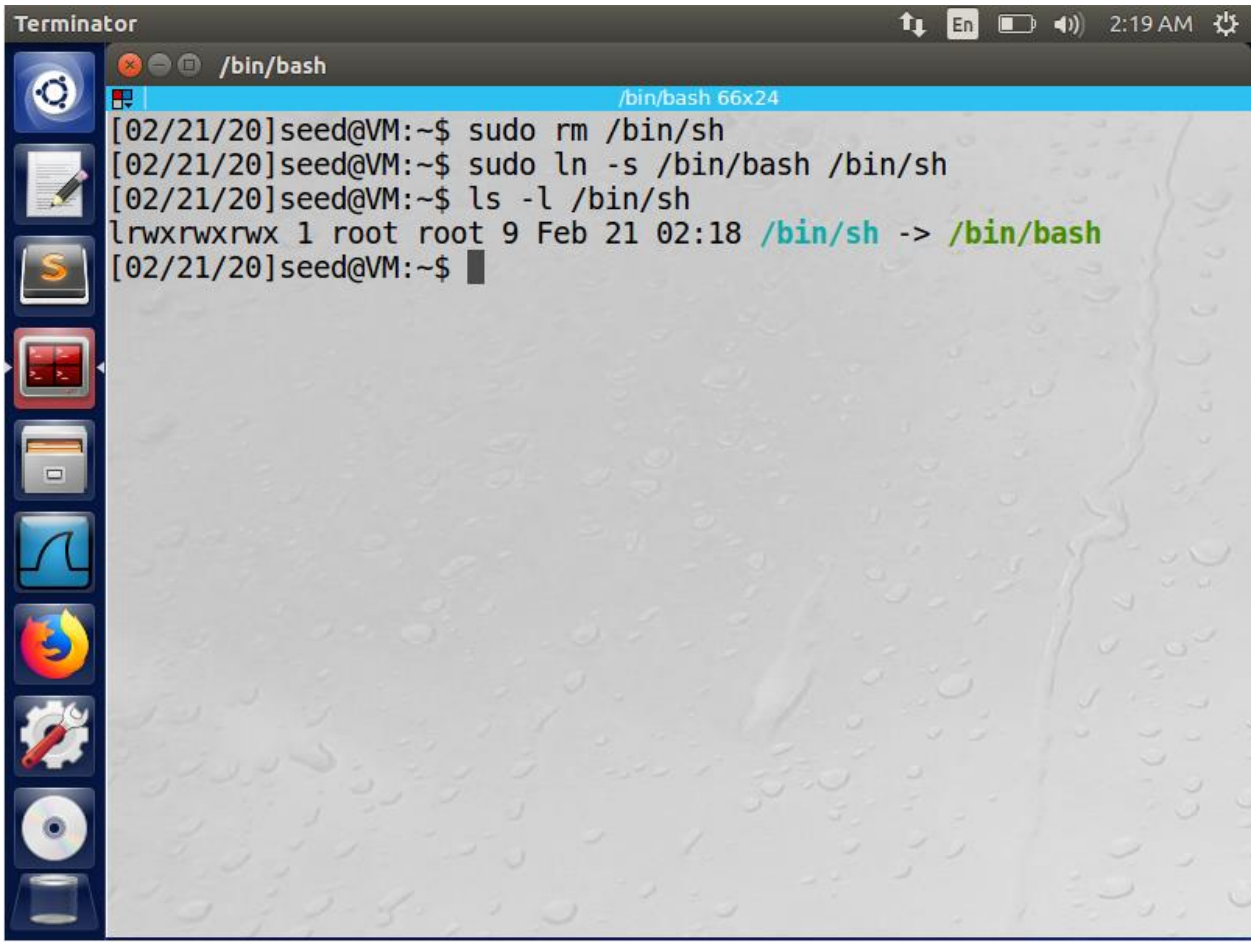
Now I am able to see the attacker is able to get the access to the root of the victims machine as www-data@vm. Now when I give the id command in the attackers terminal I am able to the uid, gid and the groups of the www-data which is the root of the victims machine. When I use the ls command in the attackers machine I will be able to view list of files from the current working directory. I also used the cat command to view the contents of the myprog.cgi and I am able to view the contents of the file which is in the victims machine. This is because of the shellshock vulnerability present in the bash, which executes the shell commands after the function definition. Shellshock vulnerabilities also makes the shell reversible to make the shell interactive between the attacker and the victims machine.



2.6 Task 6: Using the Patched Bash

Output:

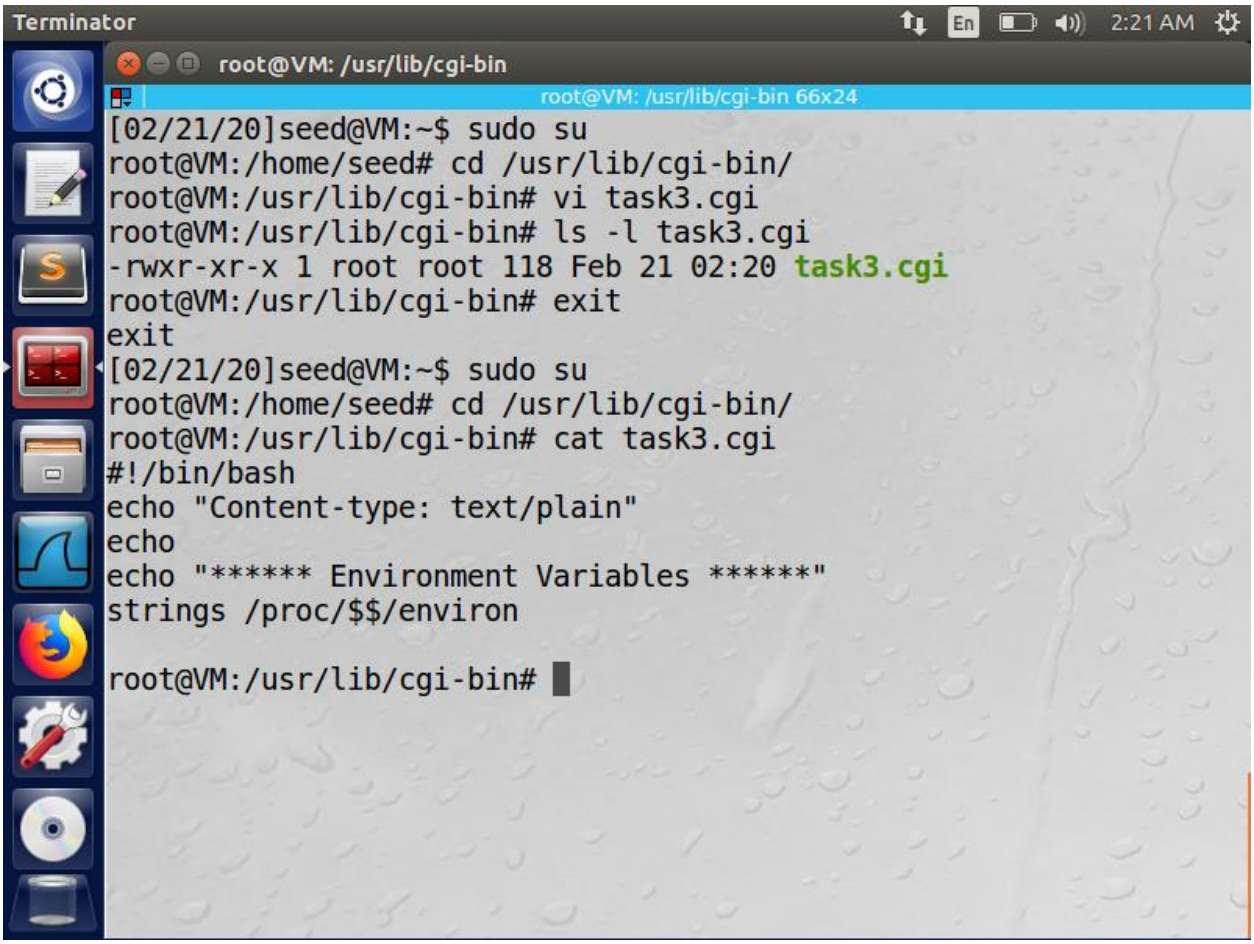
I used the sudo command to the remove the /bin/sh bash and then create a symbolic link to bash which is patched. /bin/bash is not vulnerable to the shellshock attack. And hence we use that bash to perform this task.



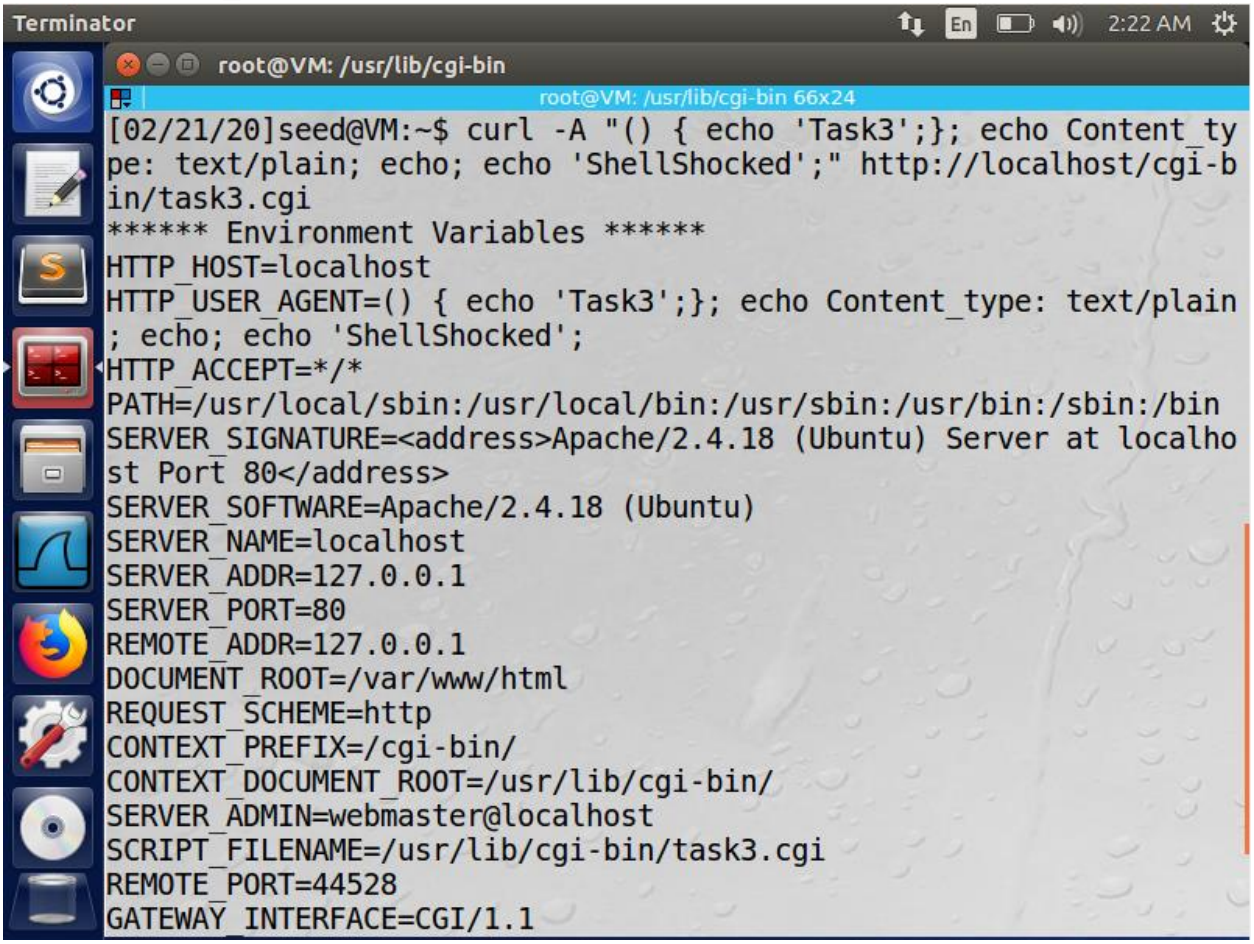
Task3:

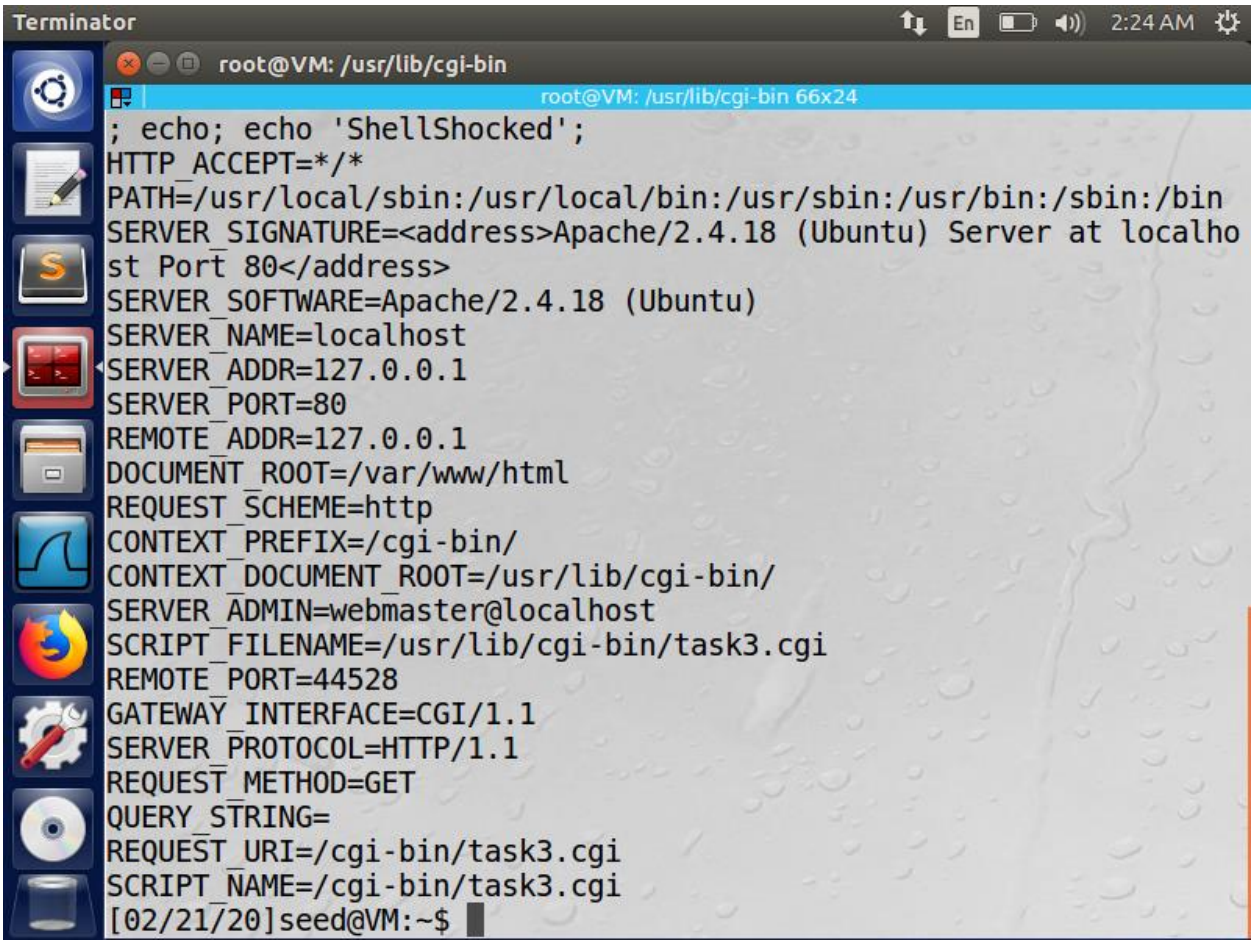
Output:

Using the sudo command I gained the access of the root user. I then navigated to the /usr/lib/cgi-bin using the cd command. I edited the task3.cgi file from #!/bin/bash_shellshock to #!/bin/bash and saved it in the same directory. I then exit from the root user.



Now I ran the curl command using the function definition and adding other shell commands with it and executed it. I was able to see that the shell command after the function definition of the shell variable was not executed. The statement 'ShellShocked' in the echo statement did not get printed. This is because we executed the curl command in the /bin/bash shell and removed the bash_shellshock line and replaced it with bash, which is already patched and is not vulnerable to the shellshock attacks. When we ran the same command with bash_shellshock we were able to see the statement getting printed because we ran using the bash_shellshock which is vulnerable to the shellshock attack.

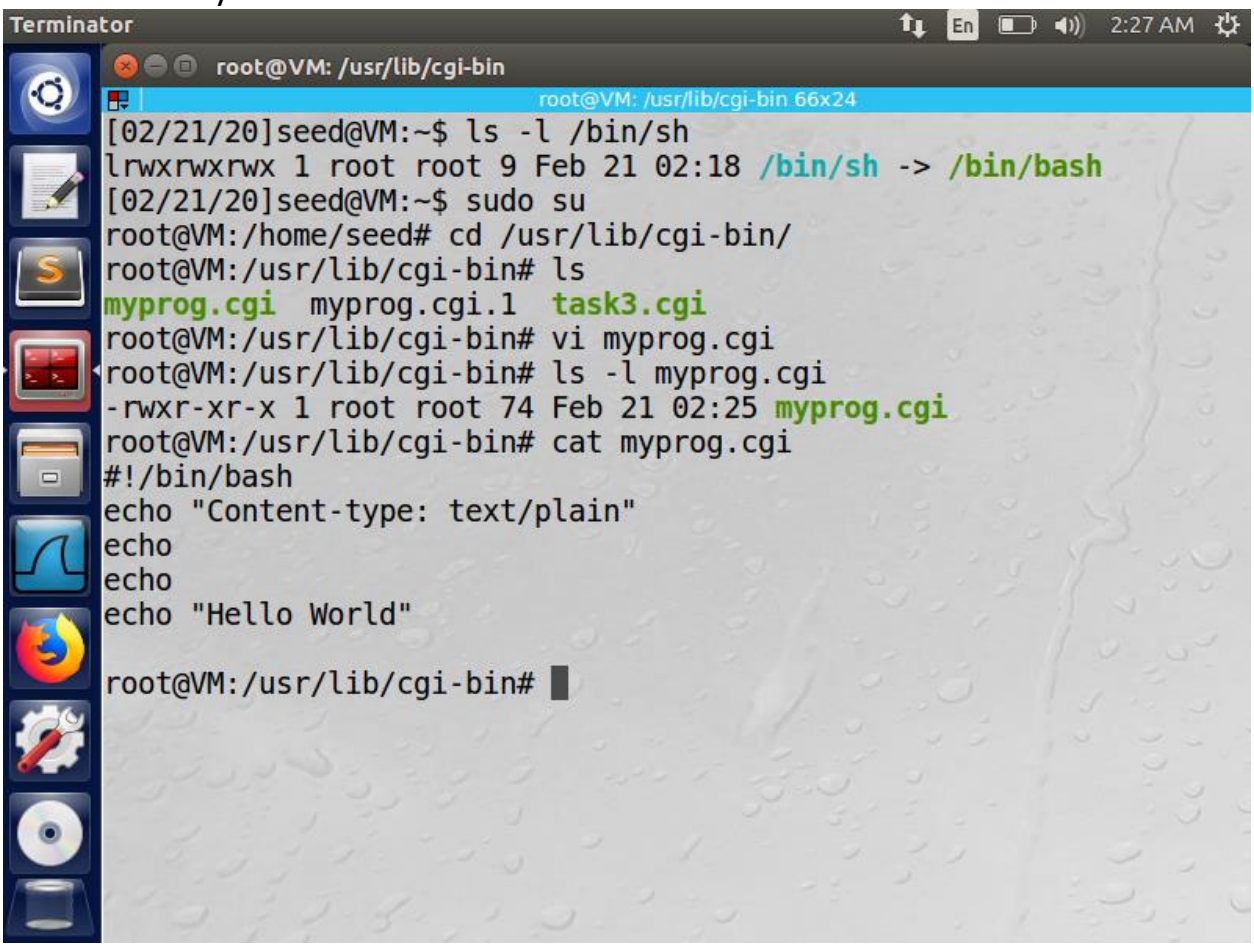



Terminator
2:24 AM
root@VM: /usr/lib/cgi-bin
root@VM: /usr/lib/cgi-bin 66x24
; echo; echo 'ShellShocked';
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localho
st Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (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/task3.cgi
REMOTE_PORT=44528
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/task3.cgi
SCRIPT_NAME=/cgi-bin/task3.cgi
[02/21/20]seed@VM:~$

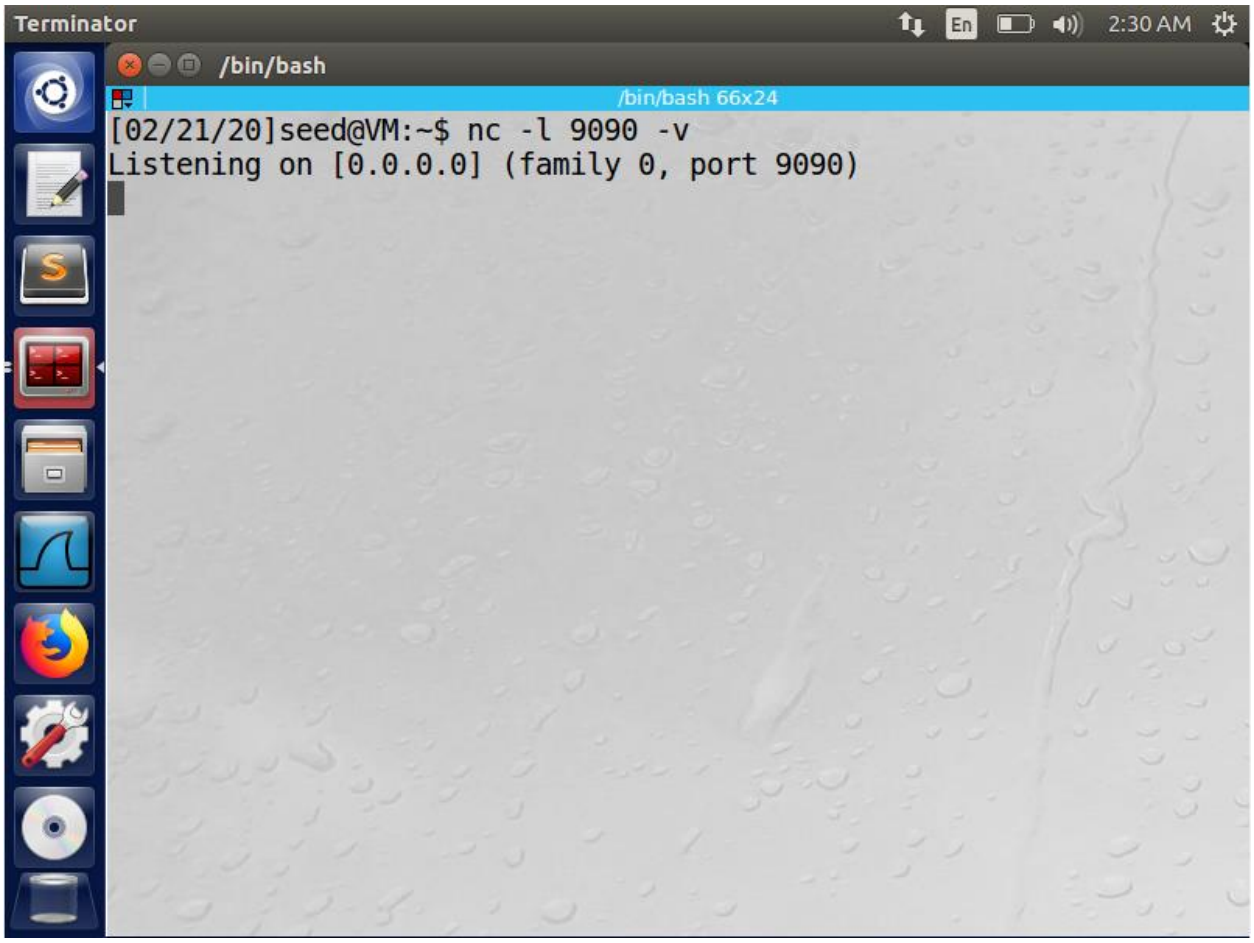
Task 5:

Output:

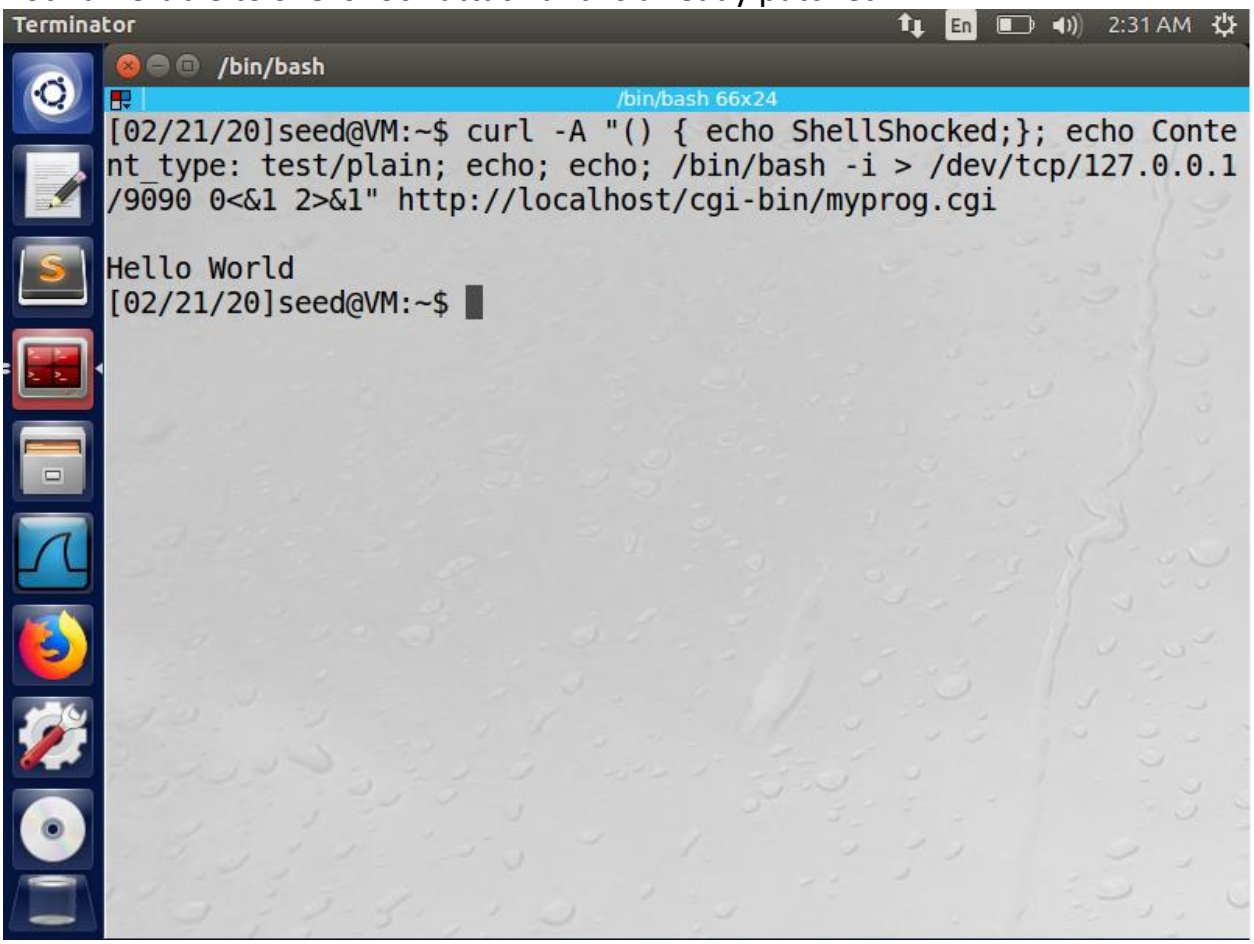
I checked to see if the bash is pointing to the bash which is patched and is not vulnerable to the shellshock attack. Then I gained the root access using the sudo command. I then navigated to the /usr/lib/cgi-bin using the cd command. I edited the myprog.cgi file from #!/bin/bash_shellshock to #!/bin/bash and saved it in the same directory. I then exit from the root user.



This is the attackers machine. I used the netcat(nc) command to listen to the victims machine. I used nc -l 9090 -v to establish the connection from the attckers machine.



This is the victims machine. I ran the curl command to make the shell interactive using the reverse shell concept. I made the reverse shell to be interactive, using the -i command and make the shell to be redirected to the TCP connection. I also make the standard output and the standard input redirect to the standard input of the attackers machine. Now when I ran the command I am not able to get the root access to the on the attackers machine. I am only able to see the Hello World statement getting printed which is present in the myprog.cgi. The statement after the function definition is not getting printed because of the bash shell which is not vulnerable to shellshock attack and is already patched.



This is the attcker's machine. When I ran the curl command on the victims machine, I should have got the root access of the victim machine on the attackers terminal. But I am not able to see the root access of the victim. This is because of the /bin/bash which is not vulnerable to shellshock attack.