**Lab Report #2**

**Name: Nicolas Rios CIS 495 + 595**

Students need to submit a detailed lab report to describe what they have done, what they have observed, and how they interpret the results. Reports should include evidences to support the observations. Evidences include packet traces, screenshots, etc.

[ENVIRONMENT SETUP 1](#_TOC_250001)

1. DNS SETTING 1
2. CONTAINER SETUP AND COMMANDS 3
3. WEB SERVER AND CGI 7

[LAB TASKS 9](#_TOC_250001)

1. EXPERIMENTING WITH BASH FUNCTION 9
2. PASSING DATA TO BASH VIA ENVIRONMENT VARIABLE 12
3. LAUNCHING THE SHELLSHOCK ATTACK 17
4. GETTING A REVERSE SHELL VIA SHELLSHOCK ATTACK 20
5. USING THE PATCHED BASH 21
6. SUMMARY 22

# Environment Setup

1. DNS Setting
   * By accessing the etc directory and using sudo to edit the hosts file, I was able to add the web server container’s IP address as well as the hostname of the server. This is to ensure that our VM can find the domain as quickly as possible.

|  |
| --- |
| **Screenshots as Evidence** |
|  |

1. Container Setup and Commands
   * For this part of the setup, I modified the docker-compose file by adding another container aside from the victim, which I identified as *attacker* (IP Address 10.9.0.70). This second container is within the same network as the victim container. I then executed the dcbuild command to create both of these containers, followed by the dcup command to start the containers, and finally the dockps command to find out the ID of each of containers.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Modifying *docker-compose.yml* |  |
| Running the dcbuild command to create the containers |  |
| Running the dcup and dockps commands |  |

1. Web Server and CGI
   * In this part of the setup, I switched to the victim container by using the docksh command, and then located the vul.cgi file that we will be using in later parts of the lab.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Step 1 |  |

# Lab Tasks

1. Experimenting with BASH Function
   * For this task, I utilized the victim container to experiment on the bash\_shellshock file that it contains. To test this, I created an environment variable named *foo*, that I declared with a function along with an extra shell command that would echo “extra”. I then exported this environment variable to the child process and ran the bash\_shellshock program. This resulted in the word “extra” being printed out since the program parsed the function definition and then executed the shell command, due to a logic error in the parse\_and\_execute function in bash\_shellshock. After this successful attack, I performed the same action on the regular /bin/bash program. However, I was not able to have this file run and my defined foo environment variable was not able to be exported to the child process.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Using the vulnerable *bash\_shellshock* |  |
| Using the regular VM /bin/bash |  |

1. Passing Data to BASH via Environment Variable
   * In this task, I initially ran the getenv.cgi file just within the directory and then using the curl command without any options/data injection. This showed that the program worked properly and was able to print out the environment variables of the HTTP request. By changing the -A (user agent) option, the -e (referrer) option , and -H (header) option, it was made clear that I was able to inject data within the HTTP header request environment variables. Since the getenv.cgi file calls on the bin/bash\_shellshock program for execution when I use these curl options I can inject commands into the header that will be executed by the bash\_shellshock program. When the webserver forks a child process to run the CGI program it passes the HTTP\_USER\_AGENT data as an environment variable to the cgi program, however in these commands we have maliciously defined this environment variable.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Running *getenv.cgi* (no data injection) |  |
| Testing -A option for curl command |  |
| Testing -e and -H options for *curl* command |  |

1. Launching the ShellShock Attack
   * For the task 3.A, I first switched to the attacker container to send my malicious remote request to the victim container. Then I utilized the -A option for the curl command to define the HTTP\_USER\_AGENT environment variable. I then followed the same function definition as in the previous attack of Task 1, by defining some arbitrary function to be parsed by the bash\_shellshock program. I then followed this up with the CGI output protocol defined in the lab tasks to have my output be returned as plain text. Finally, I then used the cat command to display the contents of the etc/passwd file and directed my attack to the victim container IP address and vul.cgi file.
   * For the task 3.B, I utilized the same attack technique as in 3.A, however I altered the shell command to /bin/id so that the server returns the process’ user ID. This resulted in the user ID of 33 as shown in the second screenshot for this task.
   * For the task 3.C, I utilized the same techniques as the previous tasks, however I altered the exploited code to first change to the /tmp directory and then utilizing the /bin/touch command to create a new file called sample.txt. I then utilized another shellshock attack to list the files within the /tmp directory from the attacker machine.
   * For the task 3.D, I altered the exploited code to use the /bin/rm command to delete the sample.txt file from the /temp directory and then used the /bin/ls shellshock attack to list the files within the /tmp directory. It listed no files after this attack.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| 3.A: Getting server to send back contents of the /etc/passwd file |  |
| 3.B: Get the server to tell you its process’ user ID. |  |
| 3.C: Get the server to create a file inside the /tmp folder. |  |
| 3.D: Get the server to delete the file that you just created inside the /tmp folder. |  |
| Question 1 | No, I will not be able to steal the content of the victim /etc/shadow because the Apache process has a user ID of 33 meaning that it does not have root privileges, which are necessary for viewing the file. |
| Question 2 | By attempting to attach some shell commands to the end of this url query, I was not successful in achieving a Shellshock attack. I believe that this ability to attach user data after the “?” mark does not get passed into the enviornment variables of the new process otherwise the attack should have resulted in some output. |

1. Getting a Reverse Shell Using the ShellShock Attack
   * In this task, the idea is for the victim machine to first create an interactive bash shell to initiate a TCP communications gateway with the attacker machine through the use of the curl command with the cgi program. The file descriptors used within this command redirect the input for the shell commands to come from the TCP connection with the attacker and will also do the same for the outputs of the commands. This creates a shell that the attacker can use rather than sending lines of code through the curl command each time. On the opposing side, the attacker has the netcat program open and listening on the 9090 port waiting to receive a connection from the server/victim machine. Once the server replies with this connection the attacker’s netcat program initiates and takes control of the bash shell program running on the server/victim machine.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Server/Victim Side |  |
| Attacker Side |  |

1. Using the Patched Bash
   * In this task, I modified the code of the vul.cgi file to the patched /bin/bash program rather than the /bin/bash\_shellshock program on the victim/server side. I then attempted to repeat the same Shellshock attack strategies from Task 3 on the attacker container. After executing numerous commands, it was evident that the patched /bin/bash program was only parsing and executing the echo “Hello World” defined in the vul.cgi program. The /bin/bash program did not execute my additional shell commands defined within the environment variables of the HTTP\_USER\_AGENT value.

|  |  |
| --- | --- |
| **Screenshots as Evidence** | |
| Changing vul.cgi file on the Victim Side |  |
| Attempting Shellshock Attacks from Task 3 on Attacker Side |  |

1. Summary
   * Overall, this lab showed me the various methods in which attackers can utilize faults within the source code to overtake a system an inject data for malicious intent. By comparing and contrasting the bash\_shellshock program with the patched bash program it is clear to see the necessity of separating data from commands/functions so that users do not take advantage and obtain unauthorized privileges or commit harmful actions.