CREATED BY: Jessica Bjurlerstam

Project Introduction

Identifying and verifying vulnerabilities in OWASP Juice Shop

Background & Motive

SQL injection is one of the most well-known and serious web application vulnerabilities. It is also included in OWASP top 10 (place A05:Injection) which highlights its relevance in modern web security.

SQL Injection represents an input validation vulnerability where user input is not properly separated from SQL query. It exists in web applications because they often rely on dynamic user input, such as search fields and login forms, which are directly connected to a database.

It is serious because it can expose, modify, or destroy sensitive data stored in the database.

Severity: High

OWASP (2025): A05 – Injection

CWE: CWE-89

Vulnerability Description

SQL Injection is a vulnerability that occurs when an application does not properly validate or sanitize user input (such as login form or search fields) before sending it to a database.

A threat actor can exploit this by injecting malicious conditions into the query. This can allow the attacker to extract sensitive information, modify database content, or affect system performance.

In this case, the vulnerability is a Blind SQL Injection (Boolean-based and Time-based). A threat actor can manipulate database queries to extract sensitive information, modify stored data, or affect system performance.

Boolean-based = Uses true/false conditions to see if a statement works.

Time-based = Uses response delay to see if a statement is true.

Demo & Testing Approach

Testing workflow

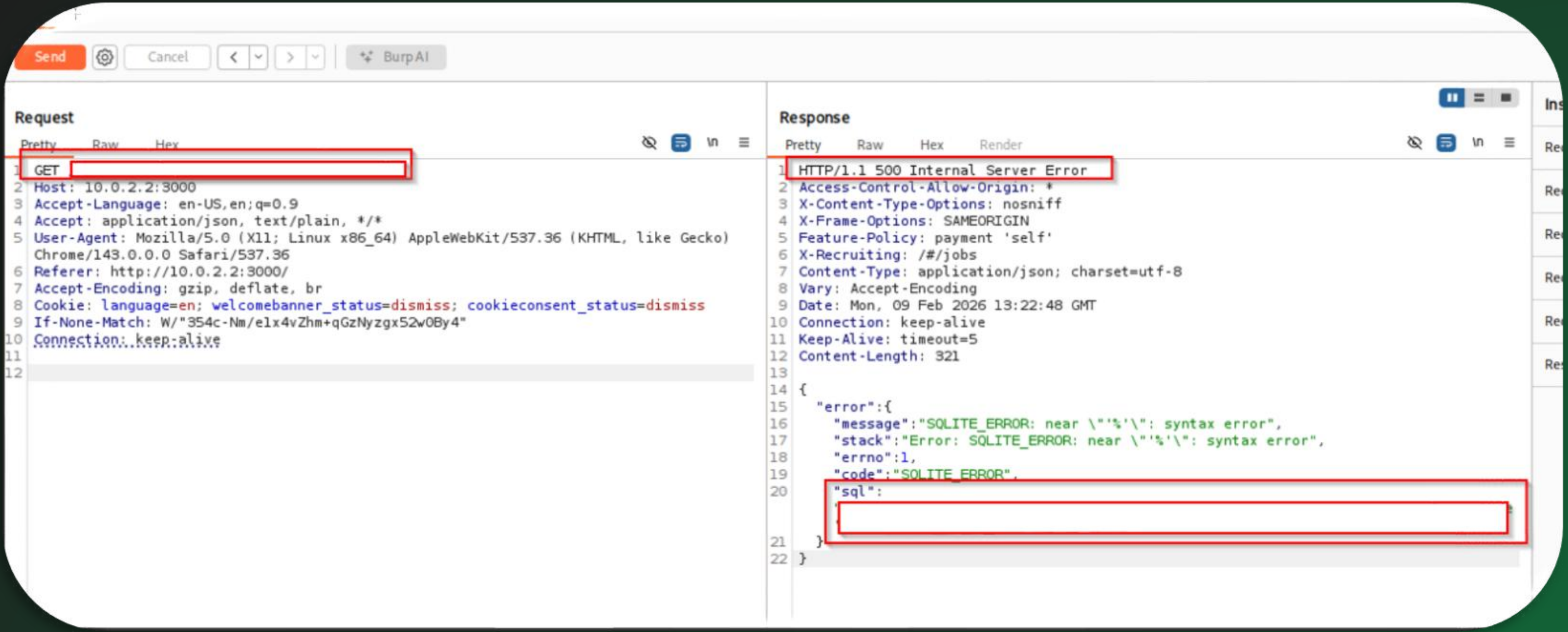
Testing was used with Burp Suite for request capture and sqlmap for injection tests.

The process involved intercepting a GET request to the search endpoint and modifying a parameter.

Sqlmap was used to confirm blind and time-based injection.

Sensitive info and payloads are excluded from slides to prevent exploitation before patching.m

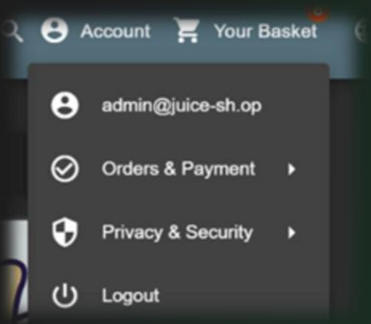
Evidence & Findings



```
GET [redacted]
Host: 10.0.2.2:3000
Accept-Language: en-US,en;q=0.9
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/143.0.0.0 Safari/537.36
Referer: http://10.0.2.2:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss
If-None-Match: W/"354c-Nm/e1x4vZhm+qGzNyzgx52w0By4"
Connection: keep-alive
```

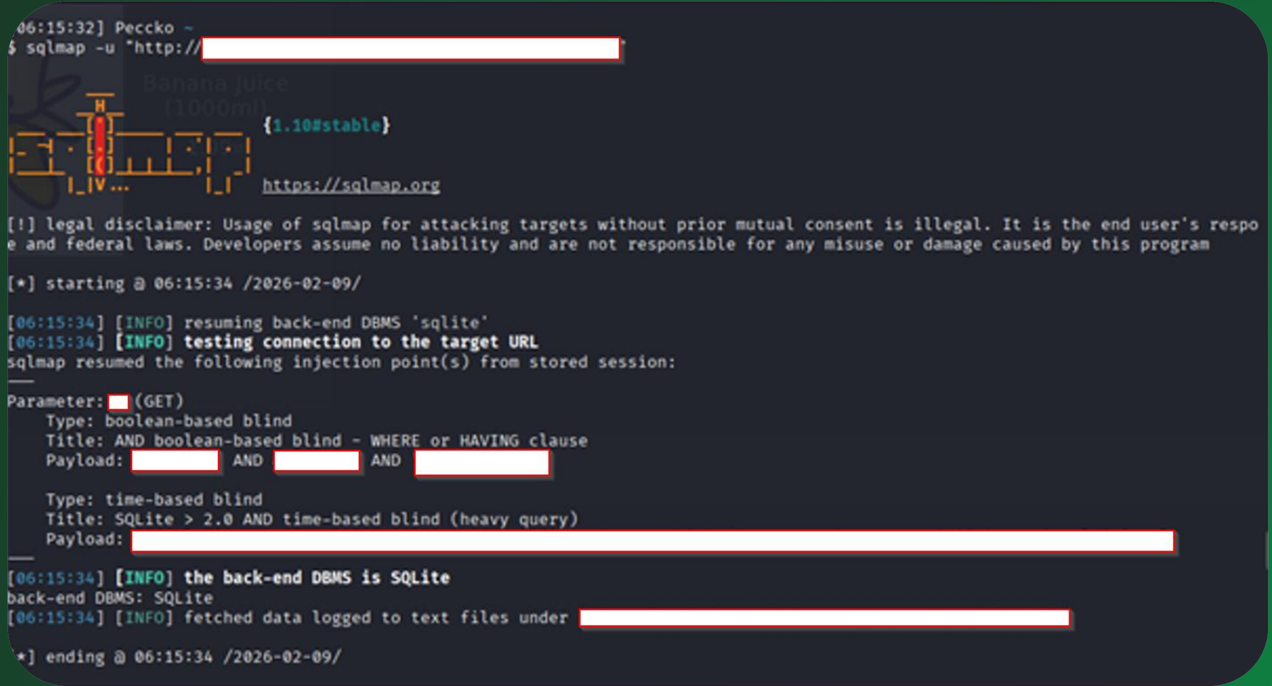
```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Mon, 09 Feb 2026 13:22:48 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 321

{
  "error":{
    "message":"SQLITE_ERROR: near \"'%'\": syntax error",
    "stack":"Error: SQLITE_ERROR: near \"'%'\": syntax error",
    "errno":1,
    "code":"SQLITE_ERROR",
    "sql":
    [redacted]
  }
}
```



sqlmap executed against the endpoint confirms that the parameter is vulnerable to SQL Injection (both Boolean-based and Time-based).

A GET request intercepted in Burp Suite shows the parameter modified, resulting in an Internal Server Error. The response includes a database-related error message, indicating improper SQL handling and confirming the presence of a SQL Injection vulnerability.



```
06:15:32] Peccko ~
$ sqlmap -u "http://[redacted]"

        {1.10#stable}

        https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's respo
e and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 06:15:34 /2026-02-09/

[06:15:34] [INFO] resuming back-end DBMS 'sqlite'
[06:15:34] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: [redacted] (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: [redacted] AND [redacted] AND [redacted]

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: [redacted]
---
[06:15:34] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[06:15:34] [INFO] fetched data logged to text files under [redacted]

*] ending @ 06:15:34 /2026-02-09/
```

Recommendations

```
query = "SELECT * FROM users WHERE email = '" + userInput + "'"
```

```
query = "SELECT * FROM users WHERE email = ?"
cursor.execute(query, (userInput,))
```

To eliminate or reduce SQL Injection, implement parameterized SQL queries (prepared statements) so that SQL code is separated from user input and cannot be interpreted as executable commands.

Avoid returning detailed SQL or database error messages to users; instead, present generic error responses to reduce information leakage that could aid the attackers.

Together these measures reduce the risk of data exfiltration and manipulation.

Conclusion & Evaluation

1. The project demonstrated that OWASP Juice Shop is vulnerable to blind SQL-Injection, enabling data extraction and manipulation while impacting confidentiality and integrity.

2. Implemented testing validated the issue and met project goals, though future work could include broader endpoint coverage and more controlled testing with other tools.

3. Always patch and update.

4. Never trust anyone.

5. Test continuously.

Q&A

Questions?