Lab5

Subjects	FSCT8561 - Security Applications
Date	@February 12, 2026

FSCT8561/Lab5 at main · ashwnn/FSCT8561

Coursework and projects from BCIT FSCT 8561 focused on building practical security and defensive networking tools. - ashwnn/FSCT8561

https://github.com/ashwnn/FSCT8561/tree/main/Lab5

ashwnn/**FSCT8561**

Coursework and projects from BCIT FSCT 8561 focused on building practical security and defensive networking tools.

1 Contributor · 0 Issues · 0 Stars · 0 Forks

Showcase

HTTP Scanner

Running the script running on the endpoint we get the following result.

```
==================================================
[INFO] Starting HTTP scan
[INFO] Target: http://vm-ubuntu:3000
==================================================
[INFO] GET / payload={} status=200 len=75055
Low Severity: missing header Content-Security-Policy
Low Severity: missing header Strict-Transport-Security
--------------------------------------------------
[INFO] GET /rest/products/search payload={'q': 'apple'} statu
s=200 len=631
Low Severity: missing header Content-Security-Policy
Low Severity: missing header Strict-Transport-Security
--------------------------------------------------
```

```
[INFO] GET /rest/products/search payload={'q': 'test'} status
=200 len=517
Low Severity: missing header Content-Security-Policy
Low Severity: missing header Strict-Transport-Security
--------------------------------------------------
[INFO] POST /rest/user/login payload={'email': 'test@test.co
m', 'password': 'test123'} status=401 len=26
Low Severity: missing header Content-Security-Policy
Low Severity: missing header Strict-Transport-Security
--------------------------------------------------
[INFO] Done
```

Looking at this we can see all of the pages were flagged for the same things:

- Missing header Content-Security-Policy: This means that the page can be effected more by XSS attacks, by default without Content Security Policy it is more successible for more damaging attacks.
- Missing header Strict-Transport-Security: This suggests HTTPS is not being enforced and traffic is sent over plain-text.

Vulnerability Detector

Running the scritp we get the following result

```
==================================================
[INFO] Starting vulnerability detector
[INFO] Target: http://vm-ubuntu:3000
==================================================
==================================================
Summary
==================================================
XSS findings: 0
SQLi findings: 3
[ALERT] SQL indicator: /rest/products/search with q=') OR 1=1
--
[ALERT] SQL indicator: /rest/user/login with email=test@test.
com'
```

```
[ALERT] SQL indicator: /rest/user/login with email=test@test.
com') OR 1=1--
Config findings: 2
Low Severity: missing header Content-Security-Policy on /
Low Severity: missing header Strict-Transport-Security on /
==================================================
[INFO] Done
```

After analyzing for XSS & SQL findings we can see that:

- SQL findings were flagged, this is showing that the OWASP Juice App is handling user input unsafely, because certain SQL-looking inputs caused server responses that are not expected. This showsthe rist of SQL injection on the search and login pages.

Reflection Questions

1. In Lab 4, we saw plaintext passwords in a PCAP. In Lab 5, we saw them potentially leaked via SQLi. Which is harder to defend, and why? SQLi is harder to defend against because it requires consistently safe server-side query handling everywhere, while plaintext exposure can be solved by using HTTPS and not transmitting in cleartext.
2. How does Parameter Tampering in Python differ from simply typing in a browser URL bar? Python parameter tampering can modify and replay GET/POST data (with headers/cookies/body) and can be automated, whereas the URL bar is a limited manual change to visible query parameters.
3. If an application is encrypted (HTTPS), can Scapy (Lab 4) still see the SQLi payloads? Can your Python script (Lab 5) still see the responses? With HTTPS, Scapy can capture packets but not read the encrypted the payloads, while Lab5 script can send payloads and read responses because it terminates TLS as the client.
4. Why is "Automated Scanning" often followed by "Manual Verification"? Automated scans find common issues faster, and with manual verification can be used to confirm if they are real and find false positives too.

Security Analysis

The first fix of the Juice Shop is the possible SQL injection risk. There were 3 SQL indicators falgged on two seperate endpoints: `/rest/products/search` and `/rest/user/login`. Even if these are only indicators and not full proof, SQL is very critical and has a high impact because it can lead to real data exposure and access to the database. The first step is reviewing how the inputs are used in database queries and replace any string-built SQL with parameterized queries. Debug also seems to be on as there are detailed database errors from being returned to users which is a huge issue, and make sure the database account used by the app has limited permissions.

Looking at the noise factor, automated scanning can create patterns that an Lab4 script can notice. Because the script sends repeated requests and unusual inputs like quotes and OR `1=1`, which look very suspicious from what's excpecrted from a normal user. Because we sniffed plain HTTP traffic, it can detect it and read the payload and match simple signatures. If the site is using HTTPS, we cannot read the exact SQL strings in the packets, but maybe due to high traffic we can detect it, especially repeated ones to the same endpoint, and lots of error responses like 401 or 500. In reality, scanners are much more precesice and can notice patterns, so rate limits and alert thresholds help a lot.

Finally, leakage from headers is usually lower risk than a direct exploitation but can provide really important context. Missing Content-Security-Policy and Strict-Transport-Security are hardening gaps that can make other attacks worse and more amplified, but they do not directly give an attacker access. SQL on the otherhand is a direct exploitation path, so it should be fixed. After the SQL fixes, I would add CSP, enable HTTPS, and then set HSTS on the HTTPS responses.