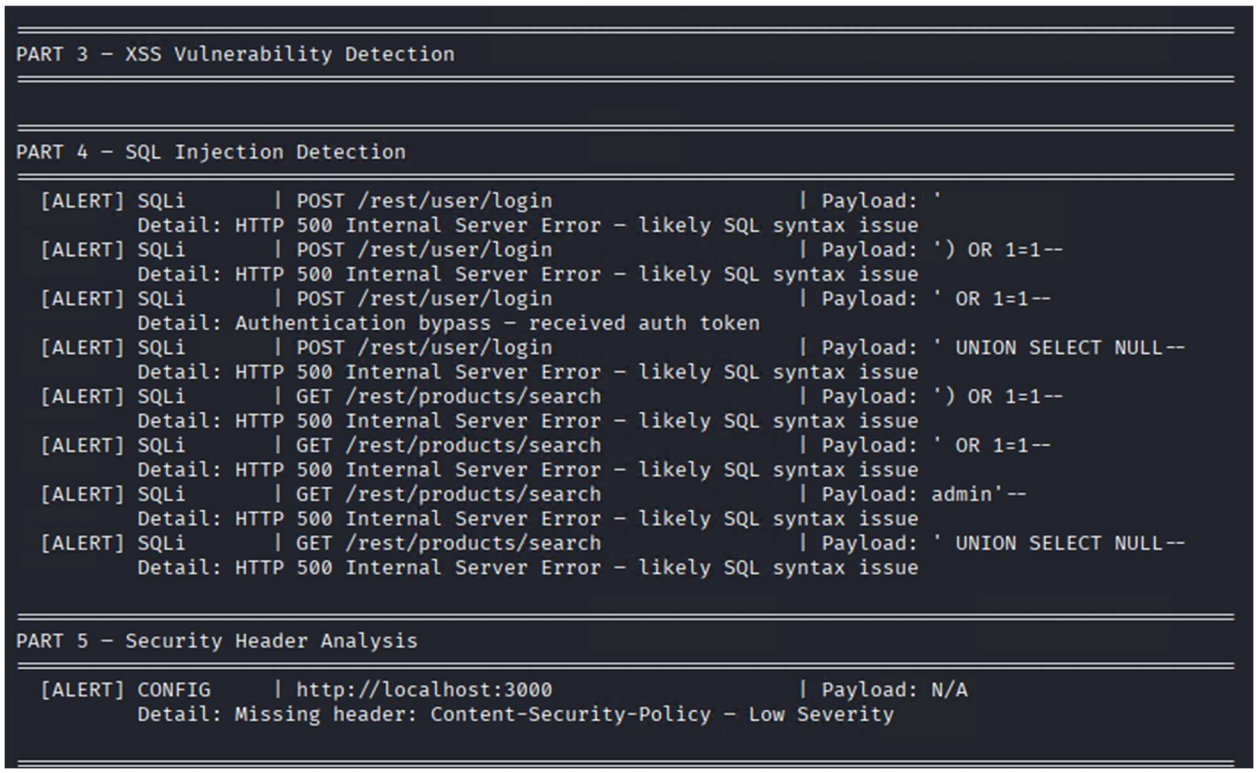
Code: https://github.com/DThomas230/FSCT-8561_Security-Applications

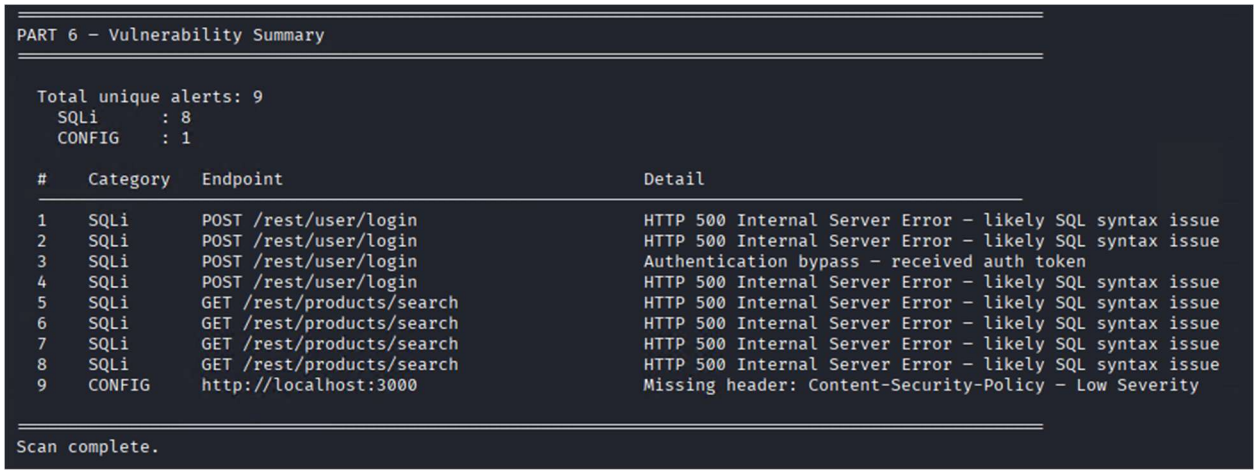
Vulnerability detector



```
PART 3 — XSS Vulnerability Detection

PART 4 — SQL Injection Detection
  [ALERT] SQLi       | POST /rest/user/login                    | Payload: '
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue
  [ALERT] SQLi       | POST /rest/user/login                    | Payload: ') OR 1=1--
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue
  [ALERT] SQLi       | POST /rest/user/login                    | Payload: ' OR 1=1--
          Detail: Authentication bypass — received auth token
  [ALERT] SQLi       | POST /rest/user/login                    | Payload: ' UNION SELECT NULL--
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue
  [ALERT] SQLi       | GET /rest/products/search                | Payload: ') OR 1=1--
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue
  [ALERT] SQLi       | GET /rest/products/search                | Payload: ' OR 1=1--
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue
  [ALERT] SQLi       | GET /rest/products/search                | Payload: admin'--
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue
  [ALERT] SQLi       | GET /rest/products/search                | Payload: ' UNION SELECT NULL--
          Detail: HTTP 500 Internal Server Error — likely SQL syntax issue

PART 5 — Security Header Analysis
  [ALERT] CONFIG     | http://localhost:3000                    | Payload: N/A
          Detail: Missing header: Content-Security-Policy — Low Severity
```



```
PART 6 — Vulnerability Summary

  Total unique alerts: 9
    SQLi      : 8
    CONFIG    : 1

  #    Category   Endpoint                          Detail
  1    SQLi       POST /rest/user/login             HTTP 500 Internal Server Error — likely SQL syntax issue
  2    SQLi       POST /rest/user/login             HTTP 500 Internal Server Error — likely SQL syntax issue
  3    SQLi       POST /rest/user/login             Authentication bypass — received auth token
  4    SQLi       POST /rest/user/login             HTTP 500 Internal Server Error — likely SQL syntax issue
  5    SQLi       GET /rest/products/search         HTTP 500 Internal Server Error — likely SQL syntax issue
  6    SQLi       GET /rest/products/search         HTTP 500 Internal Server Error — likely SQL syntax issue
  7    SQLi       GET /rest/products/search         HTTP 500 Internal Server Error — likely SQL syntax issue
  8    SQLi       GET /rest/products/search         HTTP 500 Internal Server Error — likely SQL syntax issue
  9    CONFIG     http://localhost:3000             Missing header: Content-Security-Policy — Low Severity

Scan complete.
```

Http Scanner

```
└─PS> python ./http_scanner.py

  Point-in-Time Vulnerability Assessment
  Target: OWASP Juice Shop @ http://localhost:3000

═══════════════════════════════════════════════════════════════════════════════
PART 1 — Reconnaissance (Attack Surface & Server Fingerprinting)
═══════════════════════════════════════════════════════════════════════════════

  HTTP 200 from http://localhost:3000
  All response headers (look for information leakage):
    Access-Control-Allow-Origin: *
    X-Content-Type-Options: nosniff
    X-Frame-Options: SAMEORIGIN
    Feature-Policy: payment 'self'
    X-Recruiting: /#/jobs
    Accept-Ranges: bytes
    Cache-Control: public, max-age=0
    Last-Modified: Tue, 17 Feb 2026 22:19:32 GMT
    ETag: W/"1252f-19c6dafed3e"
    Content-Type: text/html; charset=UTF-8
    Vary: Accept-Encoding
    Content-Encoding: br
    Date: Tue, 17 Feb 2026 22:21:18 GMT
    Connection: keep-alive
    Keep-Alive: timeout=5
    Transfer-Encoding: chunked

  Endpoints being tested:
    GET   /rest/products/search  param=q
    GET   /api/Products
    GET   /api/Challenges
    POST  /rest/user/login
    GET   /api/Feedbacks
    GET   /redirect  param=to
```

```
PART 2 — HTTP Endpoint Scan
  GET   | /rest/products/search                  | Payload: test                              | Status: 200 | Length: 517
  GET   | /rest/products/search                  | Payload: admin                             | Status: 200 | Length: 30
  GET   | /rest/products/search                  | Payload: <script>alert(1)</script>         | Status: 200 | Length: 30
  GET   | /rest/products/search                  | Payload: ' OR 1=1--                        | Status: 500 | Length: 942
  GET   | /rest/products/search                  | Payload: ../../../../etc/passwd            | Status: 200 | Length: 30
  GET   | /api/Products                          | Payload: N/A                               | Status: 200 | Length: 13207
  GET   | /api/Challenges                        | Payload: N/A                               | Status: 200 | Length: 59077
  POST  | /rest/user/login                       | Payload: test                              | Status: 401 | Length: 26
  POST  | /rest/user/login                       | Payload: admin                             | Status: 401 | Length: 26
  POST  | /rest/user/login                       | Payload: <script>alert(1)</script>         | Status: 401 | Length: 26
  POST  | /rest/user/login                       | Payload: ' OR 1=1--                        | Status: 200 | Length: 799
  POST  | /rest/user/login                       | Payload: ../../../../etc/passwd            | Status: 401 | Length: 26
  GET   | /api/Feedbacks                         | Payload: N/A                               | Status: 200 | Length: 1734
  GET   | /redirect                              | Payload: test                              | Status: 406 | Length: 3090
  GET   | /redirect                              | Payload: admin                             | Status: 406 | Length: 3092
  GET   | /redirect                              | Payload: <script>alert(1)</script>         | Status: 406 | Length: 3156
  GET   | /redirect                              | Payload: ' OR 1=1--                        | Status: 406 | Length: 3110
  GET   | /redirect                              | Payload: ../../../../etc/passwd            | Status: 406 | Length: 3126
```

```
PART 5 — Security Header Analysis

  Response headers from http://localhost:3000:
    [WARNING] Content-Security-Policy: MISSING  —  Low Severity
    [WARNING] Strict-Transport-Security: MISSING  —  Low Severity
    [OK]      X-Content-Type-Options: nosniff

Scan complete.
  Missing security headers: Content-Security-Policy, Strict-Transport-Security
```

Part 7 - 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?
    a. SQLi attacks (Lab 5) are more difficult to protect against since they take advantage of application logic attacks that need more thorough input validation, parameterized queries and robust coding throughout the codebase. The solution to network traffic interception (Lab 4) is one solution

that can help solve the issue, and that is through the deployment of HTTPS encryption. SQLi vulnerabilities need continued training of the developers, code reviews, testing, and network encryption is a single infrastructure modification.

2. How does Parameter Tampering in Python differ from simply typing in a browser URL bar?
    a. Python programs such as the HTTP scanner may be used to automate parameter fuzzing on a large scale and send hundreds of malicious payloads within a short period without being limited by the browser. The drawbacks of using manually manipulated URLs are typing speed and security lockouts in browsers. The Python script is also capable of manipulating POST request bodies, headers as well as maintaining complicated authentication processes that would have been fully automatic with the use of a browser, giving it more advanced attack capabilities.
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?
    a. In HTTPS, Scapy cannot view SQLi payloads over the air since TLS encryption secures the packets of data. Nonetheless, Python script may be able to access application response since it is interacting as an authentic client - the encryption is not hidden to authorized points. The script connects encrypted links and gets decrypted response and the network sniffing tools can only observe the encrypted traffic.
4. Why is "Automated Scanning" often followed by "Manual Verification"?
    a. There are numerous false positives produced by automated scanners and also probability of overlooking context-sensitive vulnerabilities. Manual verification verifies that problematic code is all the issues in question are exploitable and what their impact would be in practice. Missing security headers may be issued as a warning by the HTTP scanner, though it is a manual test that these do not represent any real hazards in the intended application scenario and business context.

Part 7 – Security Analysis

Write 300–400 words synthesizing your findings: If you were the CISO of Juice Shop, what is the first thing you would fix based on your Python script's output?

- Discuss the "Noise" factor: Does automated scanning create enough noise for the IDS (from Lab 4) to catch it?

- Compare the risk of Information Leakage (Headers) vs. Direct Exploitation (SQLi).

According to the results of the Python HTTP scanner, as the CISO of Juice Shop, I would criminalize input validation initially and parameterize queries to seal those SQL injection vulnerabilities before even paying attention to the presence of missing security header. The scan tested endpoints such as /rest/products/search and /rest/user/login with payloads such as' OR 1=1-- and that is the actual giant danger the SQLi can provide a malicious user with complete database entry, stolen logins, and even spam in privileged access. The absence of such headers as Content-Security-Policy and Strict-Transport-Security seal the gaps of defense-in-depth, but they are rather effective to prevent the secondary attacks, but not the direct database break.

Regarding noise of detection, the automated scan transmits a huge amount of traffic patterns which should be detected by modern IDS with ease. The script will do a series of tests with apparent payloads such as <script>alert (1)</script>; and path traversal moves (../../../../etc/passwd) and will be easy to create signatures. Delays, timing shuffling, and spoofing legitimate traffic are often added by real attackers to remain off the radar, therefore, the hammer and chisel approach of this scanner is in fact simpler to detect than an accomplished attack. The high-rate characteristic proves to defenders as it appears in the form of obvious anomalies in the timing of requests, payloads, and listings- thus it raises an alarm.

Comparing the risks, SQLi is the emergency situation that should be repaired immediately, whereas the verbose headers would simply allow the attackers to carry out reconnaissance to make attacks later. The header scan which reveals versions of the tech stack and server provides valuable information to the attacker, yet it is not enough to compromise the system. The SQLi itself is the threat that facilitates the theft of data, privilege hop and system compromise. Therefore the priority ladder is: Get the SQLi fixed, and then add the security headers so that the attack surface is concealed. The Python script demonstrates how reconnaissance can reveal both of these varieties of vulnerabilities, although the timeline is to fix the exploitable material first before reacting by edifying the defensive headers.