Week 3: Advanced Security -

◇ 1. Basic Penetration Testing

Objective:

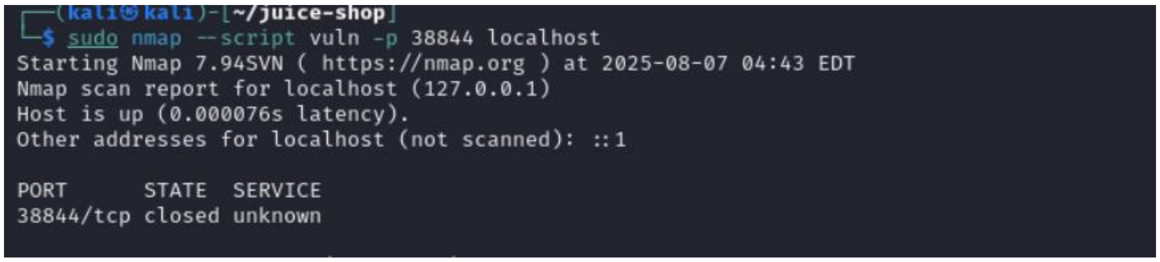
Simulate common cyberattacks using tools like Nmap to identify potential vulnerabilities in the local app.

🛠 ***Tool Used:***

Nmap – for scanning ports and running vulnerability detection scripts.

📋 ***Result:***

- Juice Shop app was active.
- Port 38844 was **closed**, so no open port vulnerabilities were detected.
- No specific vulnerabilities found during the scan.



```
┌──(kali㉿kali)-[~/juice-shop]
└─$ sudo nmap --script vuln -p 38844 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-08-07 04:43 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000076s latency).
Other addresses for localhost (not scanned): ::1

PORT      STATE  SERVICE
38844/tcp closed unknown
```

Browser-based manual testing (no Nmap used

Testing Summary:
In Week 1, we already performed and documented the following penetration tests:

- **XSS (Cross-Site Scripting):**
  Injected `<script>alert("XSS")</script>` into form fields to test client-side script execution.

- **SQL Injection:**
  Bypassed login using `' OR '1'='1` in the username field to gain unauthorized access.

- **Weak Password Storage:**
  Extracted stored hashed password and cracked it using online tools.

◇ 2. Set Up Basic Logging

Objective:

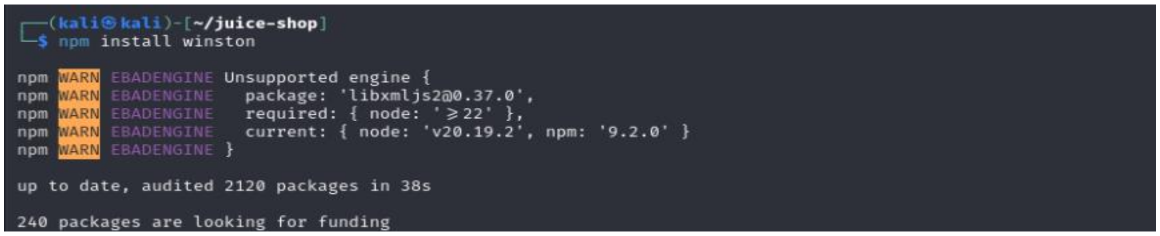
Log app-level security events using the **Winston** library to monitor and record behavior.

Tool Used:

Winston – a Node.js logging library.

Installation:

```
bash
npm install winston
```



```
┌──(kali㉿kali)-[~/juice-shop]
└─$ npm install winston

npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'libxmljs2@0.37.0',
npm WARN EBADENGINE   required: { node: '≥22' },
npm WARN EBADENGINE   current: { node: 'v20.19.2', npm: '9.2.0' }
npm WARN EBADENGINE }

up to date, audited 2120 packages in 38s

240 packages are looking for funding
```

Code Implemented:

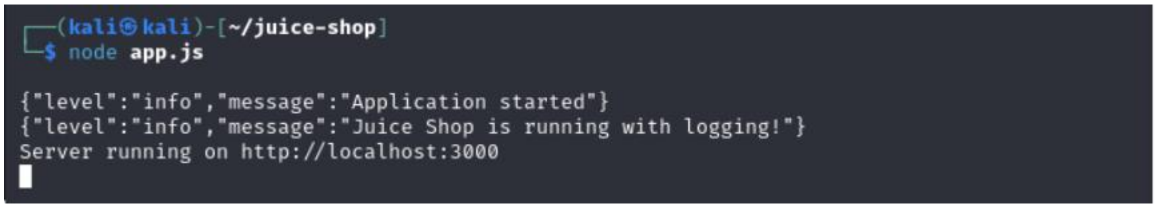
```
JavaScript
const winston = require('winston');

const logger = winston.createLogger({
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'security.log' })
  ]
});

logger.info('Application started');
```

Explanation:

- Console logs show messages during development.
- File logs are saved in security.log for audit and debugging.
- Message logged: "Application started" when the app runs.



```
┌──(kali㉿kali)-[~/juice-shop]
└─$ node app.js

{"level":"info","message":"Application started"}
{"level":"info","message":"Juice Shop is running with logging!"}
Server running on http://localhost:3000
```

Best Practices Checklist:

No.	Practice	Status
1.	Validate all user inputs to prevent XSS/SQL injection	Done
2.	Use HTTPS for secure data transfer	Not enabled in localhost
3.	Hash and salt passwords using bcrypt	Done
4.	Implement JWT for secure authentication	Done
5.	Use Helmet.js to secure HTTP headers	Done
6.	Log events using Winston	Done