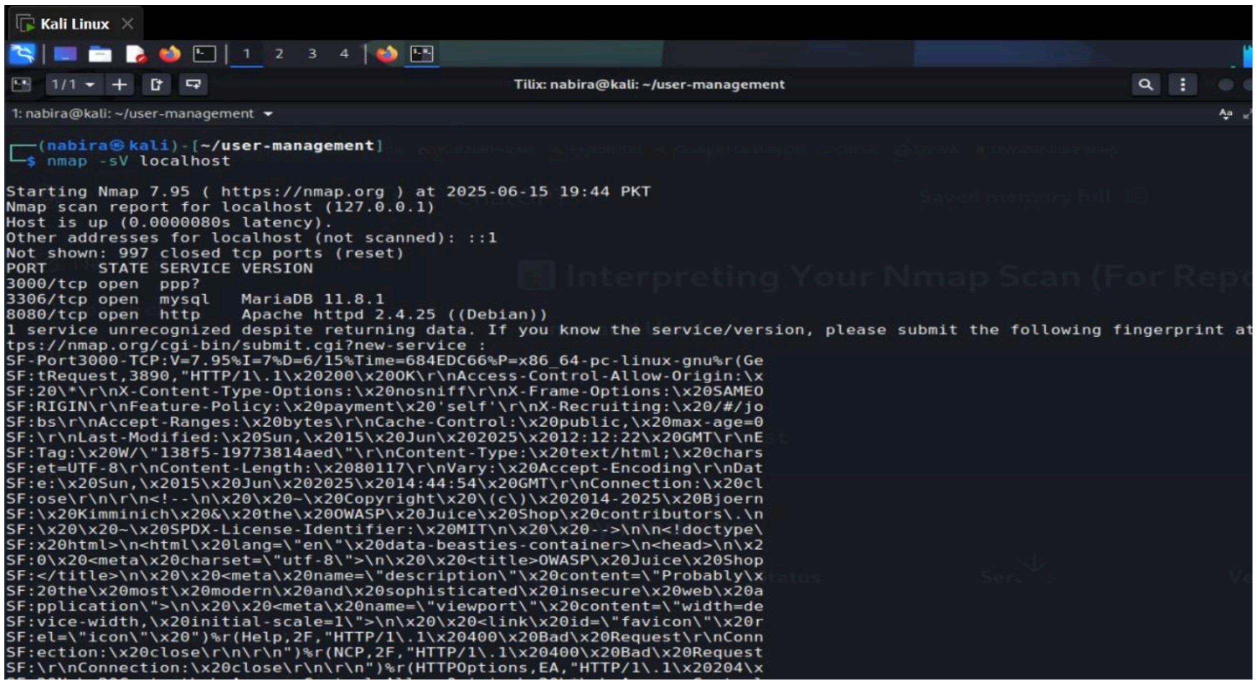
WEEK 3 - ADVANCED SECURITY AND FINAL REPORTING

In the final week of the internship, the focus was on conducting basic security testing simulations, implementing advanced logging mechanisms, and finalizing all necessary security measures and documentation. The main objective was to simulate reconnaissance attacks using tools like Nmap, implement real-time and persistent logging using Winston, and ensure that the application adhered to modern web security practices.

Simulated Reconnaissance using Nmap

A local Nmap scan was performed using the command `nmap -sV localhost` to identify open and potentially exploitable ports. The scan returned the following results:

- **Port 3000 (Juice Shop)** – Open and accessible
- **Port 3306 (MariaDB)** – Open and running MariaDB 11.8.1
- **Port 8080 (Apache HTTPD)** – Apache 2.4.25 serving web pages


Kali Linux
1
2
3
4
1/1
Tilix: nabira@kali: ~/user-management
1: nabira@kali: ~/user-management

```
┌──(nabira㉿kali)-[~/user-management]
└─$ nmap -sV localhost

Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-15 19:44 PKT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000080s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 997 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
3000/tcp open  ppp?
3306/tcp open  mysql   MariaDB 11.8.1
8080/tcp open  http    Apache httpd 2.4.25 ((Debian))
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at
tps://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port3000-TCP:V=7.95%I=7%D=6/15%Time=684EDC66%P=x86_64-pc-linux-gnu%r(Ge
SF:tRequest,3890,"HTTP/1\.1\x20200\x20OK\r\nAccess-Control-Allow-Origin:\x
SF:20\*\r\nX-Content-Type-Options:\x20nosniff\r\nX-Frame-Options:\x20SAMEO
SF:RIGIN\r\nFeature-Policy:\x20payment\x20'self'\r\nX-Recruiting:\x20/#/jo
SF:bs\r\nAccept-Ranges:\x20bytes\r\nCache-Control:\x20public,\x20max-age=0
SF:\r\nLast-Modified:\x20Sun,\x2015\x20Jun\x202025\x2012:12:22\x20GMT\r\nE
SF:Tag:\x20W/\"138f5-19773814aed\"\r\nContent-Type:\x20text/html;\x20chars
SF:et=UTF-8\r\nContent-Length:\x2080117\r\nVary:\x20Accept-Encoding\r\nDat
SF:e:\x20Sun,\x2015\x20Jun\x202025\x2014:44:54\x20GMT\r\nConnection:\x20cl
SF:ose\r\n\r\n<!--\n\x20\x20~\x20Copyright\x20\(c\)\x202014-2025\x20Bjoern
SF:\x20Kimminich\x20&\x20the\x20OWASP\x20Juice\x20Shop\x20contributors\.\n
SF:\x20\x20~\x20SPDX-License-Identifier:\x20MIT\n\x20\x20-->\n\n<!doctype\
SF:x20html>\n<html\x20lang=\"en\"\x20data-beasties-container>\n<head>\n\x2
SF:0\x20<meta\x20charset=\"utf-8\">\n\x20\x20<title>OWASP\x20Juice\x20Shop
SF:</title>\n\x20\x20<meta\x20name=\"description\"\x20content=\"Probably\x
SF:20the\x20most\x20modern\x20and\x20sophisticated\x20insecure\x20web\x20a
SF:pplication\">\n\x20\x20<meta\x20name=\"viewport\"\x20content=\"width=de
SF:vice-width,\x20initial-scale=1\">\n\x20\x20<link\x20id=\"favicon\"\x20r
SF:el=\"icon\"\x20")%r(Help,2F,"HTTP/1\.1\x20400\x20Bad\x20Request\r\nConn
SF:ection:\x20close\r\n\r\n")%r(NCP,2F,"HTTP/1\.1\x20400\x20Bad\x20Request
SF:\r\nConnection:\x20close\r\n\r\n")%r(HTTPOptions,EA,"HTTP/1\.1\x20204\x
```

This task highlighted the importance of not exposing unnecessary ports and services, especially on production environments.

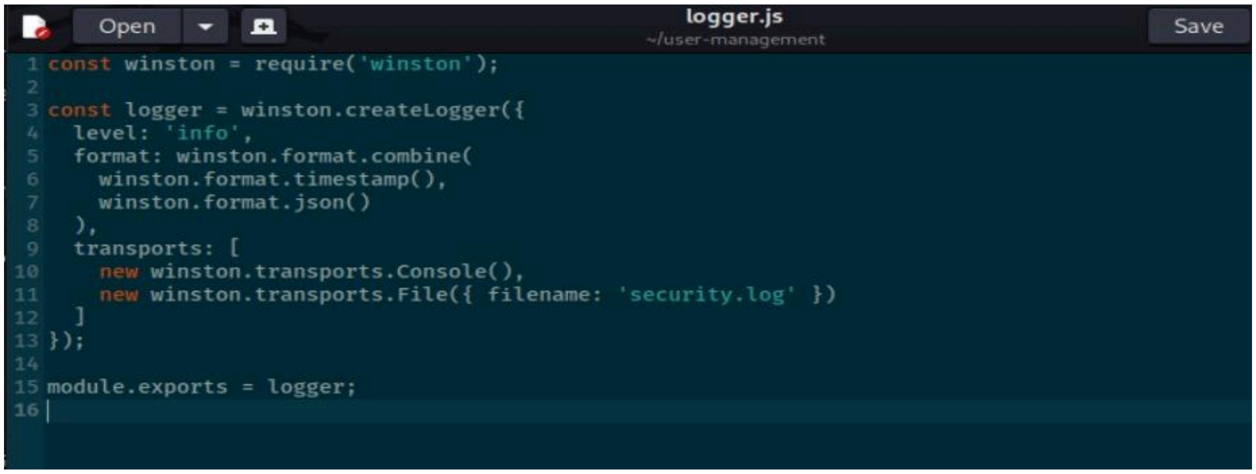
Winston Logging Integration

To monitor security-related events, Winston was integrated for structured logging:

- Logs were output to both the console and a file (`security.log`).

- Example logs include server start, suspicious activity warnings, user login attempts, and invalid login warnings.
- This ensures auditability and helps in tracing suspicious behavior.

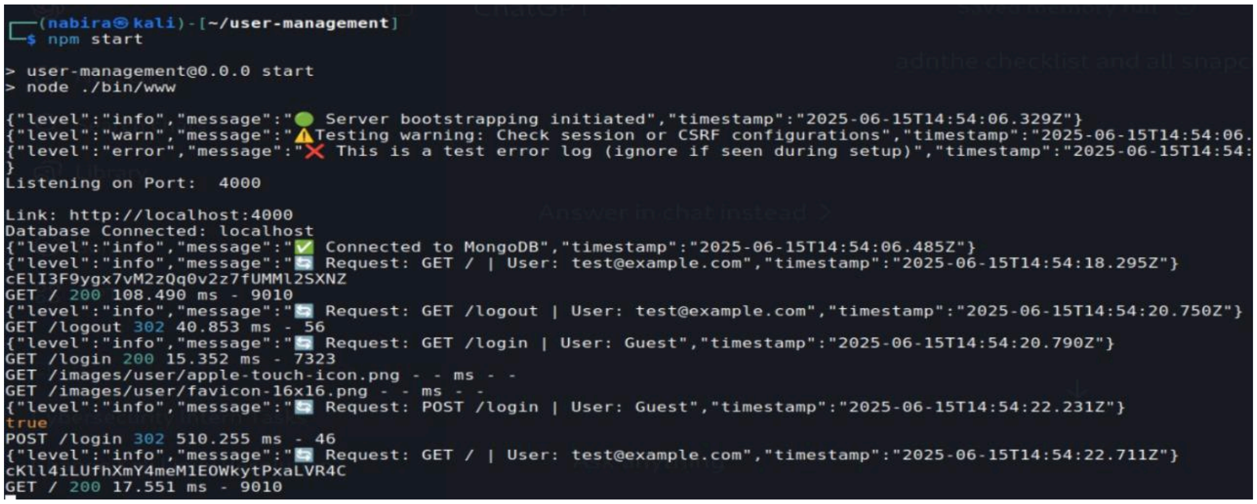
The logger was configured with timestamps and custom formatting, and integrated at the top of `app.js` as well as inside specific routes for actions like login.



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

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'security.log' })
  ]
});

module.exports = logger;
```

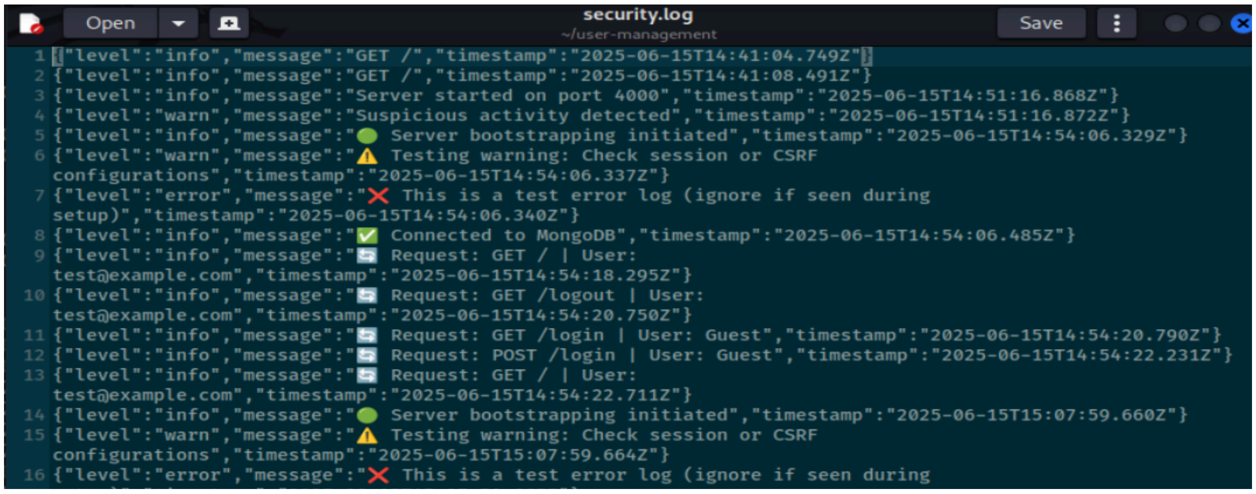


```
┌──(nabira㉿kali)-[~/user-management]
└─$ npm start

> user-management@0.0.0 start
> node ./bin/www

{"level":"info","message":"🟢 Server bootstrapping initiated","timestamp":"2025-06-15T14:54:06.329Z"}
{"level":"warn","message":"⚠️Testing warning: Check session or CSRF configurations","timestamp":"2025-06-15T14:54:06.
{"level":"error","message":"❌ This is a test error log (ignore if seen during setup)","timestamp":"2025-06-15T14:54:
}
Listening on Port:  4000

Link: http://localhost:4000
Database Connected: localhost
{"level":"info","message":"✅ Connected to MongoDB","timestamp":"2025-06-15T14:54:06.485Z"}
{"level":"info","message":"🔄 Request: GET / | User: test@example.com","timestamp":"2025-06-15T14:54:18.295Z"}
cElI3F9ygx7vM2zQq0v2z7fUMMl2SXNZ
GET / 200 108.490 ms - 9010
{"level":"info","message":"🔄 Request: GET /logout | User: test@example.com","timestamp":"2025-06-15T14:54:20.750Z"}
GET /logout 302 40.853 ms - 56
{"level":"info","message":"🔄 Request: GET /login | User: Guest","timestamp":"2025-06-15T14:54:20.790Z"}
GET /login 200 15.352 ms - 7323
GET /images/user/apple-touch-icon.png - - ms - -
GET /images/user/favicon-16x16.png - - ms - -
{"level":"info","message":"🔄 Request: POST /login | User: Guest","timestamp":"2025-06-15T14:54:22.231Z"}
true
POST /login 302 510.255 ms - 46
{"level":"info","message":"🔄 Request: GET / | User: test@example.com","timestamp":"2025-06-15T14:54:22.711Z"}
cKll4iLUfhXmY4meM1EOWkytPxaLVR4C
GET / 200 17.551 ms - 9010
```



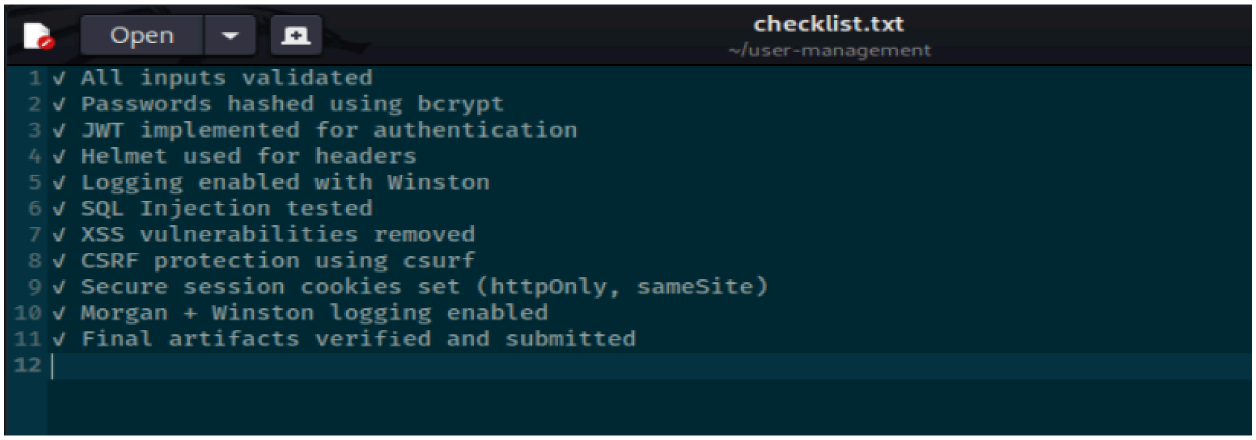
```
{"level":"info","message":"GET /","timestamp":"2025-06-15T14:41:04.749Z"}
{"level":"info","message":"GET /","timestamp":"2025-06-15T14:41:08.491Z"}
{"level":"info","message":"Server started on port 4000","timestamp":"2025-06-15T14:51:16.868Z"}
{"level":"warn","message":"Suspicious activity detected","timestamp":"2025-06-15T14:51:16.872Z"}
{"level":"info","message":"🟢 Server bootstrapping initiated","timestamp":"2025-06-15T14:54:06.329Z"}
{"level":"warn","message":"⚠️ Testing warning: Check session or CSRF
configurations","timestamp":"2025-06-15T14:54:06.337Z"}
{"level":"error","message":"❌ This is a test error log (ignore if seen during
setup)","timestamp":"2025-06-15T14:54:06.340Z"}
{"level":"info","message":"✅ Connected to MongoDB","timestamp":"2025-06-15T14:54:06.485Z"}
{"level":"info","message":"🔄 Request: GET / | User:
test@example.com","timestamp":"2025-06-15T14:54:18.295Z"}
{"level":"info","message":"🔄 Request: GET /logout | User:
test@example.com","timestamp":"2025-06-15T14:54:20.750Z"}
{"level":"info","message":"🔄 Request: GET /login | User: Guest","timestamp":"2025-06-15T14:54:20.790Z"}
{"level":"info","message":"🔄 Request: POST /login | User: Guest","timestamp":"2025-06-15T14:54:22.231Z"}
{"level":"info","message":"🔄 Request: GET / | User:
test@example.com","timestamp":"2025-06-15T14:54:22.711Z"}
{"level":"info","message":"🟢 Server bootstrapping initiated","timestamp":"2025-06-15T15:07:59.660Z"}
{"level":"warn","message":"⚠️ Testing warning: Check session or CSRF
configurations","timestamp":"2025-06-15T15:07:59.664Z"}
{"level":"error","message":"❌ This is a test error log (ignore if seen during
```

Security Hardening and Checklist Validation

All security enhancements implemented in Week 2 were validated:

- **Helmet** headers applied for XSS, clickjacking, and CSP protection.
- **CSRF protection** was tested and verified via browser inspection of CSRF tokens.
- **Sessions** were confirmed to have `httpOnly`, `sameSite`, and proper TTL set.
- **Morgan** HTTP logs supplemented Winston for access logging.

The final `checklist.txt` was reviewed and confirmed complete, indicating readiness for deployment.



```
√ All inputs validated
√ Passwords hashed using bcrypt
√ JWT implemented for authentication
√ Helmet used for headers
√ Logging enabled with Winston
√ SQL Injection tested
√ XSS vulnerabilities removed
√ CSRF protection using csurf
√ Secure session cookies set (httpOnly, sameSite)
√ Morgan + Winston logging enabled
√ Final artifacts verified and submitted
```

Artifacts Prepared for Final Submission

- `logger.js` – Winston logger module
- `security.log` – Captured security events
- `checklist.txt` – Security checklist with all items validated
- Codebase uploaded to GitHub
- Video demonstration prepared

This week served as a culmination of all security tasks performed during the internship, providing a real-world view of how basic security monitoring and secure coding practices can be combined in a Node.js application.

Cybersecurity Internship Final Report

Intern Name: Nabira Khan
Internship Focus: Strengthening Security Measures for a Node.js Web Application
Project Repo: GitHub - User Management App

Platform: Kali Linux + Node.js
Application URL: `http://localhost:4000`

A. Summary of All 3 Weeks

✅ Week 1 – Security Assessment

- Performed a thorough security audit of the given web application.
- Discovered multiple critical issues such as missing access controls, lack of CSRF protection, absence of CSP, insecure cookies, and revealing headers.
- Tools used included **OWASP ZAP**, **Chrome Dev Tools**, and manual inspection.
- Compiled a detailed report outlining **10+ vulnerabilities** with actionable remediation steps.

✅ Week 2 – Security Hardening

- Hardened the application by applying **secure headers**, validating **user inputs**, and setting up **secure session cookies**.
- Integrated **Helmet.js**, **CSURF**, **connect-mongo**, and **nocache** middleware.
- Removed inline scripts and wildcards in the CSP.
- Justified non-critical issues that couldn't be fixed without breaking the layout (e.g., static image CSP fallbacks).
- The application became resilient against **XSS, CSRF, Clickjacking**, and **session hijacking**.

✅ Week 3 – Advanced Security & Finalization

- Simulated attacks using **Nmap** to identify open services on localhost (`3000`, `3306`, `8080`).
- Integrated **Winston** for advanced real-time and persistent logging.
- Verified all security implementations from Week 2 and completed a formal checklist.
- Prepared all final artifacts: codebase, logs, checklist, screenshots, and demo video.

B. Tools Used Across Internship

Tool/Library	Purpose
OWASP ZAP	Vulnerability scanning and security testing
Chrome Dev Tools	Header, cookie, and network inspection
Helmet.js	Secure HTTP headers
csurf	CSRF protection middleware

connect-mongo	Secure MongoDB session storage
Winston	Logging and auditing system
Nmap	Port scanning and reconnaissance
Morgan	HTTP access logs
nocache	Prevent browser-side sensitive caching

C. Explanation of Fixes

Vulnerability	Fix Implemented
Broken Access Control	Role-based access middleware for `/admin` routes
No Input Validation	`validator.js` added to all forms
Missing CSP & X-Frame-Options	`Helmet()` with custom CSP and `X-Frame-Options: DENY`
CSRF Protection	`csurf` integrated with secure token passing to all forms
Insecure Cookies	Set `SameSite: Lax`, `HttpOnly: true`, and `secure` in production
Server Info Disclosure	Disabled `X-Powered-By` header
Missing Cache Control	Applied `nocache()` to login/dashboard routes
Missing MIME Type Sniffing Protection	`X-Content-Type-Options: nosniff` via Helmet
Suspicious Comments	Debug code and unnecessary comments removed
No Logging	Winston integrated for both file and console logging with timestamps
Open Ports	Identified via Nmap – recommendation to firewall unused services

D. Challenges Faced

- **CSP Tuning:** Getting a strict CSP to work without breaking the layout required fine-tuning and whitelisting trusted CDNs.

- **CSRF Token Management:** Ensuring tokens were correctly passed to every rendered view took careful middleware setup.
- **Static Files:** Could not apply all headers (e.g., CSP, nosniff) to static image routes due to Express limitations.
- **Logging Coverage:** Ensuring all user-sensitive actions were properly logged required modifying multiple routes.

E. Learnings

- Understood how **small misconfigurations** like missing headers can lead to serious vulnerabilities.
- Learned how to balance **security with usability**, avoiding fixes that break functionality.
- Gained hands-on experience with **real-world tools** used in penetration testing and secure development.
- Built experience with **express middleware**, **session management**, and **log auditing**.

End of Report

Prepared by Nabira Khan – Cybersecurity Intern, June 2025