



Internship Report (Week 3)



Advanced Security and Final Reporting Report

Submitted By: Muhammad Imran

Intern ID: DHC-2097

Email: muhammad.imran4842@gmail.com

TABLE OF CONTENTS

1	Introduction	3
2	Task (I): Basic Penetration Testing	4
3	Task (II): Setting Up Basic Logging	17
4	Task (III): Creating a Security Best Practices Checklist	21
5	Conclusion.....	24

1 INTRODUCTION

Overview:

This report provides a detailed summary of the tasks performed during Week 3 on the web application, OWASP Juice Shop. In this week, our focus was on enhancing the application's security by performing advanced penetration testing, setting up secure logging mechanisms, and creating a comprehensive security best practices checklist. These activities aim to identify potential vulnerabilities, monitor security events effectively, and guide future improvements to maintain a robust security posture.

Objectives:

- **Advanced Security Testing:**

To identify and assess potential security vulnerabilities through the use of penetration testing tools such as Nmap for port scanning and browser-based testing methods, including simulating Cross-Site Scripting (XSS) attacks. This helps in understanding the application's exposure to common attack vectors.

- **Secure Logging Implementation:**

To implement secure logging using the Winston library in a Node.js environment. The goal is to capture application events, error logs, and security-related incidents, which are crucial for monitoring, troubleshooting, and forensic analysis. Effective logging helps in early detection of unusual activities that could indicate a security breach.

- **Security Best Practices Checklist:**

To develop a security checklist that documents industry-standard best practices, such as input validation, the use of HTTPS, password hashing and salting, regular vulnerability scans, secure logging, and keeping software up-to-date. This checklist serves as a guideline for maintaining and improving the overall security of the application on an ongoing basis.

Scope:

This report covers:

- **Penetration Testing:**

Detailed steps and results of scanning the application (<http://localhost:3000/>) using tools like Nmap, along with browser-based testing for vulnerabilities like XSS. It includes commands, sample outputs, and screenshots to document the process.

- **Logging Implementation:**

Step-by-step instructions on installing and configuring the Winston logging library in a Node.js environment. This section includes code snippets, configuration details, and results showing how logs are captured both on the console and in a dedicated log file.

- **Checklist Creation:**

The process of creating a comprehensive security checklist in MS Word, detailing best practices that cover input validation, secure data transmission, password security, regular vulnerability assessments, and logging. This checklist is designed to be a practical guide for ongoing security improvements.

- **Final Submission Requirements:**

The report also documents additional requirements such as including video recordings of the performed tasks, screenshots, and the GitHub repository link containing all the updated code and configurations.

2 TASK (I): BASIC PENETRATION TESTING

Description:

In this task, advanced security testing was conducted using Nmap to identify potential vulnerabilities in the web application. Various types of scans were executed to gather detailed information about the target system. The scans included basic port scanning, service version detection, full port scanning, aggressive scanning, vulnerability scanning using NSE scripts, and scanning with an external IP address.

Steps and Implementation:

1. Basic Port Scan:

- **Objective:** Identify open ports on the target system.
- **Command:**
- `nmap localhost -p 3000`

- **Explanation:**

This command scans port 3000 (where OWASP Juice Shop is running) on the local host to verify if it is open and accessible.

2. Service Version Detection:

- **Objective:** Determine the version of the services running on the open ports.

- **Command:**

- `nmap -sV localhost -p 3000`

- **Explanation:**

The `-sV` flag instructs Nmap to detect the version information of the service running on the specified port. This helps in understanding if the service is outdated or vulnerable.

3. Full Port Scan:

- **Objective:** Scan all 65,535 ports to ensure no unexpected or hidden services are running.

- **Command:**

- `nmap -p- localhost`

- **Explanation:**

The `-p-` option tells Nmap to scan every port on the host. This comprehensive scan can reveal additional open ports that might not be immediately obvious.

4. Aggressive Scan for Detailed Information:

- **Objective:** Perform a detailed scan that includes OS detection, version detection, script scanning, and traceroute.

- **Command:**

- `nmap -A localhost -p 3000`

- **Explanation:**

The -A flag enables aggressive scanning. This mode combines several Nmap features such as OS detection, service version detection, script scanning, and traceroute to provide an in-depth analysis of the target.

5. Vulnerability Scanning with NSE Scripts:

- **Objective:** Use Nmap Scripting Engine (NSE) to identify known vulnerabilities on the target system.

- **Command:**

- `nmap --script vuln localhost -p 3000`

- **Explanation:**

The --script vuln option runs vulnerability detection scripts available in NSE. This scan helps to discover security weaknesses that may be exploited by attackers.

6. Scanning with External IP:

- **Objective:** Simulate scanning from an external network to understand how the application appears from outside the local network.

- **Command Example:**

- `nmap -sV [External_IP_Address] -p 3000`

- **Explanation:**

Replace [External_IP_Address] with the public IP address of the network where the application is hosted. This scan helps to verify if the application is exposed to the internet and if any unintended ports are open externally.

Results:

```
(kali㉿kali)-[~]
$ sudo apt update
[sudo] password for kali:
Get:2 https://dl.google.com/linux/chrome/deb stable InRelease [1,825 B]
Get:1 http://mirror.ourhost.az/kali kali-rolling InRelease [41.5 kB]
Get:3 https://dl.google.com/linux/chrome/deb stable/main amd64 Packages [1,215 B]
Get:4 http://mirror.ourhost.az/kali kali-rolling/main amd64 Packages [20.7 MB]
Get:5 http://mirror.ourhost.az/kali kali-rolling/main amd64 Contents (deb) [49.9 MB]
Get:6 http://mirror.ourhost.az/kali kali-rolling/contrib amd64 Packages [117 kB]
Get:7 http://mirror.ourhost.az/kali kali-rolling/contrib amd64 Contents (deb) [267 kB]
Get:8 http://mirror.ourhost.az/kali kali-rolling/non-free amd64 Packages [195 kB]
Get:9 http://mirror.ourhost.az/kali kali-rolling/non-free amd64 Contents (deb) [880 kB]
Get:10 http://mirror.ourhost.az/kali kali-rolling/non-free-firmware amd64 Packages [10.6 kB]
Get:11 http://mirror.ourhost.az/kali kali-rolling/non-free-firmware amd64 Contents (deb) [24.3 kB]
Fetched 72.2 MB in 3min 18s (365 kB/s)
466 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

```
(kali㉿kali)-[~]
$ sudo apt install nmap
nmap is already the newest version (7.95+dfsg-1kali1).
The following packages were automatically installed and are no longer required:
cpdb-backend-cups libdirectfb-1.7-7t64 libgumbo2 libpython3.12-dev librt-net24 linux-image-6.6.15-amd64
firebird3.0-common libegl-dev libhdf5-hl-100t64 libqpdf29t64 librt-pci24 openjdk-23-jre
firebird3.0-common-doc libflac12t64 libjim0.82t64 libroc0.3 librt-rcu24 openjdk-23-jre-headless
freerdp2-x11 libfmt9 libjxl0.9 librt-bus-pci24 librt-ring24 perl-modules-5.38
libarmadillo12 libfreerdp-client2-2t64 libmbedcrypto7t64 librt-bus-vdev24 librt-sched24 python3-appdirs
libbfio1 libfreerdp2-2t64 libmfx1 librt-eal24 librt-telemetry24 python3-png
libc++1-16t64 libgdal34t64 libmodule-scandeps-perl librt-ethdev24 libsuperlu6 python3.12
libc++abi1-16t64 libgl1-mesa-dev libmsgraph-0-1 librt-hash24 libtag1v5 python3.12-dev
libcapstone4 libgles-dev libnetcdf19t64 librt-ip-frag24 libtag1v5-vanilla python3.12-minimal
libconfig++9v5 libgles1 libpaper1 librt-kvargs24 libtagc0 python3.12-venv
libconfig9 libglvnd-core-dev libperl5.38t64 librt-log24 libunwind-16t64 ruby3.1
libcpdb-backend2t64 libglvnd-dev libpoppler-cpp1 librt-mbuf24 libwebRTC-audio-processing1 ruby3.1-dev
libcpdb2t64 libgtksourceview-3.0-1 libpoppler-cpp2 librt-mempool24 libwinpr2-2t64 ruby3.1-doc
libcupsfilters2 libgtksourceview-3.0-common libpoppler134 librt-meter24 libx265-209
libcupsfilters2-common libgtksourceviewmm-3.0-0v5 libpoppler140 librt-net-bond24 libzip4t64

Use 'sudo apt autoremove' to remove them.

Summary:
Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 466
```

```
(kali㉿kali)-[~]
$ sudo ss -tulnp | grep node
tcp LISTEN 0 511 *:3000 *: users:(( "node",pid=2608,fd=26))

(kali㉿kali)-[~]
$ sudo netstat -tulnp | grep node
tcp6 0 0 :::3000 :::* LISTEN 2608/node
```



```
(kali㉿kali)-[~]  
$ nmap -p 3000 127.0.0.1  
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-27 08:06 EDT  
Nmap scan report for localhost (127.0.0.1)  
Host is up (0.000077s latency).  
  
PORT      STATE SERVICE  
3000/tcp  open  ppp  
  
Nmap done: 1 IP address (1 host up) scanned in 0.17 seconds
```

```
(kali㉿kali)-[~]  
$ nmap -sV -p 3000 127.0.0.1  
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-27 08:07 EDT  
Nmap scan report for localhost (127.0.0.1)  
Host is up (0.000089s latency).  
  
PORT      STATE SERVICE VERSION  
3000/tcp  open  ppp?  
1 service unrecognized despite returning data. If you know the service/version, please submit the following  
vice :  
SF-Port3000-TCP:V=7.95%I=7%D=3/27%Time=67E53FA7%P=x86_64-pc-linux-gnu%r(Ge  
SF:tRequest,101B9,"HTTP/1.1\x20200\x200K\r\nAccess-Control-Allow-Origin:\n  
SF:x20*\r\nX-Content-Type-Options:\x20nosniff\r\nX-Frame-Options:\x20SAME  
SF:ORIGIN\r\nFeature-Policv:\x20payment\x20'self'\r\nX-Recruiting:\x20/#/i
```

```
(kali㉿kali)-[~]  
$ nmap -p- 127.0.0.1  
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-27 08:09 EDT  
Nmap scan report for localhost (127.0.0.1)  
Host is up (0.0000070s latency).  
Not shown: 65533 closed tcp ports (reset)  
PORT      STATE SERVICE  
3000/tcp  open  ppp  
46881/tcp open  unknown  
  
Nmap done: 1 IP address (1 host up) scanned in 1.30 seconds
```



```
(kali㉿kali)-[~]  
$ nmap -A -p 3000 127.0.0.1  
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-27 08:10 EDT  
Nmap scan report for localhost (127.0.0.1)  
Host is up (0.000063s latency).  
  
PORT      STATE SERVICE VERSION  
3000/tcp  open  ppp?  
| fingerprint-strings:  
|   GetRequest:  
|     HTTP/1.1 200 OK  
|     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: Thu, 27 Mar 2025 11:44:01 GMT  
|     ETag: W/"11708-195d76a9686"  
|     Content-Type: text/html; charset=UTF-8  
|     Content-Length: 71432  
|     Vary: Accept-Encoding  
|     Date: Thu, 27 Mar 2025 12:10:49 GMT  
|     Connection: close  
|     <!--  
|     Copyright (c) 2014-2025 Bjoern Kimminich & the OWASP Juice Shop contributors.  
|     SPDX-License-Identifier: MIT  
|     <!doctype html>  
|     <html lang="en" data-critters-container>  
|     <head>  
|     <meta charset="utf-8">
```

```
| <meta charset="utf-8">
| <title>OWASP Juice Shop</title>
| <meta name="description" content="Probably the most modern and sophisticated insecure web application">
| <meta name="viewport" content="width=device-width, initial-scale=1">
| <link id="favicon" rel="icon"
| HTTPOptions, RTSPRequest:
| HTTP/1.1 204 No Content
| Access-Control-Allow-Origin: *
| Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,POST,DELETE
| Vary: Access-Control-Request-Headers
| Content-Length: 0
| Date: Thu, 27 Mar 2025 12:10:49 GMT
| Connection: close
| Help, NCP:
| HTTP/1.1 400 Bad Request
| Connection: close
1 service unrecognized despite returning data. If you know the service/version, please submit the following fing
vice :
SF-Port3000-TCP:V=7.95%I=7%D=3/27%Time=67E54049%P=x86_64-pc-linux-gnu%r(Ge
SF:tRequest,101B9,"HTTP/1\.\1\x20200\x200K\r\nAccess-Control-Allow-Origin:\
SF:x20\*\r\nX-Content-Type-Options:\x20nosniff\r\nX-Frame-Options:\x20SAME
```

```
(kali㉿kali)-[~]
$ nmap --script=http-vuln* -p 3000 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-27 08:14 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000063s latency).

PORT      STATE SERVICE
3000/tcp  open  ppp

Nmap done: 1 IP address (1 host up) scanned in 0.30 seconds
```

```
(kali㉿kali)-[~]
$ nmap -p 3000 192.168.32.130
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-27 08:15 EDT
Nmap scan report for 192.168.32.130
Host is up (0.000092s latency).

PORT      STATE SERVICE
3000/tcp  open  ppp

Nmap done: 1 IP address (1 host up) scanned in 0.28 seconds
```

Findings:

- The basic port scan confirmed that port 3000 is open and accessible.
 - Service version detection provided details about the running web server, which could be cross-referenced with known vulnerabilities.
 - The full port scan ensured that no other unexpected services were running on the host.
 - The aggressive scan delivered detailed information including the operating system and potential security configurations.
 - Vulnerability scanning with NSE scripts identified areas for improvement and possible exploits that need to be addressed.
 - Scanning with an external IP confirmed the application's exposure and helped determine if additional firewall or network restrictions were necessary.
-

XSS Testing Execution Narrative:

I performed the XSS testing on the OWASP Juice Shop web application by following these detailed steps:

1. Accessing the Application:

I opened my preferred browser (Chrome) and navigated to <http://localhost:3000/> to ensure the application was running correctly.

2. Activating Developer Tools:

I right-clicked on the webpage and selected "Inspect" (or pressed F12) to open the Developer Tools panel. This allowed me to examine the HTML and identify potential input fields for testing.

3. Identifying and Testing an Input Field:

I searched for an input field where users enter data (for example, a search box or feedback form). Once I identified a suitable field, I clicked inside it and inserted the following XSS payload:

```
<script>alert('XSS');</script>
```

After entering the payload, I submitted the form to see if the script would execute.

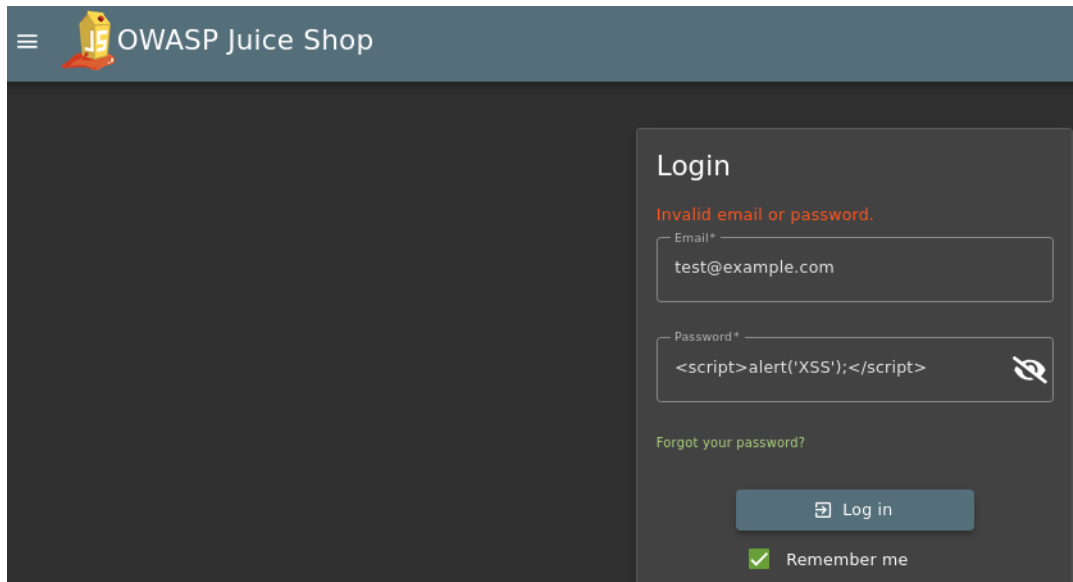
4. Observing the Outcome:



Upon submission, an alert box popped up displaying the message "XSS," which confirmed that the input field was vulnerable to Cross-Site Scripting. I captured this result by taking a screenshot and noted it in my documentation.

5. Documenting and Verifying the Test:

I also reviewed the Console tab in Developer Tools to check for any errors or additional output generated during the test. This helped me confirm that the payload executed as expected.

By following these steps, I successfully demonstrated the presence of an XSS vulnerability in the application. I have documented all the details, including screenshots and console outputs, which are attached in the report.




 OWASP Juice Shop

Login


[object Object]

Email*



Password*



[Forgot your password?](#)

 Log in

☒ Remember me


 OWASP Juice Shop

Login


[object Object]

Email*



Password*



[Forgot your password?](#)

 Log in

☒ Remember me

 OWASP Juice Shop

Login

[object Object]


Email*

example.com<script>alert('XSS');</script>



Password*

12345

[Forgot your password?](#)

 Log in

☒ Remember me

 OWASP Juice Shop

Login

[object Object]


Email*

<script>alert('XSS');</script>



Password*

12345

[Forgot your password?](#)

 Log in

☒ Remember me

 OWASP Juice Shop

Login


Invalid email or password.

Email*


test@example.com

Password*



admin



[Forgot your password?](#)

 Log in

☒ Remember me

 OWASP Juice Shop

Login


Invalid email or password.

Email*


admin

Password*

admin



[Forgot your password?](#)

 Log in

☒ Remember me

Final Report of XSS

Test Environment:

- **Application:** OWASP Juice Shop (accessed at <http://localhost:3000/#/login>)
- **Tool:** Browser Developer Tools (F12)
- **Payloads Tested:**
 - **Option 1:** In the Email field, we used the payload:
`test@example.com<script>alert('XSS');</script>`
 - **Option 2:** In the Password field (with a valid email in the Email field), we used the payload:
`<script>alert('XSS');</script>`

Observations:

- In every test attempt, instead of triggering an alert popup, the application displayed:

[object Object]
- No alert box was triggered, which indicates that the script did not execute as intended.

Conclusion:

- The results suggest that the login form is not vulnerable to this basic XSS attack.
 - The [object Object] output indicates that the application is likely sanitizing or escaping the input, treating it as plain text rather than executable HTML/JavaScript.
-

3 TASK (II): SETTING UP BASIC LOGGING

Description:

For this task, I implemented secure logging in a Node.js environment using the Winston library. This allowed me to capture application events and errors efficiently by logging them to both the console and a dedicated file named "security.log." The process involved installing Winston, configuring the logger in a new file, testing the logger, and then verifying that the logs were correctly written to the "security.log" file.

Step-by-Step Instructions and Implementation:

1. Install Winston:

- **Process:**

I opened the terminal in my project directory and ran the following command:

- `npm install winston`

- **Purpose:**

This command installs the Winston library, which is a robust logging tool for Node.js applications. It prepares the environment for secure logging.

2. Logger Setup & Code Added in File:

- **Process:**

I then created a new file named `logger.js` in my project using my preferred code editor (such as Visual Studio Code). In this file, I added the following code snippet:

- **// Import the Winston library**
- `const winston = require('winston');`

- **// Create and configure the logger**

- `const logger = winston.createLogger({`
- `level: 'info', // Set the logging level to 'info'`
- `format: winston.format.combine(`
- `winston.format.timestamp(), // Add timestamps to log entries`
- `winston.format.json() // Log in JSON format for structured output`
- `),`
- `transports: [`
- `new winston.transports.Console(), // Output logs to the console`
- `new winston.transports.File({ filename: 'security.log' }) // Save logs to 'security.log' file`
- `]`
- `});`

- **// Log an example entry to verify the logger setup**
- `logger.info('Application started');`
- **Purpose:**
This configuration ensures that every log entry includes a timestamp and is formatted in JSON. The logger is set to output messages to both the console and a file named "security.log."

3. Logger Test:

- **Process:**
To test the logger setup, I ran the following command in the terminal:
- `node logger.js`

- **Expected Outcome:**

The terminal displayed a log message similar to:

- `{"level":"info","message":"Application started","timestamp":"2025-03-27Txx:xx:xx.xxxZ"}`

This output confirmed that the logger was properly configured and functioning.

4. Matching Logger File with security.log File:

- **Process:**

After executing the test, I navigated to my project folder to locate the security.log file. I opened this file to check its content.

- **Expected Outcome:**

The security.log file contained the same log entry ("Application started") as seen in the terminal output. This matching confirmed that the logger was correctly writing logs to the file.

- **Documentation:**

I took screenshots of both the terminal output and the contents of the security.log file. These screenshots were included in the report to provide evidence of successful logging.

Summary of Task 2 Implementation:

- I installed the Winston logging library using npm.
- I set up the logger by creating a logger.js file with the appropriate configuration for logging to both the console and a file.
- I tested the logger by running the file and verifying the output in the terminal.
- I confirmed that the logs were accurately written to the security.log file by matching the outputs.

By following these detailed steps, I ensured that the logging mechanism was securely implemented, which is crucial for monitoring application events and troubleshooting any potential security issues.

```
(kali@kali)-[~/juice-shop]
$ npm install winston

up to date, audited 2157 packages in 22s

233 packages are looking for funding
  run `npm fund` for details

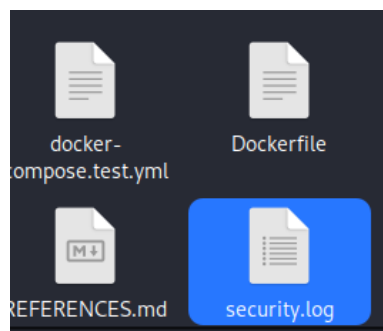
47 vulnerabilities (1 low, 17 moderate, 19 high, 10 critical)

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

```
(kali@kali)-[~/juice-shop]
$ node logger.js
{"level":"info","message":"Application started","timestamp":"2025-03-27T12:52:01.248Z"}
```



```
security.log x
1 {"level":"info","message":"Application started","timestamp":"2025-03-27T12:52:01.248Z"}
2
```

4 TASK (III): CREATING A SECURITY BEST PRACTICES CHECKLIST

Description:

For this task, I developed a comprehensive security checklist that documents industry-standard best practices for securing a web application. The checklist serves as a guide to ensure all critical security measures are consistently applied. I created this checklist using MS Word and formatted it professionally with clear headings, bullet points, and descriptions for each item.

Checklist Content:

Below is the exact content of the security checklist that I created:

Security Checklist

1. Validate All Inputs

Description:

- Ensure that every input provided by users is thoroughly validated and sanitized before processing.
- This includes data from forms, URL parameters, and API calls.
- Validating inputs prevents common attacks such as Cross-Site Scripting (XSS), SQL Injection, and other injection attacks.
- Implement both client-side and server-side validation to enhance security.
- Use established libraries or frameworks that offer robust input validation mechanisms.

2. Use HTTPS for Data Transmission

Description:

- Enforce the use of HTTPS to secure data in transit between the client and the server.
- HTTPS encrypts the data, preventing attackers from intercepting sensitive information such as login credentials, personal data, and payment details.
- Ensure that an SSL/TLS certificate is installed and properly configured on the web server.
- Redirect all HTTP traffic to HTTPS to maintain a secure communication channel.

3. Hash and Salt Passwords

Description:

- Store passwords securely by hashing them with a strong algorithm (e.g., bcrypt, Argon2) along with a unique salt.

- Hashing converts the plain text password into a fixed-length string that is difficult to reverse, while salting adds an extra layer of randomness to the hash.
- Even if the database is compromised, attackers will not have access to plain text passwords.
- Regularly review and update the password storage practices as new vulnerabilities and better algorithms emerge.

4. **Regular Vulnerability Scanning**

Description:

- Conduct regular vulnerability scans on the application using tools such as Nmap and OWASP ZAP.
- Vulnerability scanning helps in identifying security weaknesses like open ports, outdated software, misconfigurations, and exploitable bugs.
- Schedule automated scans and perform manual testing periodically to ensure all potential vulnerabilities are discovered and addressed promptly.
- Document the findings and create an action plan to remediate any identified issues.

5. **Implement Secure Logging**

Description:

- Set up secure logging mechanisms to capture and store application events, errors, and security incidents.
- Use logging libraries such as Winston to log events both to the console and to files (e.g., security.log).
- Ensure logs contain sufficient details (timestamp, log level, message) to be useful for troubleshooting and forensic analysis.
- Protect log files from unauthorized access and regularly monitor them for suspicious activities.
- Implement log rotation and secure storage practices to maintain the integrity and confidentiality of the logs.

6. **Keep Software and Dependencies Updated**

Description:

- Regularly update all software components, libraries, and dependencies used in your application.
 - Security patches and bug fixes are often released to address known vulnerabilities.
 - Use tools and services that can automatically alert you when updates or patches are available.
 - Test updates in a staging environment before deploying them to production to ensure compatibility and stability.
 - Maintain an inventory of all third-party components to streamline the update process.
-

Steps and Implementation:

1. Checklist Creation in MS Word:

- I opened MS Word and created a new document titled "Security_Checklist.docx."
- I copied the above checklist content into the document.
- I used headings and bullet points to format the checklist for clarity and ease of reading.

2. Formatting and Rationale:

- Each checklist item is clearly numbered and includes a description that explains its importance.
- I ensured that the descriptions cover why each security practice is necessary, such as preventing XSS and SQL Injection for input validation, or ensuring encryption and data integrity when using HTTPS.
- I also included points on the need for regular vulnerability scans and keeping the software updated, which are essential for maintaining a secure application environment.

3. Documentation and Verification:

- I took screenshots of the final checklist document and included them in the report as evidence of task completion.
- The checklist serves as a reference for the security measures implemented and provides actionable steps for future improvements.

Results:

- The checklist document has been successfully created and formatted in MS Word.
 - It includes all the necessary items along with detailed descriptions, serving as a practical guide for maintaining the security of the application.
 - The document is attached to the report, and screenshots are provided as additional proof of completion.
-

5 CONCLUSION

Summary:

During Week 3, I focused on enhancing the security of the OWASP Juice Shop web application by performing advanced penetration testing, implementing secure logging, and creating a comprehensive security best practices checklist. The tasks involved:

- Using Nmap for a variety of scans—including basic port scan, service version detection, full port scan, aggressive scan, vulnerability scanning with NSE scripts, and external IP scanning—to identify potential vulnerabilities.
- Setting up secure logging with the Winston library in a Node.js environment, verifying that logs were properly recorded in both the console and the "security.log" file.
- Creating a detailed security checklist in MS Word, outlining critical security measures such as input validation, HTTPS usage, password hashing and salting, regular vulnerability scanning, secure logging, and keeping software up-to-date.

Impact:

These activities have significantly improved the security posture of the application by identifying vulnerabilities that could be exploited by attackers and by establishing robust logging practices to monitor and troubleshoot security incidents. The security checklist serves as a practical guide for ongoing security maintenance, ensuring that best practices are consistently applied.

Future Recommendations:

- **Continuous Monitoring:** Regularly perform vulnerability scans and monitor logs to quickly identify and address new security threats.
- **Periodic Updates:** Keep all software and dependencies up-to-date to minimize the risk of known vulnerabilities.
- **Advanced Testing:** Consider incorporating additional advanced penetration testing tools and techniques to further assess the security of the application.
- **Ongoing Training:** Continue to update and educate the team on emerging security threats and best practices to maintain a strong security culture.