

# Security Assessment Report

**Project Title:**

*Web Application Vulnerability Assessment (Project 1)*

**Prepared By:**

**Shashanka U N**  
**Cybersecurity Intern**  
**Future interns**

**Prepared For:**

**Future interns**

## **Contents**

- 1. Introduction**
- 2. Background and need**
- 3. Scope and objective**
- 4. Methodology**
- 5. Tools and technology**
- 6. Project preparation**
- 7. Results and analysis**
- 8. Vulnerability report**
- 9. Conclusion**

## 1. Introduction

The OWASP Juice Shop is a purposely insecure web application maintained by the Open Web Application Security Project (OWASP) as a training ground for web security testing and awareness. It contains a wide range of vulnerabilities, from basic to highly complex, covering the entire OWASP Top 10 list and beyond.

This project involves conducting a penetration test on the Juice Shop instance to simulate how an attacker might discover and exploit weaknesses. The goal is not only to identify vulnerabilities but also to understand the techniques behind their discovery, the risks they pose, and the best practices for mitigating them. By following a structured testing process, the results will serve as both a security assessment and a learning exercise, helping to improve skills in vulnerability detection and secure coding.

## 2. Background and Need

With the increasing number of cyber threats targeting web applications, understanding and identifying security vulnerabilities has become essential for both developers and security professionals. OWASP Juice Shop is a deliberately vulnerable web application designed for practicing and improving penetration testing skills in a safe environment.

The primary need for this assessment is to simulate real-world attack scenarios, uncover weaknesses, and enhance knowledge of common vulnerabilities as described in the OWASP Top 10. By conducting this hands-on exercise, I aim to strengthen my skills in vulnerability identification, exploitation, and mitigation — skills that are crucial for securing modern applications.

## 3. Scope & Objectives

### Scope

This report focuses on testing the OWASP Juice Shop application in a controlled environment. The assessment will involve identifying, exploiting, and documenting various security vulnerabilities, following ethical hacking practices. The activities will be limited to:

- Web application security testing.
- OWASP Top 10 vulnerability categories.
- Hands-on exploitation in a safe lab environment.

### Objectives

The main objectives of this assessment are:

1. To gain practical experience in web application penetration testing.
2. To identify and exploit vulnerabilities within the OWASP Juice Shop platform.
3. To document findings with evidence and technical details.
4. To recommend mitigation strategies for discovered vulnerabilities.
5. To improve overall cybersecurity awareness and skill set.

## 4. Methodology

This assessment was conducted using a structured penetration testing methodology aligned with OWASP Testing Guide v4 and industry best practices. The process comprised the following phases:

### 1. Reconnaissance

- Collected intelligence on the application's architecture, frameworks, and hosting environment.
- Mapped application endpoints, features, and authentication flows.

### 2. Vulnerability Assessment

- Performed both automated and manual analysis to identify potential security weaknesses.
- Classified vulnerabilities against the OWASP Top 10 risk categories.

### 3. Exploitation

- Validated identified vulnerabilities through controlled exploitation in a safe, isolated environment.
- Captured technical evidence to support each finding.

### 4. Impact Analysis

- Assessed the business and technical impact of each successfully exploited vulnerability.
- Evaluated potential data exposure, privilege escalation, and application misuse scenarios.

### 5. Documentation & Reporting

- Consolidated findings with detailed descriptions, severity ratings, and proof-of-concept evidence.
- Developed actionable remediation recommendations prioritised by risk level.

## 5. Tools and Technologies

- **OWASP Zed Attack Proxy (ZAP):** An open-source web application security scanner.
- **Docker:** Containerization platform used to run the vulnerable web application locally.
- **Vulnerable Web Application:** OWASP Juice Shop, chosen for its comprehensive security challenges.

## 6. Project Setup

### 6.1 Setting Up Docker Environment and Pulling the Vulnerable Web Application

Docker, a containerization platform, was used to create an isolated environment for running the vulnerable web application. This approach ensures consistency and avoids interference with the host system.

- The official Docker Desktop application was installed on the host machine (Windows/Mac/Linux).
- To verify the installation, the following command was executed in the terminal:

*Bash: docker --version*

```
PS C:\Users\shash> docker --version
Docker version 28.3.2, build 578ccf6
PS C:\Users\shash> |
```

- The vulnerable web application chosen for this project was OWASP Juice Shop, a deliberately insecure app designed for security testing and learning. The official Docker image was pulled from Docker Hub using the command:

*Bash: `docker pull bkimminich / juice-shop`*

```
PS C:\Users\shash> docker pull bkimminich/juice-shop
Using default tag: latest
```

Login prior to pull:

Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com/> to create one.

You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using SSO. Learn more at <https://docs.docker.com/go/access-tokens/>

```
Username (shashanka04): shashanka04
Password:
```

```
latest: Pulling from bkimminich/juice-shop
Digest: sha256:89337a9b216106f276ecc9875800c4c9c47b9d89b8a785b1111d5c94a0fed5c2
Status: Image is up to date for bkimminich/juice-shop:latest
docker.io/bkimminich/juice-shop:latest
```

## 6.2 Running the Vulnerable Web Application Container

After successfully pulling the Juice Shop image, the application was launched as a Docker container to enable local access for scanning.

- The container was started with port mapping to expose the application on the host machine's port 3000:  
*Bash: `docker run -d -p 3000:3000 bkimminich/juice-shop`*
- This setup allowed the web application to be accessed via a web browser at the URL: <http://localhost:3000>
- Running the application within a Docker container provided a controlled, repeatable environment essential for reliable security testing.

```
PS C:\Users\shash> docker run --rm -p 3000:3000 bkimminich/juice-shop
info: Detected Node.js version v22.18.0 (OK)
info: Detected OS linux (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file index.html is present (OK)
info: Required file styles.css is present (OK)
info: Required file main.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file vendor.js is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
info: Restored 1-star directoryListingChallenge (Confidential Document)
info: Restored 1-star errorHandlerChallenge (Error Handling)
```

### 6.3 Configuring and Using OWASP ZAP for Scanning

- Launched OWASP Zed Attack Proxy (ZAP) with the target application running locally at `http://localhost:3000`.
- Set the target URL in ZAP to ensure the scan focused exclusively on the Juice Shop application.
- Verified connectivity between ZAP and the target to confirm requests were being routed correctly.
- Used ZAP's **Spider** tool to crawl the application, mapping all accessible pages, forms, and hidden resources.
- Reviewed the site map generated by the spider to ensure comprehensive coverage before starting the active scan.
- Initiated an **Active Scan** against the discovered endpoints to detect vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), missing security headers, and insecure cookies.
- Collected scan results, with alerts categorized by severity: **High**, **Medium**, **Low**, and **Informational**.
- Reviewed each alert, noting detailed evidence, potential impact, and recommended remediation steps.
- Manually validated high and medium severity findings to confirm accuracy and reduce false positives.

History

Search

Output

Spider

AJAX Spider

Active Scan

Alerts

New Scan

Progress: 0: http://localhost:3000

100%

Current Scans: 0

URLs Found: 113

Nodes Added: 72

Export

URLs

Added Nodes

Messages

Processed	Method	URI	
	GET	http://localhost:3000	Seed
	GET	http://localhost:3000/robots.txt	Seed
	GET	http://localhost:3000/sitemap.xml	Seed
	GET	http://localhost:3000/ftp	
	GET	http://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.css	Out of Scope
	GET	http://localhost:3000/assets/public/favicon_js.ico	
	GET	http://localhost:3000/styles.css	
	GET	http://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js	Out of Scope
	GET	http://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js	Out of Scope
	GET	http://localhost:3000/runtime.js	
	GET	http://localhost:3000/polyfills.js	
	GET	http://localhost:3000/vendor.js	
	GET	http://localhost:3000/main.js	
	GET	https://owasp.org/	Out of Scope
	GET	https://owasp-juice-shop/	Out of Scope
	GET	https://www.googleapis.com/oauth2/v1/userinfo?alt=json	Out of Scope
	GET	http://www.w3.org/1999/html	Out of Scope
	GET	https://github.com/juice-shop/juice-shop	Out of Scope
	GET	https://ponzico.win/ponzico.pdf	Out of Scope
	GET	https://www.sec.gov/investor/alerts/ia_virtualcurrencies.pdf	Out of Scope
	GET	http://fizzed.com/oss/font-mfizz	Out of Scope

Alerts

0

3

2

3

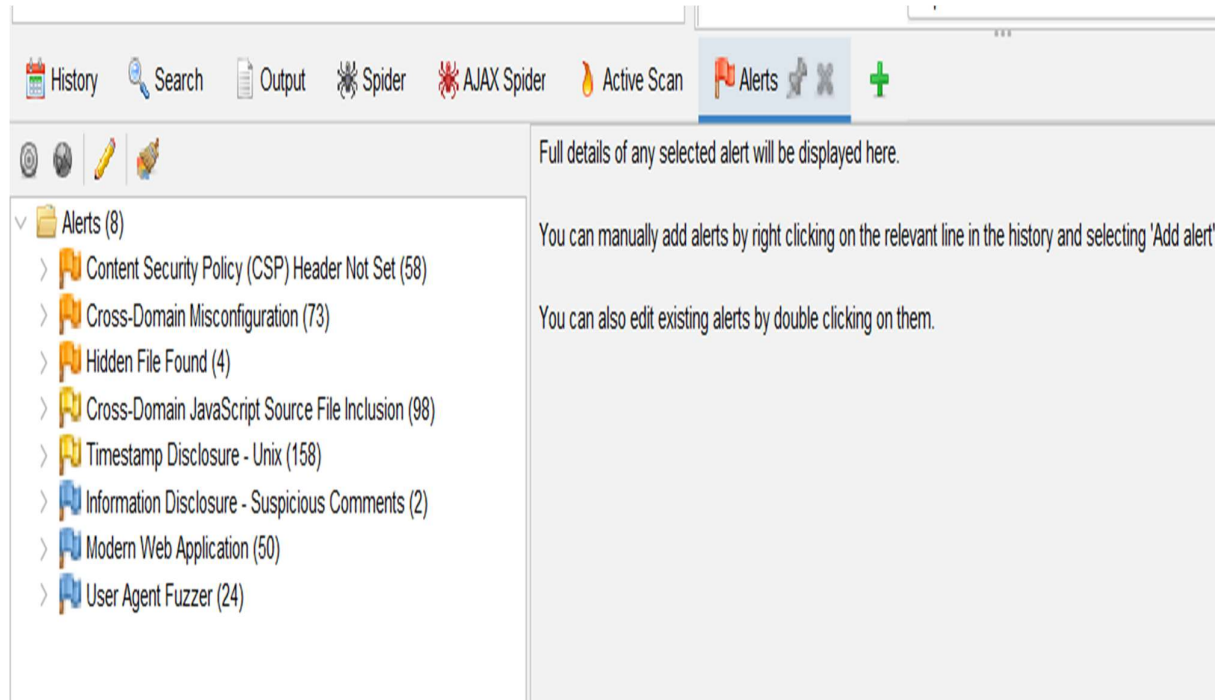
Main Proxy: localhost:8080

Current Status



## 7. Results & Analysis

Following the automated security assessment using OWASP ZAP, multiple alerts were identified. These alerts represent potential vulnerabilities that could be exploited by an attacker if left unaddressed. A detailed breakdown of each finding is provided in the subsequent sections. For complete technical evidence, refer to the ZAP-generated **Alerts** page, which contains full request/response data for each vulnerability.



## 8. Vulnerability Report

### 1. Content Security Policy (CSP) Header Not Set

**Risk:** Medium | **Confidence:** High

Description: The application does not set a Content-Security-Policy header. Without CSP, attackers may inject malicious scripts, leading to XSS or data theft.


Impact:

- Increased risk of Cross-Site Scripting (XSS)
- Possible data compromise and content manipulation

### Recommendation:

Configure the server to set a strict Content-Security-Policy header allowing only trusted sources for scripts, styles, and media.

**Content Security Policy (CSP) Header Not Set**

URL: http://localhost:3000/sitemap.xml  
Risk:  Medium  
Confidence: High  
Parameter:  
Attack:  
Evidence:  
CWE ID: 693  
WASC ID: 15  
Source: Passive (10038 - Content Security Policy (CSP) Header Not Set)  
Alert Reference: 10038-1  
Input Vector:

**Description:**  

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

**Solution:**  

Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

**Reference:**  
[https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing\\_Content\\_Security\\_Policy](https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy)  
[https://cheatsheetseries.owasp.org/cheatsheets/Content\\_Security\\_Policy\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html)  
<https://www.w3.org/TR/CSP/>

**Alert Tags:**

Key	Value
CWE-693	<a href="https://cwe.mitre.org/data/definitions/693.html">https://cwe.mitre.org/data/definitions/693.html</a>
OWASP_2021_A05	<a href="https://owasp.org/Top10/A05_2021-Security_Misconfiguration/">https://owasp.org/Top10/A05_2021-Security_Misconfiguration/</a>
OWASP_2017_A06	<a href="https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html">https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html</a>
POLICY_QA_STD	
POLICY_PENTEST	

## 2. Cross-Domain Misconfiguration

**Risk:** Medium | **Confidence:** High

### Description:

The application's cross-domain settings are misconfigured, potentially allowing unauthorized domains to access resources. This can expose sensitive data or enable malicious scripts to interact with the application.


### Impact:

- Unauthorized data access from external domains
- Increased risk of Cross-Site Scripting (XSS) and data theft



### Recommendation:

Restrict cross-domain access by configuring proper CORS policies and limiting access only to trusted domains.

**Cross-Domain Misconfiguration**  
URL: <http://localhost:3000/polyfills.js>  
Risk:  Medium  
Confidence: Medium  
Parameter:  
Attack:  
Evidence: Access-Control-Allow-Origin: \*  
CWE ID: 264  
WASC ID: 14  
Source: Passive (10098 - Cross-Domain Misconfiguration)  
Input Vector:  
Description:  
Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server.  
Other Info:  
The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.  
Solution:  
Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance).  
Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.  
Solution:  
Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance).  
Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.  
Reference:  
[https://vuln.catfortify.com/en/detail?id=desc.config.dotnet.html5\\_overly\\_permissive\\_cors\\_policy](https://vuln.catfortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy)  
Alert Tags:  

Key	Value
OWASP_2021_A01	<a href="https://owasp.org/Top10/A01_2021-Broken_Access_Control/">https://owasp.org/Top10/A01_2021-Broken_Access_Control/</a>
OWASP_2017_A05	<a href="https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html">https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html</a>
CWE-264	<a href="https://cwe.mitre.org/data/definitions/264.html">https://cwe.mitre.org/data/definitions/264.html</a>
POLICY_QA_STD	
POLICY_PENTEST	

## 3. Hidden File Found

**Risk:** Medium | **Confidence:** Low

### Description:


A sensitive file was found accessible on the server. It may contain administrative, configuration, or credential information that attackers could exploit for further attacks or social engineering.

### Impact:

- Disclosure of sensitive information
- Increased risk of targeted attacks

### Recommendation:

Remove unnecessary files from production. If needed, secure them with proper authentication and authorization, or restrict access to specific Ips.

<b>Hidden File Found</b>	
URL:	http://localhost:3000/.hg
Risk:	 Medium
Confidence:	Low
Parameter:	
Attack:	
Evidence:	HTTP/1.1 200 OK
CWE ID:	538
WASC ID:	13
Source:	Active (40035 - Hidden File Finder)
Input Vector:	
Description:	A sensitive file was identified as accessible or available. This may leak administrative, configuration, or credential information which can be leveraged by a malicious individual to further attack the system or conduct social engineering efforts.
Other Info:	
Solution:	Consider whether or not the component is actually required in production, if it isn't then disable it. If it is then ensure access to it requires appropriate authentication and authorization, or limit exposure to internal systems or specific source IPs, etc.
Reference:	<a href="https://blog.hboeck.de/archives/892-Introducing-Snallygaster-a-Tool-to-Scan-for-Secrets-on-Web-Servers.html">https://blog.hboeck.de/archives/892-Introducing-Snallygaster-a-Tool-to-Scan-for-Secrets-on-Web-Servers.html</a>

## 4. Cross-Domain JavaScript Source File Inclusion


**Risk:** Medium | **Confidence:** High

### Description:

The application includes JavaScript files from external domains. If these external sources are compromised, malicious scripts could be executed in the user's browser.

### Impact:

- Potential for injected malicious code
- Risk of data theft, session hijacking, or application compromise

<b>Cross-Domain JavaScript Source File Inclusion</b>	
URL:	http://localhost:3000/itemap.xml
Risk:	 Low
Confidence:	Medium
Parameter:	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack:	
Evidence:	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
CWE ID:	829
WASC ID:	15
Source:	Passive (10017 - Cross-Domain JavaScript Source File Inclusion)
Input Vector:	
Description:	The page includes one or more script files from a third-party domain.
Other Info:	
Solution:	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.

## 5. Timestamp Disclosure – Unix

**Risk:** Low | **Confidence:** Low

### Description:

The application/web server disclosed a timestamp (1650485437 → 2022-04-21 01:40:37). This may help attackers identify system or software changes, potentially aiding in targeted attacks.

### Impact:

- Reveals system information
- May assist attackers in finding patterns or vulnerabilities

### Recommendation:

Avoid exposing timestamp data unless required. If needed, ensure it does not reveal sensitive system or update information.

<b>Timestamp Disclosure - Unix</b>	
URL:	http://localhost:3000
Risk:	Low
Confidence:	Low
Parameter:	
Attack:	
Evidence:	1650485437
CWE ID:	497
WASC ID:	13
Source:	Passive (10096 - Timestamp Disclosure)
Input Vector:	
Description:	A timestamp was disclosed by the application/web server. - Unix
Other Info:	1650485437, which evaluates to: 2022-04-21 01:40:37.
Solution:	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Reference:	<a href="https://cwe.mitre.org/data/definitions/200.html">https://cwe.mitre.org/data/definitions/200.html</a>

## 6. Information Disclosure – Suspicious Comments

**Risk:** Informational | **Confidence:** Low

### Description:

The application contains comments in its code that may provide hints or useful information to an attacker.

### Impact:

- May expose sensitive implementation details
- Could assist in targeted attacks

### Recommendation:

Remove any code comments that reveal potentially sensitive information and address the underlying issues mentioned in them.

**Information Disclosure - Suspicious Comments**  
URL: http://localhost:3000/main.js  
Risk:  Informational  
Confidence: Low  
Parameter:  
Attack:  
Evidence: query  
CWE ID: 615  
WASC ID: 13  
Source: Passive (10027 - Information Disclosure - Suspicious Comments)  
Input Vector:  
Description:  
The response appears to contain suspicious comments which may help an attacker.  
Other Info:  
The following pattern was used: `\bQUERY\b` and was detected in likely comment: `"//owasp.org" target='_blank'>Open Worldwide Application Security Project (OWASP)</a>` and is developed and maintained by volunteer", see evidence field for the suspicious comment/snippet.  
Solution:  
Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.

## 7. Modern Web Application

**Risk:** Informational | **Confidence:** Medium

### Description:

The application appears to be a modern web application, likely using client-side JavaScript frameworks and AJAX. Standard crawling tools may not fully explore its functionality; an AJAX spider might be more effective.

### Impact:

- No direct security risk.
- May require specialized tools for complete automated testing.

### Recommendation:

No action is required. This is an informational finding only.

#### Modern Web Application

URL: http://localhost:3000

Risk:  Informational

Confidence: Medium

Parameter:

Attack:

Evidence: `<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>`

CWE ID:

WASC ID:

Source: Passive (10109 - Modern Web Application)

Input Vector:

Description:

The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.

Other Info:

No links have been found while there are scripts, which is an indication that this is a modern web application.

Solution:

This is an informational alert and so no changes are required.

## 8. UserAgentFuzzer

**Risk:** Informational | **Confidence:**Medium

### Description:

The application responds differently when the User-Agent header is changed (e.g., simulating mobile devices or search engine crawlers). This test compares the response status code and content hash with the original response.

### Impact:

- No direct security vulnerability.
- Could indicate different behavior or content delivery for specific clients.

### Recommendation:

Review and ensure that alternate responses do not expose sensitive information or unintended functionality.

User Agent Fuzzer
URL: http://localhost:3000/assets
Risk:  Informational
Confidence: Medium
Parameter: Header User-Agent
Attack: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Evidence:
CWE ID: 0
WASC ID: 0
Source: Active (10104 - User Agent Fuzzer)
Input Vector:
Description: Check for differences in response based on fuzzed User Agent (eg. mobile sites, access as a Search Engine Crawler). Compares the response statuscode and the hashcode of the response body with the original response.
Other Info:
Solution:
Reference: <a href="https://owasp.org/lwstg">https://owasp.org/lwstg</a>

## 9. Conclusion

The **OWASP ZAP** scan successfully identified 8 security alerts of varying severity levels in the target web application. These vulnerabilities include missing security headers, cookie security misconfigurations, and potential exposure to common web-based attacks such as Cross-Site Scripting (XSS) and Clickjacking. While some findings pose medium risk and can be exploited to compromise user data or application integrity, others are low-risk but still represent security best practice gaps.

Addressing these issues promptly by implementing recommended mitigations—such as enforcing strong security headers, securing cookies, and validating user inputs—will significantly reduce the attack surface and improve the overall security posture of the application.