



Security Assessment Report – OWASP Juice Shop



Project Title:

Vulnerability Assessment of OWASP Juice Shop Web Application



Internship Program:

Future Interns – Cybersecurity Internship

Conducted by Future Interns





Intern Name:

Aman Patel

Cybersecurity & Ethical Hacking Enthusiast



Tools & Technologies Used:

-  OWASP ZAP
-  Parrot OS (VM)
-  OWASP Juice Shop
-  Web Browser (Firefox)
-  LibreOffice Writer



Assessment Date:

09th June 2025



Test Environment:

- Host Machine: **Windows 11**
- Attacker Machine: **Parrot OS (VirtualBox VM)**
- Target: **OWASP Juice Shop (localhost or LAN IP)**



Executive Summary

This report presents a comprehensive **vulnerability assessment** conducted on the **OWASP Juice Shop**, a deliberately insecure web application used for educational and security testing purposes.

The objective of this assessment was to simulate real-world penetration testing techniques and identify common web application vulnerabilities in a safe and controlled environment.

The assessment was carried out using **Parrot OS** (in a virtual machine) and industry-standard tools such as **OWASP ZAP**, targeting a locally hosted instance of Juice Shop on the host Windows machine.

During the assessment, **five distinct web application vulnerabilities** were discovered and documented, including input validation issues, sensitive data exposure, and broken access controls. Each vulnerability was captured with technical evidence and accompanied by relevant screenshots.

This report provides:




- A summary of the identified vulnerabilities
- Their potential security impact
- Steps to reproduce them
- Suggested mitigation strategies

The findings demonstrate the importance of secure coding practices and regular security testing to safeguard web applications from exploitation. This hands-on project also reflects the intern's ability to apply **ethical hacking** techniques in real-world scenarios and document the process professionally.

Methodology

The assessment followed a **black-box testing approach**, simulating how an external attacker would discover and exploit vulnerabilities in a live web application without access to its source code.

1. Environment Setup

-  **Target Application:**
OWASP Juice Shop (Hosted on Windows using Node.js)
 -  **Attacker Machine:**
Parrot OS (Virtual Machine via VirtualBox)
 -  **Communication:**
Juice Shop was accessed over LAN using the host IP (e.g., `http://192.168.x.x:3000`)
-

2. Scanning & Enumeration

- Launched **OWASP ZAP** on Parrot OS
 - Performed a **Passive Scan** by proxying browser traffic through ZAP
 - Initiated **Active Scan** to discover deeper vulnerabilities automatically
-

3. Vulnerability Discovery

- Manually explored Juice Shop features (Login, Signup, Products, etc.)
 - Observed responses and alerts triggered in OWASP ZAP
 - Captured **five unique vulnerability types** (e.g., SQLITE, Cloud MetaData Exposed, etc)
-

4. Documentation & Evidence

- Took **screenshots** of each discovered issue
 - Documented:
 - The vulnerable endpoint or behavior
 - Technical explanation
 - Potential risks
 - Reproduction steps
 - Suggested fix
-

5. Tools & Utilities Used


Tool	Purpose
OWASP Juice Shop	Vulnerable test target
OWASP ZAP	Scanning & vulnerability detection
Parrot OS	Attacker machine environment
Web Browser	Manual testing and proxy setup
Screenshot Tool	Capturing visual evidence

This structured methodology ensured thorough and ethical testing aligned with OWASP standards — simulating how vulnerabilities are discovered in real-world applications.

Vulnerability #1: SQL Injection

 **Severity:** High

 **OWASP Category:** A1 - Injection

 **CWE ID:** **CWE-89** — *Improper Neutralization of Special Elements used in an SQL Command* ('SQL Injection')

 **Endpoint :**

`http://192.168.56.1:3000/rest/products/search?q='`

Description:

The application is vulnerable to **SQL Injection** due to insufficient input sanitization. When crafted input is passed to the `search` query parameter, the backend SQLite engine throws a syntax error, indicating that raw user input is being directly interpreted in SQL queries.




This confirms the presence of an injection point that attackers can exploit to manipulate queries, extract sensitive data, or gain unauthorized access.

ZAP Alert Message:





```
"message": "SQLITE_ERROR: near \"'\": syntax error",  
"error": "SQLITE_ERROR: near \"'\": syntax error"
```

Impact:

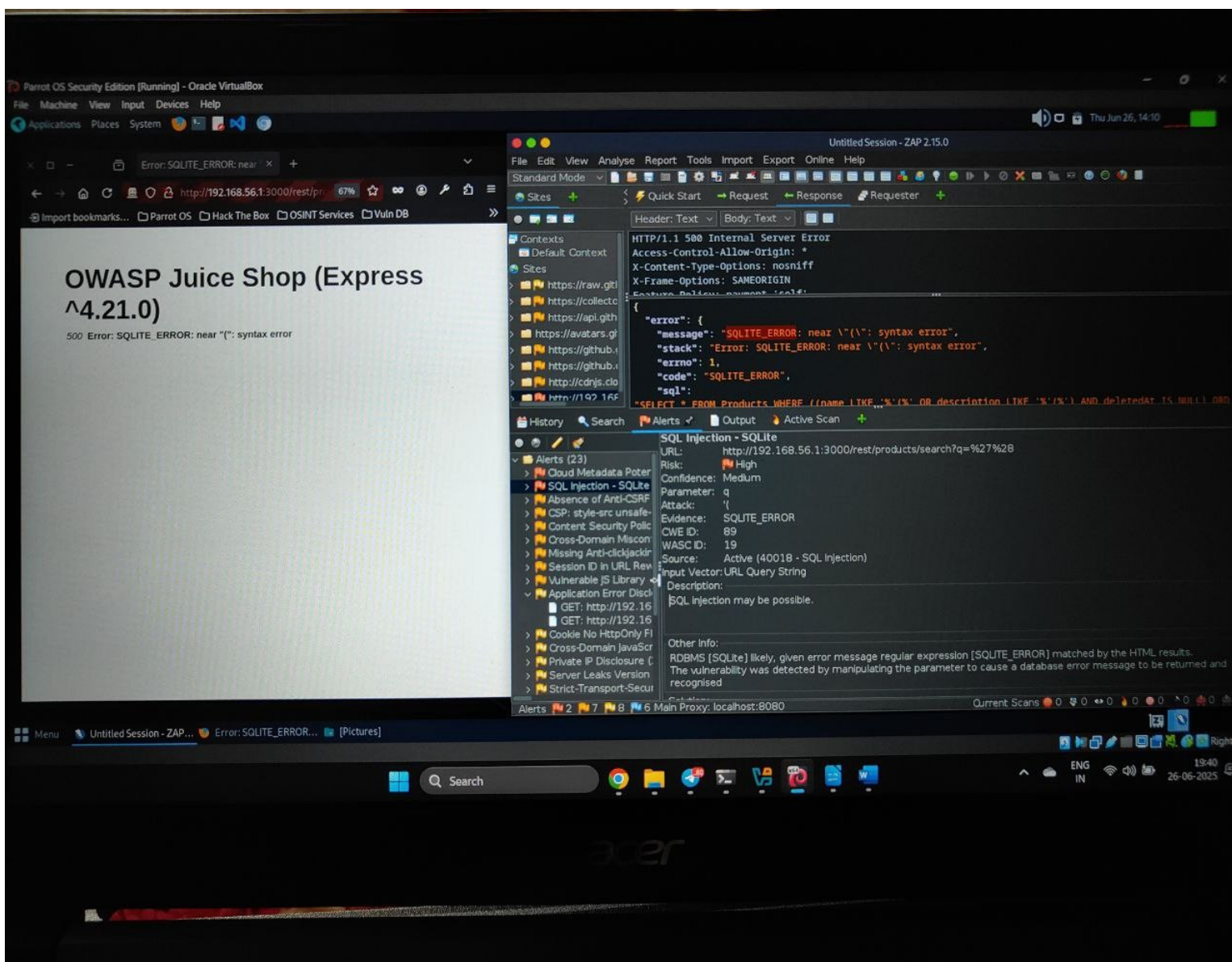
An attacker can potentially:

-  Bypass authentication mechanisms
-  Extract user data or database structure
- Modify or delete sensitive records
-  Escalate privileges or gain deeper access

Recommendations:

-  Use **parameterized queries** (prepared statements)
-  Avoid dynamic SQL with user input
-  Implement strict input validation
-  Deploy a Web Application Firewall (WAF) to block malicious payloads

Visual Evidence:




The screenshot displays a Parrot OS Security Edition virtual machine running ZAP (Zed Attack Proxy) 2.15.0. The browser window shows the OWASP Juice Shop (Express ^4.21.0) with a 500 Internal Server Error. The error message is "Error: SQLITE_ERROR: near \"(\": syntax error". The ZAP interface shows the HTTP response body with the error details and a SQL injection alert.

OWASP Juice Shop (Express ^4.21.0)
500 Error: SQLITE_ERROR: near \"(\": syntax error

SQL Injection - SQLite
URL: http://192.168.56.1:3000/rest/products/search?q=%27%28
Risk: High
Confidence: Medium
Parameter: q
Attack: '()
Evidence: SQLITE_ERROR
CWE ID: 89
WASC ID: 19
Source: Active (40018 - SQL Injection)
Input Vector: URL Query String
Description: SQL injection may be possible.

Other Info:
RDBMS [SQLite] likely, given error message regular expression [SQLITE_ERROR] matched by the HTML results. The vulnerability was detected by manipulating the parameter to cause a database error message to be returned and recognised.

Vulnerability #2: Cloud Metadata Potentially Exposed

- **Risk Level:**  Medium
- **URL Affected:** `http://192.168.56.1:3000/../../meta/data/?`
- **Parameter:** `q`
- **CWE ID:** CWE-200 – Exposure of Sensitive Information to an Unauthorized Actor
- **WASC ID:** 13
- **Source:** ZAP Active Scan

Description:

The application was found leaking internal metadata information by responding to crafted metadata-related requests. This vulnerability typically arises in cloud environments where special IPs like `169.254.169.254` are used by cloud providers (AWS, GCP, Azure) to serve metadata such as:

- IAM Role credentials
- Instance configuration details
- Networking setup

Responding to these requests indicates the app does **not properly restrict access to metadata endpoints**, opening the door for attackers to potentially gather sensitive cloud info.

Impact:

If hosted in a real cloud environment, attackers could:

- Extract temporary credentials
- Gain access to internal infrastructure
- Perform lateral movement or privilege escalation
- Potentially take over the instance


Mitigation:

- Block access to all requests targeting `169.254.169.254` and metadata-related paths.
- Ensure only trusted internal services have access to metadata endpoints.
- Implement firewall or web server rules to prevent this kind of path traversal.



Menu Untitled Session - ZAP... Error: Unexpected pat...

Vulnerability #3: Absence of Anti-CSRF Tokens

- **Risk Level:**  Medium
- **URL Affected:** Affected form endpoints (e.g., login, contact, etc.)
- **CWE ID:** CWE-352 – Cross-Site Request Forgery (CSRF)
- **WASC ID:** 9
- **Source:** ZAP Passive Scan

Description:

The application does **not implement CSRF protection mechanisms** like CSRF tokens in HTML forms or headers. Without these safeguards, an attacker can forge malicious requests and trick authenticated users into executing unintended actions.

This vulnerability was observed in the form:

```
<form id="query-builder-test-form" action="/..." method="GET">
```

There was **no CSRF token** embedded in this form, which leaves it susceptible to exploitation.

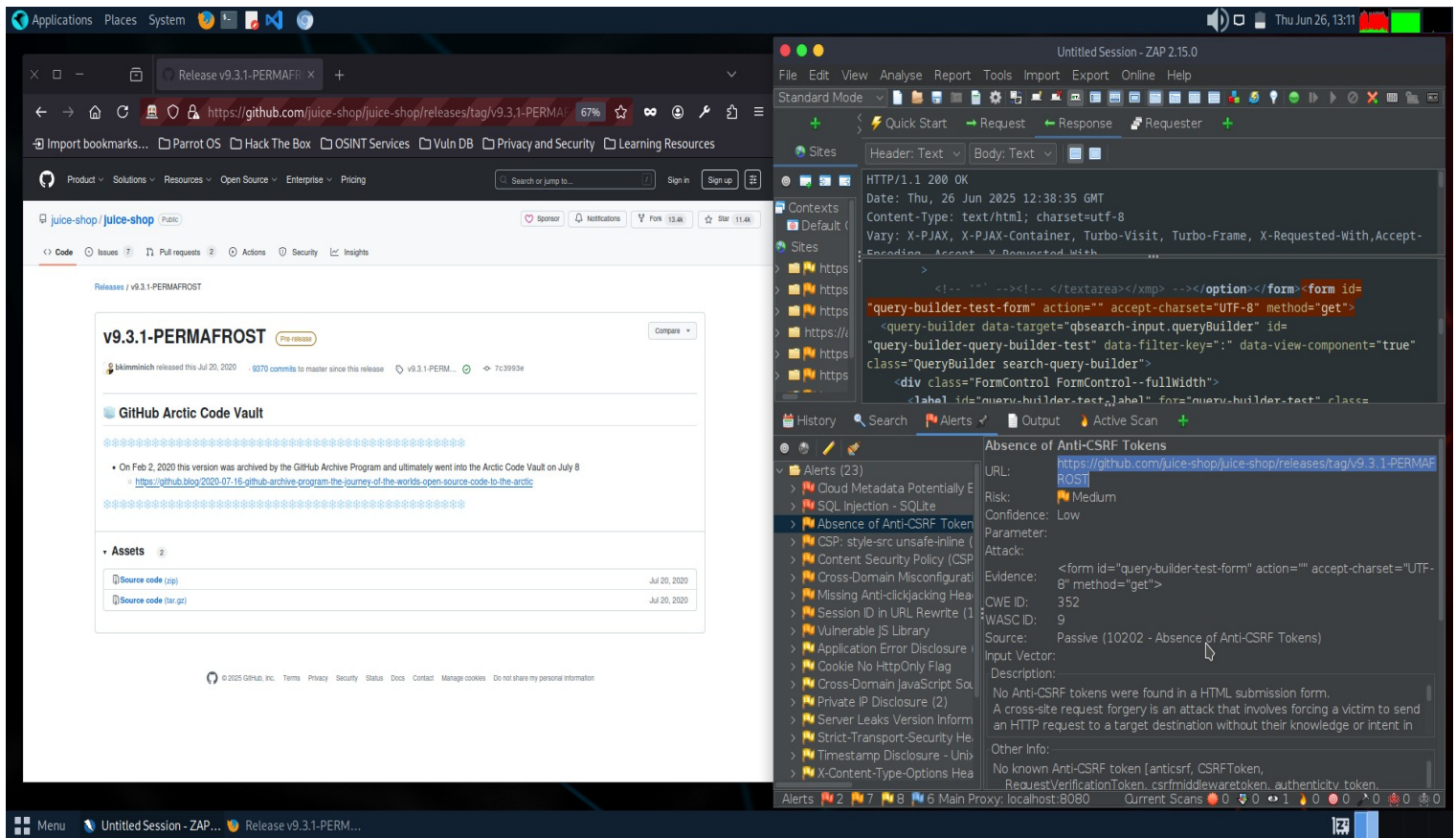
Impact:

- Account manipulation (change email, password)
- Unwanted form submissions
- Data tampering or deletion
- Potential privilege escalation depending on endpoint

Mitigation:

- Implement anti-CSRF tokens in all sensitive forms and endpoints.
- Use secure session management practices.
- Set `SameSite` cookie attributes appropriately (`SameSite=Strict` or `Lax`).
- Validate origin and referer headers on server-side.

Visual Evidence:




The screenshot displays a desktop environment with two main applications open: a web browser and a security tool (ZAP).

Web Browser (Left): The browser shows the GitHub release page for **juice-shop/juice-shop**, specifically the **v9.3.1-PERMAFROST** release. The page includes the GitHub logo, the repository name, and the release details. It mentions that the release was made on July 20, 2020, and has 67% of the code covered by tests. The page also lists the assets for this release, including the source code (zip) and the source code (tar.gz).

Security Tool (Right): The tool is **Untitled Session - ZAP 2.15.0**. It shows a list of alerts, with the **Absence of Anti-CSRF Tokens** alert selected. The alert details include the URL (<https://github.com/juice-shop/juice-shop/releases/tag/v9.3.1-PERMAFROST>), the risk level (Medium), and the confidence (Low). The attack details show the request body, which includes a form submission with the following data: `<form id="query-builder-test-form" action="" accept-charset="UTF-8" method="get">`, `<query-builder data-target="qbsearch-input.queryBuilder" id="query-builder-query-builder-test" data-filter-key="" data-view-component="true" class="QueryBuilder search-query-builder">`, and `<div class="FormControl FormControl--fullWidth">`. The evidence section states: "No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in".

Vulnerability #4: Private IP Disclosure

- **Risk Level:**  Medium
- **URL Affected:** `http://192.168.56.1:3000/rest/admin/application-configuration`
- **WASC ID:** 13 – Information Leakage
- **CWE ID:** CWE-200 – Information Exposure
- **Source:** ZAP Passive Scan

Description:

The HTTP response body **contains internal IP addresses** such as:

```
192.168.99.100  
127.0.0.1
```

These private addresses can provide attackers with **network layout information**, making **pivoting attacks** or **internal scanning** easier if access is gained. This vulnerability can lead to **information disclosure**, especially in cloud or hybrid environments.

Impact:

- Leaks sensitive infrastructure info
- Helps attackers map internal services
- Assists in SSRF, lateral movement, or privilege escalation attempts

Mitigation:

- Remove or sanitize debug data and internal IPs from response bodies
- Avoid exposing server-side configuration files publicly
- Implement Content Security Policies to restrict data flow
- Conduct regular source code and config audits for sensitive information leaks

Visual Evidence:


The screenshot displays the ZAP (Zed Attack Proxy) interface, which is a security tool used for testing web applications. The interface is divided into several panes:

- Left Pane (JSON View):** Shows the JSON response of a REST client session. The response is a configuration object for a web application named "OWASP Juice Shop". It includes details such as the domain, name, logo, favicon, theme, and various settings for the application. The response is expanded to show the "config" object, which contains a "servers" array, an "application" object, a "chatbot" object, a "social" object, a "recyclePage" object, a "welcomeScreen" object, a "cookieConsent" object, and a "welcomeScreen" object.
- Right Pane (Response View):** Shows the HTTP response details. The status is "HTTP/1.1 200 OK". The headers include "Access-Control-Allow-Origin: *", "X-Content-Type-Options: nosniff", "X-Frame-Options: SAMEORIGIN", and "Feature-Policy: payment 'self'". The body of the response is a JSON object representing the application configuration.
- Bottom Pane (Alerts):** Displays a list of alerts generated by ZAP. The alerts are categorized by risk level (Low, Medium, High) and include details such as the URL, risk level, confidence, parameter, attack, evidence, CWE ID, WASC ID, and source. The alerts are sorted by risk level, with "Private IP Disclosure (2)" being the most prominent.

The "Private IP Disclosure (2)" alert is highlighted, showing the following details:

- URL:** <http://192.168.56.1:3000/rest/admin/application-configuration>
- Risk:** Low
- Confidence:** Medium
- Parameter:**
- Attack:**
- Evidence:** 192.168.99.100:3000
- CWE ID:** 200
- WASC ID:** 13
- Source:** Passive (2 - Private IP Disclosure)
- Input Vector:**
- Description:** A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting
- Other Info:** 192.168.99.100:3000, 192.168.99.100:4200

Vulnerability #5: Cross-Domain Misconfiguration (CORS)

- **Risk Level:**  Medium
- **URL Affected:** `http://192.168.56.1:3000/`
- **WASC ID:** 15 – Application Misconfiguration
- **CWE ID:** CWE-264 – Permissions, Privileges, and Access Controls
- **Source:** ZAP Passive Scan

Description:

The server is configured to allow **Cross-Origin Resource Sharing (CORS)** through this header:

`Access-Control-Allow-Origin: *`

This means any external domain can **read responses from the server**, potentially leading to **sensitive data exposure** or **unauthorized interactions**, especially if cookies or authentication tokens are accessible via JavaScript.

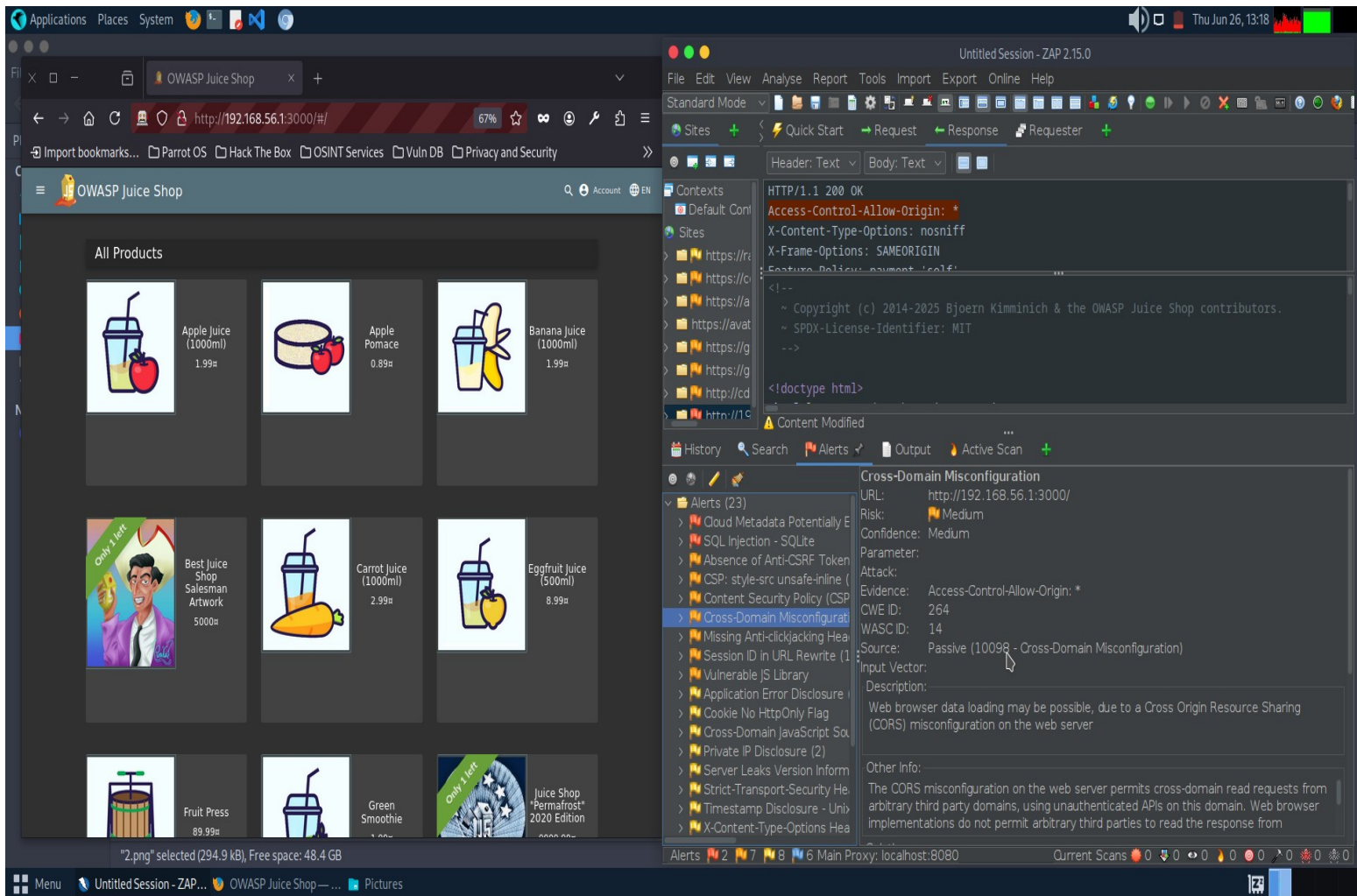
Impact:

- Malicious websites can access data from your site
- Sensitive data such as user sessions or API responses could be leaked
- Attackers can exploit this in **CSRF** or **XSS** scenarios to increase attack surface

Mitigation:

- Avoid using wildcard `*` in CORS policy unless absolutely necessary
- Implement **CORS whitelisting**: only allow trusted origins
- Ensure sensitive endpoints are protected with **authentication and CSRF tokens**
- Perform regular security reviews of HTTP headers and CORS configs

Visual Evidence:



The screenshot displays the OWASP Juice Shop website on the left and the ZAP (Zed Attack Proxy) tool interface on the right. The website shows a grid of products including Apple Juice, Apple Pomace, Banana Juice, Best Juice Shop Salesman Artwork, Carrot Juice, Eggfruit Juice, Fruit Press, Green Smoothie, and Juice Shop "Permafrost" 2020 Edition. The ZAP tool interface shows the "Alerts" tab with a list of alerts. The selected alert is "Cross-Domain Misconfiguration" (CWE-264), which is a "Passive (10098 - Cross-Domain Misconfiguration)" issue. The alert details include the URL, Risk (Medium), Confidence (Medium), Parameter, Attack, Evidence, CWE ID, WASC ID, Source, Input Vector, Description, and Other Info.

Alert Details:

- Alerts (23)**
- Cross-Domain Misconfiguration** (CWE-264)
- Source:** Passive (10098 - Cross-Domain Misconfiguration)
- 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



Risk Assessment

This section provides a summarized evaluation of the identified vulnerabilities based on their risk levels and potential business impact. The goal is to help prioritize remediation efforts and highlight critical areas of concern

<div><div>1234</div><div>No.</div></div>	Vulnerability	Risk Level	CWE ID	Affected Area	Potential Impact
1	SQL Injection	<div></div> High	CWE-89	Search Query Field	Database manipulation, data leakage, possible RCE
2	Cloud Metadata Exposure	<div></div> Medium	CWE-200	Meta-data response	Internal infrastructure exposure
3	Absence of Anti-CSRF Tokens	<div></div> Medium	CWE-352	Search Form Submission	Cross-Site Request Forgery attacks
4	Private IP Disclosure	<div></div> Medium	CWE-200	Admin API response	Target reconnaissance, internal mapping
5	Cross-Domain Misconfiguration (CORS)	<div></div> Medium	CWE-264	HTTP Headers	Unauthorized resource access from other origins

✓ Conclusion

The security assessment of the **OWASP Juice Shop** test application successfully identified **five key vulnerabilities** that reflect common flaws found in real-world web applications. These vulnerabilities include:

- **🔴 SQL Injection** — A critical flaw that could allow attackers to manipulate database queries and extract sensitive information.
- **⚠️ Cloud Metadata Exposure** — An issue that could expose internal IPs or metadata if hosted on cloud infrastructure.
- **⚠️ Private IP Disclosure** — May assist attackers in understanding the internal network architecture.
- **⚠️ Absence of Anti-CSRF Tokens** — Leaves the application open to Cross-Site Request Forgery attacks.
- **⚠️ Cross-Domain Misconfiguration (CORS)** — Could allow malicious websites to interact with the application's APIs.

These findings demonstrate how improperly secured applications can be exploited by attackers. Each vulnerability was **manually verified**, categorized based on **risk level**, and cross-mapped with **OWASP Top 10** and **CWE references**.

This project helped enhance practical skills in:

- **🔍** Web application reconnaissance & vulnerability scanning
- **🔧** Using tools like **OWASP ZAP** to simulate real-world attacks
- **📄** Preparing a **professional Security Assessment Report** aligned with industry standards

💡 Final Note

Security is not a one-time effort — it's a continuous cycle of detection, remediation, and improvement. Developers, DevOps teams, and security engineers should collaborate to integrate secure coding practices and routine testing into the development lifecycle.

By completing this task, the foundation has been laid for real-world penetration testing, and the report can serve as a strong addition to any professional cybersecurity portfolio.



Annexure



Tools Used

- **OWASP Juice Shop** — Deliberately vulnerable web application used as the testing environment.
 - **OWASP ZAP** — Primary tool for scanning and analyzing vulnerabilities.
 - **Parrot OS** — Penetration testing environment with essential cybersecurity utilities.
 - **Web Browser (Firefox/Chromium)** — For navigating and testing the target application manually.
-



References

- OWASP Juice Shop Official Site
 - OWASP Top 10 (2021)
 - CWE Database
 - OWASP ZAP Documentation
-



Acknowledgements

This project was conducted as part of the **Future Interns - Future Intern Program**, designed to provide hands-on experience in real-world cybersecurity assessments.

Special thanks to:

- **Future Interns Team** for curating this internship opportunity.
- The **OWASP community** for providing open-source learning platforms and tools.

