

OWASP JUICE SHOP

PENETRATION TESTING

REPORT

Threat Analysis & Penetration Testing Report



DECEMBER 10, 2025
ATHARVA KASAR

Contents

1.	Introduction.....	2
2.	Executive Summary.....	3
3.	Identified Vulnerabilities.....	4
4.	Proof of Concept (PoC) Evidence.....	6
5.	Risk Analysis	16
6.	Impact	18
7.	Mitigations.....	19
8.	Real World Scenarios.....	20
9.	Conclusion	21

1. Introduction

Web applications have become an essential component of modern digital infrastructure, supporting critical business functions and handling sensitive user data. Their widespread accessibility, complex architectures, and reliance on dynamic user interaction make them a primary target for cyberattacks. Vulnerabilities introduced through insecure coding practices, improper configuration, or insufficient validation mechanisms can be easily exploited by attackers, leading to data breaches, unauthorized access, and service disruption. As attack techniques continue to evolve, understanding and addressing web application security risks has become a fundamental requirement for organizations.

OWASP Juice Shop is a deliberately vulnerable web application developed to provide a realistic platform for learning and practicing web application security testing. Built using modern web technologies, Juice Shop closely resembles real-world single-page applications and incorporates a broad range of vulnerabilities mapped to the OWASP Top 10. These include Broken Authentication, Broken Access Control, Security Misconfiguration, Sensitive Data Exposure, and various client-side security flaws. The application is designed to simulate practical attack scenarios, allowing security professionals to understand how vulnerabilities manifest in contemporary web environments.

This security assessment focuses on identifying and analyzing vulnerabilities present within OWASP Juice Shop by adopting an attacker-oriented testing approach. The objective is to evaluate how weaknesses in authentication, authorization, API design, and client-side logic can be exploited to compromise application security. By examining these vulnerabilities in a controlled environment, the assessment highlights the importance of proactive security testing and secure development practices.

2. Executive Summary

This security assessment report presents a comprehensive analysis of a purposely vulnerable web application, **OWASP Juice Shop**, which is widely used within the cybersecurity community for training, simulation, and vulnerability assessment. OWASP Juice Shop is designed to replicate real-world web application architectures using modern frameworks and API-driven functionality. The primary objective of this evaluation is to identify, understand, and assess security weaknesses based on the OWASP Top 10 vulnerability categories, which are widely recognized as industry benchmarks for web application security.

Through detailed manual testing and request-level analysis using tools such as Burp Suite and browser-based inspection techniques, this assessment uncovered multiple high and medium severity vulnerabilities commonly observed in production environments. The key findings include:

- **Broken Access Control flaws**, allowing unauthorized users to access restricted resources, administrative functionalities, or sensitive data by directly manipulating application endpoints.
- **Broken Authentication vulnerabilities**, which expose the application to credential-stuffing and brute-force attacks due to weak login controls, insufficient rate limiting, and improper session management.
- **Sensitive Data Exposure**, resulting from insecure handling of personal and financial information through poorly protected APIs, exposed files, or insufficient encryption mechanisms.
- **CAPTCHA Bypass vulnerabilities**, which allow automated abuse of application features due to weak or client-side CAPTCHA validation mechanisms.
- **Security Misconfiguration issues**, including verbose error messages, exposed framework details, and misconfigured HTTP headers that provide attackers with valuable information about the internal application structure.

These vulnerabilities closely mirror those frequently exploited in real-world cyberattacks, reinforcing the urgent need for organizations to embed security controls throughout the application development lifecycle. The findings from this assessment highlight how multiple seemingly independent flaws can be chained together by attackers to escalate privileges and compromise application integrity.

Ultimately, this report emphasizes the importance of continuous security testing, secure coding practices, and security awareness across development and operational teams. Each identified issue is supported by technical analysis, risk evaluation, and mitigation strategies aimed at strengthening the overall security posture of modern web applications. With the increasing complexity of web technologies and evolving threat landscapes, proactive security measures remain essential to safeguard applications against exploitation.

3. Identified Vulnerabilities

Vulnerability	Description	Severity
Broken Access Control	Improper input validation allows attackers to manipulate backend SQL queries and alter application logic.	High
Broken Authentication	SQL injection vulnerability where database responses are inferred using time delays instead of visible output.	High
Sensitive Data Exposure	Malicious JavaScript payloads are stored on the server and executed whenever users access the affected page.	High
CAPTCHA Bypass	Client-side JavaScript insecurely processes user input directly in the DOM, enabling script execution without server involvement.	Medium
Security Misconfiguration	Malicious scripts are injected via request parameters and reflected immediately in server responses.	Medium

Overview of Vulnerability Distribution

- **High Severity Vulnerabilities**
 - Broken Access Control
 - Broken Authentication
 - Sensitive Data Exposure

These vulnerabilities represent serious security risks as they allow attackers to gain unauthorized access to restricted resources, bypass authentication mechanisms, and access or leak sensitive personal and financial data. Exploitation of these flaws can lead to account takeover, privilege escalation, and complete compromise of application confidentiality and integrity.

- **Medium Severity Vulnerabilities**

- CAPTCHA Bypass
- Security Misconfiguration

While these vulnerabilities may not immediately lead to direct system compromise, they significantly weaken the application's overall security posture. CAPTCHA bypass can enable automated abuse and brute-force attacks, while security misconfigurations expose internal application details that assist attackers in planning further exploitation. When combined with high-severity issues, these vulnerabilities can greatly amplify attack impact.

4. Proof of Concept (PoC) Evidence

1) Broken Access Control

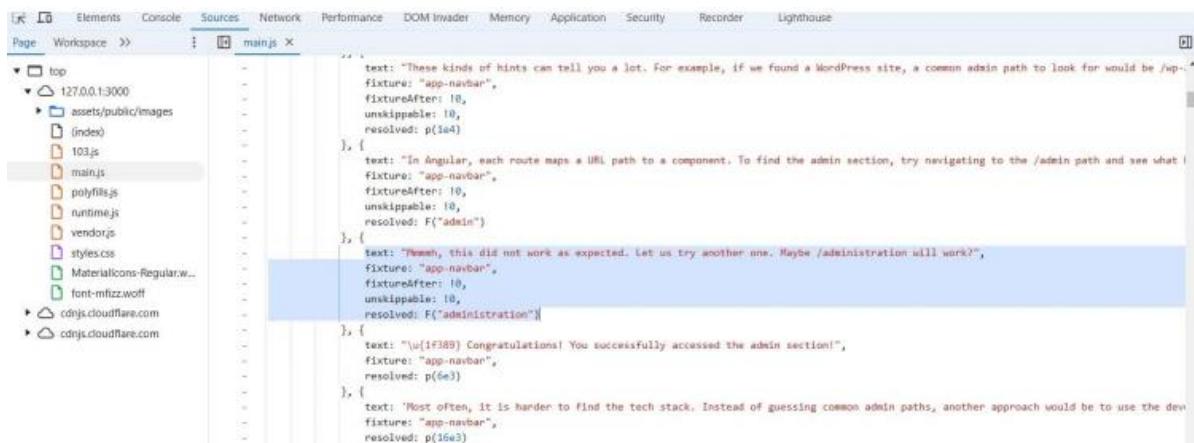
During the security assessment of OWASP Juice Shop, a Broken Access Control vulnerability was identified that allowed unauthorized users to access restricted functionalities intended only for privileged roles. This vulnerability exists due to improper enforcement of authorization checks on the server side, relying primarily on client-side controls to restrict access.

To exploit this vulnerability, I logged into the application as a standard, non-administrative user and manually accessed a restricted administrative endpoint by directly entering the URL in the browser. Despite the absence of any administrative privileges, the application loaded the restricted page successfully without displaying an authorization error. The response confirmed that the backend failed to verify user roles before granting access, allowing unauthorized exposure of administrative functionality and sensitive controls.

Key Execution Steps:

- Logged in as a standard (non-admin) user.
- Attempted to access the admin page directly by visiting /administration.
- Found that the page loaded successfully without any access restrictions.
- Confirmed the presence of admin-only content and options available to a regular user.

The successful access to restricted resources confirmed a Broken Access Control vulnerability. In real-world applications, such flaws can lead to privilege escalation, unauthorized data access, and complete compromise of application integrity.



The screenshot shows the Chrome DevTools Network tab with a request to 'main.js'. The response body contains several JSON objects representing test cases for different tech stacks. One object is highlighted in blue, showing the following content:

```
    }, {
      text: "In Angular, each route maps a URL path to a component. To find the admin section, try navigating to the /admin path and see what I",
      fixture: "app-navbar",
      fixtureAfter: 10,
      unskippable: 10,
      resolved: F("admin")
    }, {
      text: "Hmmm, this did not work as expected. Let us try another one. Maybe /administration will work?",
      fixture: "app-navbar",
      fixtureAfter: 10,
      unskippable: 10,
      resolved: F("administration")
    }, {
      text: "\ud83d\udc89 Congratulations! You successfully accessed the admin section!",
      fixture: "app-navbar",
      resolved: p(6e3)
    }, {
      text: "Most often, it is harder to find the tech stack. Instead of guessing common admin paths, another approach would be to use the dev",
      fixture: "app-navbar",
      resolved: p(3e3)
    }
```

The screenshot shows the OWASP Juice Shop application running locally at 127.0.0.1:3000. The main page displays a green banner stating, "This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wot!" with a "Me want it!" button. Below the banner, the page title is "All Products". The developer tools (Chrome DevTools) are open, specifically the Sources tab, which shows the application's code structure. The right sidebar of the DevTools contains the Breakpoints and Call Stack sections.

2) Broken Authentication

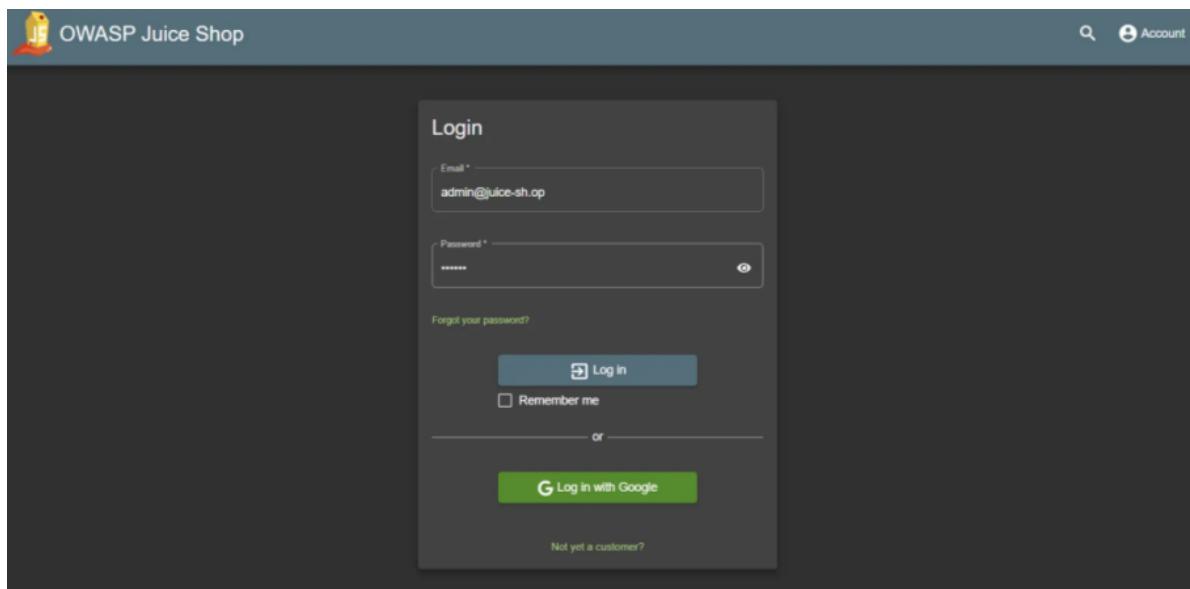
Broken Authentication was identified during testing of the login functionality in OWASP Juice Shop. This vulnerability arises due to weak authentication controls, including lack of rate limiting and insufficient protection against automated login attempts, making the application susceptible to brute-force and credential-stuffing attacks.

To validate this vulnerability, the login request was intercepted and analyzed to understand its structure. Multiple authentication attempts were then sent using different credential combinations without triggering any account lockout, CAPTCHA enforcement, or rate-limiting mechanism. The application continued to respond normally even after numerous failed login attempts, indicating the absence of effective brute-force protection.

Key Execution Steps:

- Accessed the /login endpoint in OWASP Juice Shop.
- Intercepted login request with Burp Suite to analyze the request structure.
- Used Burp Suite Intruder to automate login attempts with a wordlist.
- Observed that no CAPTCHA, rate limiting, or account lockout was in place.
- Successfully logged in with a valid credential after multiple attempts.

This behavior confirms that the authentication mechanism is weak and vulnerable to automated attacks. In real-world scenarios, such vulnerabilities can result in account takeover, unauthorized access, and exposure of sensitive user data.



A screenshot of the Burp Suite Professional interface. The top navigation bar includes "Burp", "Project", "Intruder", "Repeater", "View", "Auth Analyzer", "Help", "Param Miner", "Burp Suite Professional v2025.1 - Temporary Project - Licensed to: DrFarFar (WwW.Dr-FarFar.Cold)", "Dashboard", "Target", "Proxy", "Intruder", "Repeater", "Collaborator", "Sequencer", "Decoder", "Comparer", "Logger", "Organizer", "Extensions", "Learn", "Burp Bounty Free", "Auth Analyzer", "Bypass WAF", "XSS Validator", "Search", "Settings", and "Logout". The main window shows the "Intruder" tab selected. On the left, there's a "Sniper attack" configuration with a "Target" set to "http://127.0.0.1:3000". Below it are buttons for "Positions", "Add \$", "Clear \$", and "Auto \$". The main pane displays a POST request to "/rest/user/login" with various headers and a JSON payload. The payload is defined in the "Payloads" panel on the right, which shows a "Simple list" type with 999,998 entries. The first few entries are "123456", "password", "12345678", "12345", "1234", "111111", and "1234567". Below the payloads is a "Payload configuration" section. The bottom right panel shows "Payload processing" with a table for defining rules.

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
13369	admin123	200	239			1197	Contains a JWT
0		401	453			413	
1	123456	401	412			413	
2	password	401	204			413	
3	12345678	401	466			413	
4	qweerty	401	548			413	
5	123456789	401	104			413	
6	12345	401	5640			413	
7	1234	401	372			413	
8	111111	401	644			413	
9	1234567	401	595			413	
10	dragon	401	282			413	
11	123123	401	423			413	
12	baseball	401	425			413	
13	abct123	401	423			413	
14	pearl	401	422			413	
15	monkey	401	437			413	
16	lmlein	401	448			413	
17	696969	401	439			413	
18	shadow	401	455			413	
19	master	401	463			413	
20	666666	401	441			413	
21	qweertyuiop	401	406			413	
22	123321	401	399			413	
23	mustang	401	399			413	
24	1234567890	401	398			413	

3) Sensitive Data Exposure

Sensitive Data Exposure was identified during analysis of application endpoints and exposed resources within OWASP Juice Shop. This vulnerability exists due to improper access control and insecure handling of sensitive information, allowing unauthorized users to access confidential data.

During exploitation, publicly accessible endpoints and directories were accessed directly through the browser. These endpoints exposed files and API responses containing sensitive information without requiring authentication. Network traffic analysis further revealed that sensitive data was transmitted without adequate protection, making it accessible to unauthorized users.

Key Execution Steps:

- Accessed /ftp endpoint in browser and found publicly visible files.
- Opened and downloaded exposed files containing potentially sensitive data.
- Verified lack of authentication and encryption over the connection.
- Intercepted traffic via Burp Suite and confirmed plain-text transmission.

The exposure of sensitive information confirms a critical data protection flaw. In production environments, such vulnerabilities can lead to data breaches, identity theft, and serious compliance violations.

Request

	Request	Response	Inspector
1	GET /ftp/legal.md http/1.1	HTTP/1.1 304 Not Modified	Request attributes
2	Host: 127.0.0.1:3000	Access-Control-Allow-Origin: *	Request cookies
3	User-Agent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/137.36	Content-Type: application/javascript; charset: UTF-8	Request headers
4	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng;q=0.8,application/xsigned-exchange;v=b3;q=0.7	ETag: W/"be7-1057e376939"	Response headers
5	Sec-Fetch-Site: none	Date: Mon, 10 Mar 2025 04:05:21 GMT	
6	Sec-Fetch-Mode: navigate	Connection: keep-alive	
7	Sec-Fetch-User: ?1	Keep-Alive: timeout=5	
8	Sec-Fetch-Dest: document	14	
9	Referrer: http://127.0.0.1:3000/	15	
10	Accept-Encoding: gzip, deflate, br		
11	Cookie: language=en; welcomebanner_status=dimiss		
12	**		

```

Request
Pretty Raw Hex
1 GET /ftp/legal.md HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua: "Not A[Brand];v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss
17 If-Match: W/"be7-1957e376939"
18 If-Modified-Since: Mon, 10 Mar 2025 04:01:55 GMT
19 Connection: keep-alive

Response
Pretty Raw Hex Render
1 HTTP/1.1 304 Not Modified
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#jobs
7 Accept-Ranges: bytes
8 Cache-Control: public, max-age=0
9 Last-Modified: Mon, 10 Mar 2025 04:01:55 GMT
10 ETag: W/"be7-1957e376939"
11 Date: Mon, 10 Mar 2025 04:05:47 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15

Request
Pretty Raw Hex
1 GET /ftp HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua: "Not A[Brand];v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss
17 If-Match: W/"be7-1957e376939"
18 If-Modified-Since: Mon, 10 Mar 2025 04:01:55 GMT
19 Connection: keep-alive

Response
Pretty Raw Hex Render
1 <script>
2 </head>
3 <body class="directory">
4   <input id="search" type="text" placeholder="Search" autocapitalize="off" />
5   <ul id="files" class="view-tiles">
6     <li>
7       <a href="#">
8         <script>
9           </script>
10        </a>
11        <a href="ftp">
12          ftp
13        </a>
14        <ul id="files" class="view-tiles">
15          <li>
16            <a href="ftp/quarantine" class="icon icon-directory" title="quarantine">
17              quarantine
18            </a>
19            <span class="size">
20              10/28/2024 3:55:15 PM
21            </span>
22          </li>
23        </li>
24      </ul>
25    <li>
26      <a href="ftp/acquisitions.md" class="icon icon-icon-test" title="acquisitions.md">
27        acquisitions.md
28      </a>
29      <span class="size">
30        909
31      </span>
32    </li>
33  </ul>
34</body>

```

4) CAPTCHA Bypass

A CAPTCHA Bypass vulnerability was identified in OWASP Juice Shop within functionality designed to prevent automated abuse. The CAPTCHA mechanism was implemented weakly and lacked proper server-side validation, allowing attackers to bypass it programmatically.

To exploit this vulnerability, the request generated during form submission was intercepted and examined. The CAPTCHA value was found to be included directly in the request parameters. By replaying and modifying the request without solving the CAPTCHA challenge, multiple submissions were successfully processed by the server, confirming that CAPTCHA validation was not enforced on the backend.

Key Execution Steps:

- Navigated to the /contact feedback form.
- Observed CAPTCHA parameter in the request payload via Burp Suite.
- Captured and replayed the request using Burp Repeater without solving CAPTCHA.
- Automated the request with a custom script to submit feedback repeatedly.
- Confirmed that CAPTCHA challenge was bypassed and not validated server-side.

This demonstrates that the CAPTCHA mechanism is ineffective against automated attacks. In real-world applications, such weaknesses can enable large-scale abuse, spam attacks, and automation of other vulnerabilities.

The screenshot shows the 'Customer Feedback' form on the OWASP Juice Shop website. The form includes fields for 'Author' (set to 'anonymous'), 'Comment' (containing 'afsdifgdfsg'), and a 'Rating' slider set to 3. A CAPTCHA challenge is present: 'CAPTCHA: What is 3*5+1?'. Below it, a red-bordered input field prompts the user to 'Please enter the result of the CAPTCHA.' A note below the field says 'Please enter the result of the CAPTCHA.' A 'Submit' button is at the bottom.

Request	Response
<pre>Pretty Raw Hex 1 POST /api/Feedbacks/ HTTP/1.1 2 Host: 127.0.0.1:3000 3 Content-Length: 78 4 sec-ch-ua-platform: "Windows" 5 Accept-Language: en-US,en;q=0.9 6 Accept: application/json, text/plain, */* 7 sec-ch-ua: "Not A[Brand];v="8", "chromium";v="132" 8 Content-Type: application/json 9 sec-ch-ua-mobile: ?0 10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) 11 Chrome/132.0.6600.136 Safari/537.36 12 Origin: http://127.0.0.1:3000 13 Sec-Fetch-Site: same-origin 14 Sec-Fetch-Mode: cors 15 Sec-Fetch-Dest: empty 16 Referer: http://127.0.0.1:3000/ 17 Accept-Encoding: gzip, deflate, br 18 Cookie: language=en; welcomebanner_status.dismiss; cookieconsent_status.dismiss; 19 continueCode=jxH0Qpd5nR7aEVzLThCIM4oGMcfYyu7luH6opq9jQIDbJxyEHB0Ye3vZM 20 Connection: keep-alive 21 22 { 23 "captchaId":3, 24 "captcha": "16", 25 "comment": "afsdifgdfsg (anonymous)", 26 "rating":3 27 }</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 201 Created 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Location: /api/Feedbacks/17 8 Content-Type: application/json; charset=utf-8 9 Content-Length: 177 10 Vary: Accept-Encoding 11 Date: Mon, 10 Mar 2025 19:03:04 GMT 12 Connection: keep-alive 13 Keep-Alive: timeout=5 14 15 16 { 17 "status": "success", 18 "data": { 19 "id": 17, 20 "comment": "afsdifgdfsg (anonymous)", 21 "rating": 3, 22 "updatedAt": "2025-03-10T19:03:04.114Z", 23 "createdAt": "2025-03-10T19:03:04.114Z", 24 "userId": null 25 } 26 }</pre>

The screenshot shows the JSON Web Tokens tool interface. At the top, there's a toolbar with various icons. Below it, the 'JSON Web Tokens' tab is active. The 'Positions' tab is selected, showing a list of captured headers and their values. The 'Payloads' tab is also visible, showing settings for payload generation. A message at the bottom indicates that this payload type generates payloads whose value is an empty string.

The screenshot shows a success message: "You successfully solved a challenge: CAPTCHA Bypass (Submit 10 or more customer feedbacks within 20 seconds.)". Below this, a modal window titled "Customer Feedback" is displayed. It has fields for "Author" (set to "anonymous"), "Comment" (with a note about max 160 characters), "Rating" (a slider set to 1), and a "CAPTCHA" field containing the code "8-1-8". There is also a "Result" field which is currently empty.

5) Security Misconfiguration

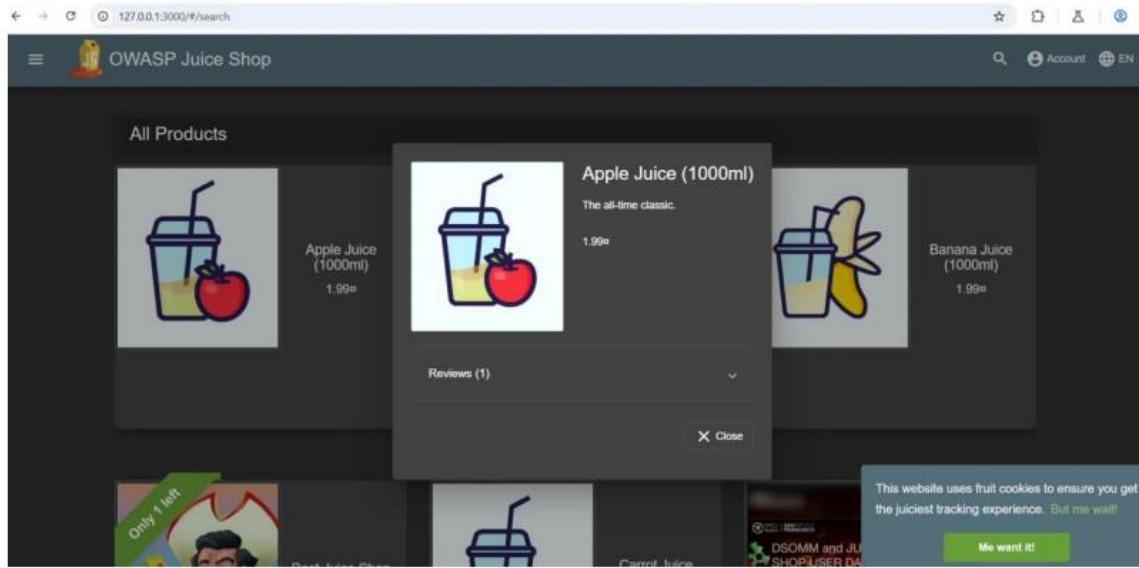
Security Misconfiguration was identified during inspection of application responses, headers, and error handling mechanisms in OWASP Juice Shop. This vulnerability arises when development or debugging configurations are exposed in a production-like environment.

While interacting with various application endpoints, verbose error messages and unnecessary response headers revealing internal framework details were observed. Certain API responses exposed internal object structures and configuration information when invalid requests were sent. These disclosures provided valuable insight into the application's internal workings.

Key Execution Steps:

- Accessed the /rest/products/ endpoint and observed verbose responses.
- Inspected response headers (e.g., X-Powered-By) disclosing framework details.
- Sent malformed input to test error handling and received detailed error messages.
- Confirmed exposure of internal structures, suggesting insecure configuration.

Such misconfigurations significantly aid attackers in reconnaissance and vulnerability chaining. Although they may not cause immediate compromise on their own, they greatly increase the effectiveness of other attacks when exploited together.



The screenshot shows the NetworkMiner tool interface. The top navigation bar includes tabs for Intercept, HTTP history (which is selected), WebSockets history, Match and replace, and Proxy settings. Below the tabs is a filter settings dropdown. The main pane displays a list of captured network requests and responses, each with columns for Host, Method, URL, Params, Status code, Length, MIME type, Extension, Title, Notes, TLS, IP, Cookies, Time, Listener port, and Start response. The list contains approximately 100 entries, mostly from the OWASP Juice Shop application. The bottom of the interface has three panels: Request (Pretty, Raw, Hex), Response (Pretty, Raw, Hex, Render), and Inspector (Request attributes, Request cookies).

#	Host	Method	URL	Params	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
78	http://127.0.0.1:3000	GET	/#/legal/nd		200	3517	text	md			127.0.0.1			09:35:10 10 -	8081	17
79	http://127.0.0.1:3000	GET	/favicon.ico		200	4214	HTML	ico	OWASP Juice Shop		127.0.0.1			09:35:10 10 -	8081	28
80	http://127.0.0.1:3000	GET	/#/legal/nd		304	391					127.0.0.1			09:35:21 10 -	8081	14
82	http://127.0.0.1:3000	GET	/socket.io/BIO+4&transport=polling..✓		200	326	JSON	js			127.0.0.1			09:35:28 10 -	8081	6
83	http://127.0.0.1:3000	POST	/socket.io/BIO+4&transport=polling..✓		200	215	text	js			127.0.0.1			09:35:28 10 -	8081	5
84	http://127.0.0.1:3000	GET	/socket.io/BIO+4&transport=polling..✓		200	262	JSON	js			127.0.0.1			09:35:28 10 -	8081	5
85	http://127.0.0.1:3000	GET	/socket.io/BIO+4&transport=webso..✓		101	129					127.0.0.1			09:35:28 10 -	8081	21
86	http://127.0.0.1:3000	GET	/socket.io/BIO+4&transport=polling..✓		200	230	text	js			127.0.0.1			09:35:28 10 -	8081	92
87	http://127.0.0.1:3000	GET	/rest/continue-code		200	463	JSON				127.0.0.1			09:49:31 10 -	8081	10
88	http://127.0.0.1:3000	GET	/103.js		200	11606	script	js			127.0.0.1			09:49:31 10 -	8081	30
89	http://127.0.0.1:3000	GET	/rest/products/search?q=✓		304	306					127.0.0.1			09:51:33 10 -	8081	105
90	http://127.0.0.1:3000	GET	/api/Quantity/✓		304	306					127.0.0.1			09:51:33 10 -	8081	199
103	http://127.0.0.1:3000	GET	/rest/users/vh.com		200	394	JSON				127.0.0.1			09:56:35 10 -	8081	15
104	http://127.0.0.1:3000	GET	/rest/products/1/reviews		200	357	JSON				127.0.0.1			09:56:35 10 -	8081	80
105	http://127.0.0.1:3000	GET	/rest/products/1/reviews		304						127.0.0.1			09:56:35 10 -	8081	19
106	http://127.0.0.1:3000	GET	/rest/products/1/reviews		304	304					127.0.0.1			09:56:36 10 -	8081	23

Request

Pretty Raw Hex

```

1 GET /rest/documents HTTP/1.1
2 Host: 127.0.0.1:3000
3 accept-encoding: "gzip, deflate"
4 Accept-Language: en-US,en;q=0.8
5 Accept: application/json, text/plain, */*
6 sec-ch-ua: "Not A Brand";v="0", "Chromium";v="102"
7 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36
8 sec-ch-ua-mobile: ?0
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: http://127.0.0.1:3000/
13 Accept-Encoding: gzip, deflate, br
14 Cookie: language=en; welcomebanner_status=dimension; continueCode=
YzIxREUjafoxBGIMkVZuJ5qCiyIPAZP0M3epzBQ34XbBgwDE7nVrYgkjql08
15 Connection: keep-alive
16
17

```

Response

Pretty Raw Hex Render

```

1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 Content-Type: application/json; charset=utf-8
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#jobs
7 Content-Type: application/json; charset=utf-8
8 Vary: Accept-Encoding
9 Date: Mon, 10 Mar 2023 04:38:02 GMT
10 Connection: keep-alive
11 Keep-Alive: timeout=5
12 Content-Length: 1045
13
14 {
15   "error": {
16     "message": "Unexpected path: /rest/documents",
17     "stack": "Error: Unexpected path: /rest/documents\n    at /juice-shop/build/routes/angular.js:38:10\n    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)\n    at /juice-shop/node_modules/express/lib/router/index.js:266:9\n    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:320:12)\n    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)\n    at /juice-shop/build/routes/verify.js:171:5\n    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)\n    at /juice-shop/node_modules/express/lib/router/index.js:266:9\n    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:320:12)\n    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)\n    at /juice-shop/build/routes/verify.js:105:5\n    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)\n    at /juice-shop/node_modules/express/lib/router/index.js:266:9\n    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:320:12)\n    at /juice-shop/build/routes/verify.js:345:12\n    at n

```

5. Risk Analysis

Risk analysis focuses on evaluating the likelihood of exploitation and the potential severity of impact associated with the vulnerabilities identified in OWASP Juice Shop. The assessment considers factors such as ease of exploitation, attacker skill requirements, exposure of vulnerable components, and the potential damage to confidentiality, integrity, and availability. Although OWASP Juice Shop is a deliberately vulnerable application used for learning purposes, the risks discussed here closely mirror those present in real-world web applications with similar security flaws.

1) Broken Access Control

Broken Access Control represents a high-risk vulnerability as it enables unauthorized users to access restricted resources or administrative functionalities. Attackers can exploit such flaws with minimal technical effort by directly manipulating URLs or API endpoints. The likelihood of exploitation is high, especially in applications with publicly accessible endpoints. If left unaddressed, this vulnerability can result in privilege escalation and full compromise of application integrity.

2) Broken Authentication

Broken Authentication poses a high security risk due to weak protections against brute-force and credential-stuffing attacks. Attackers can automate login attempts without triggering account lockouts or alerts. The absence of rate limiting and secondary verification significantly increases the probability of successful exploitation. This vulnerability often leads to unauthorized account access and identity compromise.

3) Sensitive Data Exposure

Sensitive Data Exposure carries a high-risk impact because it involves direct leakage of confidential information. Improper access control, insecure APIs, or unencrypted data transmission can expose personal and financial data to attackers. Such vulnerabilities are particularly damaging due to regulatory and legal implications. Exploitation can result in long-term reputational and financial consequences for organizations.

4) CAPTCHA Bypass

CAPTCHA Bypass is typically considered a medium-risk vulnerability, but its impact increases when combined with other weaknesses. Attackers can automate form submissions, brute-force attacks, or abuse application functionality at scale. The ease of bypass significantly lowers the barrier for automated exploitation. Over time, this can lead to service abuse and increased attack surface.

5) Security Misconfiguration

Security Misconfiguration represents a medium-risk vulnerability that aids attackers during reconnaissance and attack planning. Exposed error messages, debug information, or misconfigured headers reveal internal application details. While not always directly exploitable, such information greatly increases the effectiveness of targeted attacks. When chained with other vulnerabilities, misconfigurations can amplify overall security risk.

6. Impact

1) Broken Access Control

If exploited, Broken Access Control can have a severe impact on application security by allowing unauthorized users to access sensitive or administrative resources. Attackers may escalate privileges, manipulate application data, or gain control over restricted functionalities. Such incidents can compromise data integrity and confidentiality. In real-world environments, this often leads to large-scale data exposure and operational disruption.

2) Broken Authentication

Broken Authentication can result in direct account compromise through brute-force or credential-stuffing attacks. Once an attacker gains access to user or administrative accounts, they can perform unauthorized actions, access sensitive information, or further escalate privileges. The impact extends beyond individual users, as compromised accounts can be used to attack other parts of the system. This vulnerability can severely undermine user trust and system reliability.

3) Sensitive Data Exposure

Sensitive Data Exposure has a high impact due to the potential leakage of personal, financial, or confidential information. Exploitation can lead to identity theft, financial fraud, and violations of data protection regulations. The long-term consequences often include reputational damage and legal penalties. Such breaches can affect both users and organizations on a large scale.

4) CAPTCHA Bypass

CAPTCHA Bypass can lead to widespread abuse of application functionality when exploited at scale. Attackers can automate form submissions, spam services, or assist brute-force authentication attacks. Although it may not directly compromise sensitive data, the resulting abuse can degrade system performance and increase security exposure. The impact becomes more significant when combined with authentication or access control weaknesses.

5) Security Misconfiguration

Security Misconfiguration can have a moderate but compounding impact by exposing internal application details to attackers. Information disclosure through misconfigured headers or verbose errors can aid in crafting more effective attacks. While the immediate damage may be limited, such misconfigurations often act as enablers for more serious vulnerabilities. Over time, this can lead to deeper system compromise.

7. Mitigations

1) Broken Access Control

Broken Access Control can be mitigated by enforcing server-side authorization checks for every request, regardless of client-side restrictions. Role-Based Access Control (RBAC) should be implemented to ensure users can only access resources permitted to their role. Sensitive endpoints must not be accessible through direct URL manipulation. Regular access control testing and code reviews should be conducted to prevent privilege escalation issues.

2) Broken Authentication

To mitigate Broken Authentication, applications should implement rate limiting, account lockout mechanisms, and CAPTCHA enforcement on login attempts. Strong password policies and secure session management practices must be followed. Multi-factor authentication (MFA) should be enabled for sensitive or administrative accounts. Authentication logic should be regularly tested against brute-force and credential-stuffing attacks.

3) Sensitive Data Exposure

Sensitive Data Exposure can be prevented by ensuring strong encryption for data both at rest and in transit. Access to sensitive APIs and files should be restricted through proper authentication and authorization controls. Sensitive information must never be stored or transmitted in plain text. Regular audits should be performed to identify and eliminate unnecessary exposure of confidential data.

4) CAPTCHA Bypass

CAPTCHA Bypass can be mitigated by implementing server-side CAPTCHA validation rather than relying on client-side checks. CAPTCHA challenges should be dynamically generated and tied to user sessions. Additional protections such as rate limiting and behavior-based detection can further reduce automated abuse. CAPTCHA effectiveness should be periodically tested to ensure it cannot be bypassed through request manipulation.

5) Security Misconfiguration

Security Misconfiguration can be reduced by disabling debug modes, removing verbose error messages, and hiding framework or server details from responses. Proper security headers should be configured to strengthen application defenses. Unused services, endpoints, and default configurations should be removed before deployment. Regular configuration reviews and security audits are essential to maintain a hardened application environment.

8. Real World Scenarios

1) Broken Access Control

In many real-world applications, broken access control has allowed regular users to access administrative panels simply by modifying URLs or API endpoints. Attackers often discover hidden endpoints through browser tools or API inspection and exploit missing server-side authorization checks. Such incidents have resulted in unauthorized data access, privilege escalation, and manipulation of application settings. These cases show how relying on client-side restrictions alone can lead to full system compromise.

2) Broken Authentication

Broken authentication has frequently been exploited in environments where login protections are weak or missing. In several real incidents, attackers used automated scripts to perform credential-stuffing attacks using leaked password databases. Without rate limiting or account lockout mechanisms, attackers successfully gained access to multiple user accounts. These breaches often went undetected until users reported suspicious activity, highlighting the risk of inadequate authentication controls.

3) Sensitive Data Exposure

Sensitive data exposure has caused major incidents where confidential user information was unintentionally made publicly accessible through APIs or exposed files. Attackers commonly scan applications for unsecured endpoints that return personal or financial data without proper authorization. In many cases, organizations only became aware of the breach after the data was indexed or shared publicly. Such incidents underline the importance of strict access control and secure data handling practices.

4) CAPTCHA Bypass

CAPTCHA bypass vulnerabilities have been exploited in real-world applications to automate abuse at scale. Attackers have bypassed weak CAPTCHA implementations to submit spam, brute-force login forms, or abuse feedback and registration features. Because the CAPTCHA validation was performed only on the client side, automated requests easily bypassed protection mechanisms. These incidents often led to service degradation and increased exposure to further attacks.

5) Security Misconfiguration

Security misconfiguration has played a critical role in many real-world breaches by exposing sensitive system information. Verbose error messages, debug interfaces, and exposed framework details have provided attackers with valuable insights into application architecture. In multiple cases, such information was used to plan targeted attacks or chain vulnerabilities together. These incidents demonstrate how even minor configuration oversights can significantly increase attack success.

10. Conclusion

The comprehensive security assessment of OWASP Juice Shop has provided deep insights into a broad spectrum of vulnerabilities that closely mirror those frequently encountered in modern, real-world web applications. This evaluation clearly illustrates how insecure design decisions, weak authentication mechanisms, improper access control enforcement, and misconfigurations can collectively expose applications to serious security risks. Vulnerabilities such as Broken Access Control, Broken Authentication, and Sensitive Data Exposure represent high-impact threats that, if exploited in a production environment, could result in unauthorized access to restricted resources, leakage of confidential information, and complete compromise of application integrity. These findings strongly reinforce the need for organizations to prioritize security from the early stages of application development rather than addressing issues only after exploitation occurs.

In addition to high-severity vulnerabilities, the assessment also highlighted several medium-risk issues, including CAPTCHA Bypass and Security Misconfiguration. While these vulnerabilities may appear less critical when evaluated in isolation, they can significantly increase overall risk when combined with other weaknesses. Attackers often exploit such issues to automate attacks, gather internal application details, or bypass protective mechanisms, which can then be leveraged to execute more severe exploits. This demonstrates how seemingly minor misconfigurations or validation flaws can play a crucial role in chained attacks, ultimately amplifying their impact on application security.

The findings from this assessment further emphasize the interconnected nature of modern web applications, particularly those built using API-driven architectures and client-side frameworks. Weaknesses in authentication logic, authorization checks, or configuration management can have cascading effects across multiple components of the application. Understanding how data flows between frontend interfaces, backend services, and exposed APIs is therefore essential for effective security planning. The OWASP Juice Shop assessment serves as a practical demonstration of how attackers analyze application behavior and exploit trust boundaries when security controls are inconsistently applied.

Finally, this report highlights that securing web applications is not a one-time activity or a checklist-based exercise. Instead, application security must be treated as a continuous process embedded throughout the Software Development Life Cycle (SDLC). Regular code reviews, secure coding standards, automated and manual security testing, and ongoing security awareness training for developers and administrators are essential to maintaining a strong security posture. By fostering a security-first culture and implementing layered defense strategies, organizations can significantly reduce their attack surface, protect sensitive data, and ensure long-term resilience against evolving cyber threats.