# Penetration Testing Report

Student Name: Carter Wright
Course: ITT-340 – Cybersecurity and Ethical Hacking
Date: 04-26-2026
Instructor(s): Ajay (AJ) Joshi, Joe Urbaszewski

## OWASP Juice Shop Assessment

Topic 8: Benchmark - Penetration Assessment

GRAND CANYON UNIVERSITY

# 1.0 Executive Summary

This report documents a comprehensive penetration test conducted against the OWASP Juice Shop web application. The objective of the assessment was to identify, exploit, and analyze security vulnerabilities using a combination of automated tools and manual testing techniques.

The assessment revealed multiple critical and moderate vulnerabilities, including:

- Broken Authentication (weak password enforcement)
- Broken Anti-Automation (CAPTCHA bypass)
- Unvalidated Redirects
- Improper Input Validation
- Sensitive endpoint exposure

Exploitation of these vulnerabilities demonstrated that attackers could bypass authentication controls, automate restricted actions, access sensitive data, and manipulate application behavior. These findings highlight systemic weaknesses in core security areas such as authentication, input validation, access control, and secure configuration.

Overall, the application presents a **Moderate to High risk level**, primarily due to insufficient server-side validation, weak authentication controls, and lack of layered security defenses.

This assessment reinforces the importance of implementing secure development practices and proactive security testing throughout the software development lifecycle.

# Table of Contents

# 2.0 Scope and Methodology

## 2.1 Scope

The scope of this penetration test included:

- OWASP Juice Shop hosted at:
  https://gcu-cce-js.ctfd.io/

Testing was performed in a controlled lab environment using authorized access through the GCU virtual lab.

---

## 2.2 Methodology

The penetration test followed a structured methodology based on industry standards:

1. Reconnaissance
2. Scanning and Enumeration
3. Vulnerability Identification
4. Exploitation
5. Post-Exploitation Analysis
6. Risk Assessment

---

## 2.3 Tools Used

- Kali Linux
- **Burp Suite Community Edition**
- **OWASP ZAP**
- Browser Developer Tools
- Nmap (basic recon)
- Manual testing techniques

# 3.0 Phase Testing Details

## 3.1 Reconnaissance

Reconnaissance was conducted to gather information about the target application and identify potential entry points.

- Identified application structure and endpoints using browser developer tools
- Observed API calls and routes via Burp Suite proxy
- Discovered hidden endpoints and resources through manual exploration
- Identified challenge-based structure of Juice Shop

**Tools Used:**

- Browser DevTools
- Burp Suite Proxy

📌 *(see figure 1 + figure 2)*

---

## 3.2 Scanning and Enumeration

Automated scanning was performed to identify vulnerabilities and misconfigurations.

- OWASP ZAP automated scan detected:
  - Cloud metadata exposure risk
  - Missing security headers
  - Potential sensitive data exposure
- Burp Suite captured HTTP requests and parameters
- Identified API endpoints such as:
  - /rest/admin
  - /api/Products
  - /rest/user/login

📌 *(see figure 3 + figure 4)*

---

## 3.3 Exploitation

Manual and automated techniques were used to exploit identified vulnerabilities.

Exploitation included:

- Authentication bypass (Admin login)
- Cross-Site Scripting (DOM XSS)

- Accessing restricted endpoints (Admin Section)
- CAPTCHA bypass
- Viewing sensitive data (Confidential Document, Metrics)

Each vulnerability is detailed in Section 4.0.

---

### 3.4 Post-Exploitation

After gaining elevated access, further actions were performed:

- Accessed admin-only functionality
- Retrieved sensitive documents
- Viewed internal application metrics
- Manipulated application logic via intercepted requests

These actions demonstrate how an attacker could escalate privileges and maintain access.

---

### 3.5 Risk Analysis and Impact

The vulnerabilities identified pose significant risks:

- **Confidentiality Risk:** Exposure of sensitive data and documents
- **Integrity Risk:** Ability to manipulate application behavior
- **Availability Risk:** Potential for abuse via automation (CAPTCHA bypass)

Overall Risk Level: **Moderate to High**

## 4.0 Detailed Findings and Recommendations

---

### 4.1 Login Admin (Authentication Bypass)

**Description**

The application is vulnerable to authentication bypass through improper input validation in the login functionality. By intercepting and modifying the login request, it is possible to inject malicious input into the authentication query, allowing unauthorized access to an administrative account.

**Impact**

This vulnerability allows attackers to:

- Gain **unauthorized administrative access**
- Bypass all authentication controls

- Access sensitive data and restricted functionality
- Perform actions with elevated privileges

This represents a **Critical risk** to the application.

## Tools Used

- Burp Suite Community Edition
- Browser (Firefox)

## Steps to Reproduce

1. Navigate to the Juice Shop login page *(see Figure 5)*
2. Enter any credentials and submit the login form
3. Intercept the request using Burp Suite Proxy
4. Send the request to **Repeater**
5. Modify the request body:

```
{
  "email": "' OR 1=1--",
  "password": "testpassword"
}
```

6. Send the modified request *(see Figure 6)*
7. Observe the response containing authentication token
8. Return to the application and confirm admin access *(see Figure 7)*

## Remediation

To mitigate this vulnerability:

- Implement **parameterized queries (prepared statements)**
- Enforce strict **server-side input validation**
- Use **ORM frameworks** to prevent SQL injection
- Implement **authentication hardening** (e.g., MFA)
- Sanitize and validate all user inputs

---

## 4.2 DOM XSS

**Description**

The application is vulnerable to DOM-based Cross-Site Scripting (XSS) through the search functionality. User input is dynamically processed and rendered in the browser without proper sanitization or encoding, allowing execution of malicious JavaScript code.

**Impact**

This vulnerability allows attackers to:

- Execute arbitrary JavaScript in a victim's browser
- Steal session tokens or cookies
- Perform actions on behalf of the user
- Deface or manipulate application content

This represents a **High risk** vulnerability.

**Tools Used**

- Web Browser (Firefox)

**Steps to Reproduce**

1. Navigate to the Juice Shop homepage *(see Figure 8)*
2. Locate the search bar at the top of the page
3. Enter the following payload into the search field:

```
<iframe src="javascript:alert('xss')">
```

4. Submit the search query
5. Observe that the payload is reflected in the page *(see Figure 9)*
6. A JavaScript alert box appears, confirming execution *(see Figure 10)*

**Remediation**

To mitigate this vulnerability:

- Implement **proper output encoding** for all user inputs
- Use **input validation and sanitization**
- Apply **Content Security Policy (CSP)** headers
- Avoid directly inserting user input into the DOM
- Use secure frameworks that automatically escape output

## 4.3 Confidential Document

**Description**

The application exposes sensitive internal endpoints that allow unauthorized users to access confidential information. By analyzing network traffic and intercepting requests, it is possible to identify and directly interact with hidden API endpoints that return protected data.

**Impact**

This vulnerability allows attackers to:

- Access **confidential internal documents**
- Discover hidden application functionality
- Enumerate sensitive endpoints
- Potentially escalate further attacks

This represents a **High risk** vulnerability due to improper access control.

**Tools Used**

- Browser Developer Tools (Network Tab)
- Burp Suite Community Edition

**Steps to Reproduce**

1. Navigate to the Juice Shop application
2. Open browser Developer Tools and go to the **Network tab** *(see Figure 11)*
3. Interact with the application to generate traffic
4. Observe API endpoints being called, including hidden or internal routes
5. Identify a suspicious or sensitive endpoint
6. Intercept the request using Burp Suite
7. Analyze the response returned from the server *(see Figure 12)*
8. Confirm that confidential information is exposed without authentication

**Remediation**

To mitigate this vulnerability:

- Enforce **proper access control checks** on all endpoints
- Restrict access to sensitive APIs using **authentication and authorization**
- Avoid exposing internal endpoints to the client

- Implement **role-based access control (RBAC)**
- Validate all requests server-side before returning sensitive data

---

## 4.4 Admin Section

**Description**

The application fails to properly enforce access control restrictions on administrative functionality. After bypassing authentication, it is possible to directly access the administration panel, which contains sensitive data and privileged operations.

**Impact**

This vulnerability allows attackers to:

- Access **restricted administrative features**
- View sensitive user data and feedback
- Modify or delete application data
- Perform actions with full administrative privileges

This represents a **Critical risk** due to complete privilege escalation.

**Tools Used**

- Web Browser (Firefox)
- Burp Suite Community Edition

**Steps to Reproduce**

1. Perform authentication bypass using SQL injection *(see Figure 13)*
2. Log in as an administrator
3. Navigate to the account menu in the top-right corner
4. Select the administration option or manually access:

```
/#/administration
```

5. Observe that the administration panel loads without restriction *(see Figure 14)*
6. Confirm access to sensitive data such as registered users and customer feedback

**Remediation**

To mitigate this vulnerability:

- Enforce **strict role-based access control (RBAC)**
- Validate user roles on the **server side** for every request
- Restrict direct access to sensitive routes
- Implement proper **session validation and authorization checks**
- Log and monitor unauthorized access attempts

---

## 4.5 View Basket

**Description**

The application is vulnerable to an Insecure Direct Object Reference (IDOR) in the basket functionality. By intercepting and modifying requests, it is possible to access another user's basket without proper authorization checks.

**Impact**

This vulnerability allows attackers to:

- View other users' basket contents
- Access sensitive user data
- Modify or manipulate another user's cart
- Potentially interfere with transactions

This represents a **High risk** vulnerability due to broken access control.

**Tools Used**

- Burp Suite Community Edition
- Web Browser (Firefox)

**Steps to Reproduce**

1. Add an item to your basket *(see Figure 15)*
2. Intercept the request using Burp Suite Proxy
3. Identify the request to the basket endpoint *(see Figure 16)*
4. Send the request to **Repeater**
5. Modify the user or basket identifier in the request

6. Forward the modified request
7. Observe that another user's basket is returned *(see Figure 17)*
8.

**Remediation**

To mitigate this vulnerability:

- Implement **server-side authorization checks** for all object access
- Ensure users can only access resources they own
- Use **indirect references** instead of exposing direct object IDs
- Validate user identity against requested resources
- Apply **access control enforcement at every request**

---

## 4.6 Exposed Metrics

**Description**

The application exposes an internal metrics endpoint that provides detailed system and performance information. This endpoint is publicly accessible without authentication, allowing unauthorized users to view sensitive operational data.

**Impact**

This vulnerability allows attackers to:

- Gather **internal system information**
- Identify application behavior and performance metrics
- Discover potential attack vectors
- Aid in further exploitation through reconnaissance

This represents a **Medium to High risk** vulnerability due to sensitive information exposure.

**Tools Used**

- Web Browser (Firefox)
- Manual testing techniques

**Steps to Reproduce**

1. Navigate to the Juice Shop application
2. Manually access the following endpoint:

```
/metrics
```

3. Observe that the endpoint loads without authentication
4. Review the exposed data, including system metrics and performance information *(see Figure 18)*
5. Confirm that the challenge is completed *(see Figure 19)*

**Remediation**

To mitigate this vulnerability:

- Restrict access to metrics endpoints using **authentication and authorization**
- Disable public exposure of internal monitoring endpoints
- Implement **network-level access controls** (e.g., IP restrictions)
- Use secure monitoring tools that require authentication
- Avoid exposing sensitive system data in production environments

---

## 4.7 Outdated Allowlist

**Description**

The application contains an outdated allowlist used to validate redirect destinations. By analyzing the client-side source code, it is possible to identify redirect logic that fails to properly restrict external URLs, allowing users to be redirected to unauthorized destinations.

**Impact**

This vulnerability allows attackers to:

- Redirect users to **malicious external websites**
- Conduct phishing attacks
- Trick users into interacting with untrusted content
- Damage user trust and application integrity

This represents a **Medium risk** vulnerability with potential for social engineering attacks.

**Tools Used**

- Browser Developer Tools (Debugger / Sources)
- Web Browser (Firefox)

**Steps to Reproduce**

1. Open browser Developer Tools and navigate to the **Sources tab**
2. Search for keywords such as redirect in the application code *(see Figure 20)*
3. Identify redirect logic and allowlist validation
4. Modify or craft a URL containing a redirect parameter
5. Navigate to the crafted URL
6. Observe that the application redirects to an external cryptocurrency address *(see Figure 21)*
7. Confirm successful exploitation *(see Figure 22)*

**Remediation**

To mitigate this vulnerability:

- Maintain a **strict and updated allowlist** of approved redirect URLs
- Validate all redirect destinations on the **server side**
- Avoid using user-controlled input for redirects
- Implement **URL validation and normalization**
- Use indirect references or internal routing mechanisms

---

## 4.8 Password Strength

**Description**

The application allows authentication using weak and easily guessable credentials. The administrator account password does not meet standard complexity requirements and can be compromised without advanced attack techniques such as SQL injection or brute force.

**Impact**

This vulnerability allows attackers to:

- Gain **unauthorized access** to privileged accounts
- Escalate privileges to administrative level
- Access sensitive data and system functionality

- Compromise the entire application

This represents a **High risk** vulnerability due to direct account compromise.

## Tools Used

- Web Browser (Firefox)

## Steps to Reproduce

1. Navigate to the login page of the application
2. Enter the administrator email:
    - admin@juice-sh.op
3. Enter a weak password:
    - admin123 *(or similar simple password)*
4. Click **Log In**
5. Observe that authentication is successful *(see Figure 23)*
6. Confirm challenge completion *(see Figure 24)*

## Remediation

To mitigate this vulnerability:

- Enforce **strong password policies**:
    - Minimum length (12+ characters)
    - Uppercase, lowercase, numbers, special characters
- Implement **password complexity validation**
- Use **password hashing (bcrypt/argon2)**
- Enforce **account lockout** after failed attempts
- Implement **multi-factor authentication (MFA)**

---

## 4.9 Error Handling

**Description**

The application fails to properly handle errors and exposes internal system messages to the user. When invalid input is submitted, the system returns raw error output ([object Object]) instead of a user-friendly message.

This indicates that backend errors are not properly sanitized before being displayed in the frontend.

**Impact**

This vulnerability can allow attackers to:

- Gain insight into **internal application logic**
- Identify **frameworks, objects, or backend structures**
- Use error messages to assist in **further attacks (e.g., injection, enumeration)**

This is considered a **Medium risk** vulnerability.

**Tools Used**

- Web Browser (Firefox)

**Steps to Reproduce**

1. Navigate to the login page
2. Enter invalid or malformed input into the email field (e.g., random characters or unexpected values)
3. Enter any password
4. Click **Log In**
5. Observe the error message displayed as [object Object] *(see Figure 25)*
6. Confirm challenge completion *(see Figure 26)*

**Remediation**

To fix this vulnerability:

- Implement **proper error handling mechanisms**
- Return **generic user-friendly error messages**
- Avoid exposing internal objects or stack traces
- Log detailed errors **server-side only**
- Use centralized error handling middleware

---

## 4.10 CAPTCHA Bypass

**Description**

The application implements a CAPTCHA mechanism to prevent automated submissions. However, this protection can be bypassed by intercepting and replaying requests using Burp Suite. The server does not properly validate CAPTCHA attempts or enforce rate limiting, allowing multiple submissions in rapid succession.

**Impact**

This vulnerability allows attackers to:

- Automate large numbers of requests
- Perform spam or abuse actions (e.g., feedback flooding)
- Bypass bot protection mechanisms
- Potentially execute brute force or enumeration attacks

This is considered a **Medium to High risk** vulnerability.

**Tools Used**

- Burp Suite (Proxy + Intruder)

**Steps to Reproduce**

1. Navigate to the **Customer Feedback** form *(see Figure 27)*
2. Enter sample data and submit the form
3. Capture the request using **Burp Proxy Intercept** *(see Figure 28)*
4. Send the request to **Burp Intruder**
5. Configure payload positions for submission fields
6. Add multiple payload values to simulate repeated submissions *(see Figure 29)*
7. Launch the Intruder attack to send multiple requests rapidly
8. Observe successful submissions without solving CAPTCHA each time
9. Confirm challenge completion *(see Figure 30)*

**Remediation**

To prevent this vulnerability:

- Implement **server-side CAPTCHA validation**
- Enforce **rate limiting / request throttling**
- Use **session-based CAPTCHA tokens**
- Apply **bot detection mechanisms** (e.g., reCAPTCHA v3)
- Monitor for abnormal request patterns

## 5.0 Security Design Principles

The vulnerabilities found in this test connect directly to some basic security design principles that weren't followed or were only partially implemented:

- **Least Privilege**
  The app didn't properly limit what users could access based on their role. It was possible to reach admin features without the right permissions, which shows role checks weren't enforced.
- **Defense in Depth**
  There weren't enough layers of protection. For example, CAPTCHA could be bypassed because there were no backups like rate limiting or proper server-side checks.
- **Input Validation and Output Encoding**
  User input wasn't being validated or cleaned correctly. This led to issues like DOM-based XSS and SQL injection, which are common but serious vulnerabilities.
- **Authentication and Authorization**
  Password rules were weak, and the login system had flaws that made it easier to get unauthorized access. On top of that, authorization checks weren't applied consistently across the app.
- **Secure Defaults**
  Some sensitive endpoints, like /metrics, were exposed without requiring login. The app wasn't set up with secure settings by default, which increases risk.
- **Fail Securely (Error Handling)**
  Error messages exposed internal details like "[object Object]," which can give attackers clues about how the system works behind the scenes.

If these principles were followed more consistently, the app would be a lot harder to attack and overall much more secure.

## 6.0 Conclusion

The OWASP Juice Shop app has a lot of security issues that you would also see in real-world web apps. Using a mix of automated tools and hands-on testing, I found some major problems with login systems, access control, input validation, and overall app logic.

Exploiting these issues showed how easy it can be for an attacker to get into accounts, view sensitive data, and get around basic security checks. The biggest risks came from weak authentication and missing server-side validation, which leave the app wide open.

This test really shows why secure coding matters. Strong input validation, better access controls, and multiple layers of security can make a big difference. It also helps to test for security issues throughout development instead of waiting until the end.

Fixing these problems and following standard security practices would lower the risk a lot and make the app much more secure overall.

## 7.0 References

Grand Canyon University. (n.d.). *ITT-340 Benchmark Assignment: Penetration testing using OWASP Juice Shop*. Halo | Learn. https://halo.gcu.edu/resource/e4581a0a-ba8f-421b-b66f-ee2683d47bdd

Grand Canyon University. (n.d.). *LopesCloud virtual environment platform*. https://lopescloud.gcu.edu/#/student/assignments

Kali Linux. (n.d.). *Kali Linux penetration testing platform*. https://www.kali.org/

Mozilla. (n.d.). *HTTP overview*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTTP

OWASP Foundation. (n.d.). *Authentication Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

OWASP Foundation. (n.d.). *Cross Site Scripting (XSS)*. https://owasp.org/www-community/attacks/xss/

OWASP Foundation. (n.d.). *Cross-Site Request Forgery Prevention Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

OWASP Foundation. (n.d.). *Error Handling Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

OWASP Foundation. (n.d.). *Input Validation Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

OWASP Foundation. (n.d.). *OWASP Juice Shop*. GitHub.

https://github.com/juice-shop/juice-shop

OWASP Foundation. (n.d.). *OWASP Top Ten Web Application Security Risks*.

https://owasp.org/www-project-top-ten/

OWASP Foundation. (n.d.). *OWASP Zed Attack Proxy (ZAP)*.

https://www.zaproxy.org/

OWASP Juice Shop CTF Platform. (n.d.). *Challenges*.

https://gcu-cce-js.ctfd.io/challenges

PortSwigger Ltd. (n.d.). *Burp Suite Community Edition*.

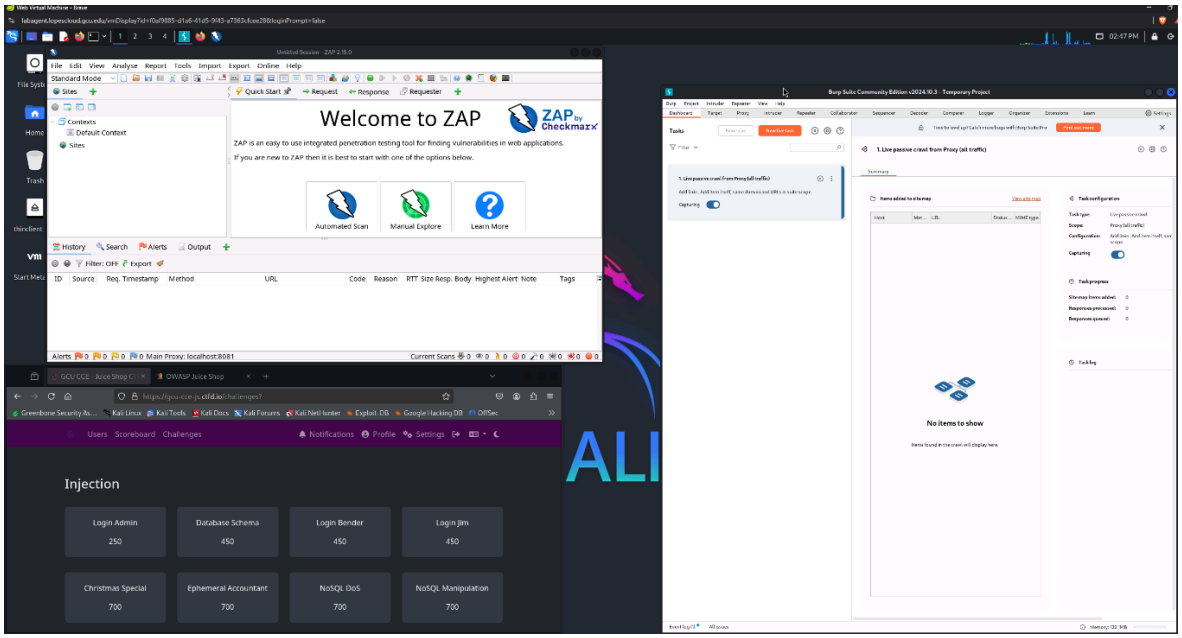
https://portswigger.net/burp

PortSwigger Ltd. (n.d.). *Web security testing guide*.

https://portswigger.net/web-security

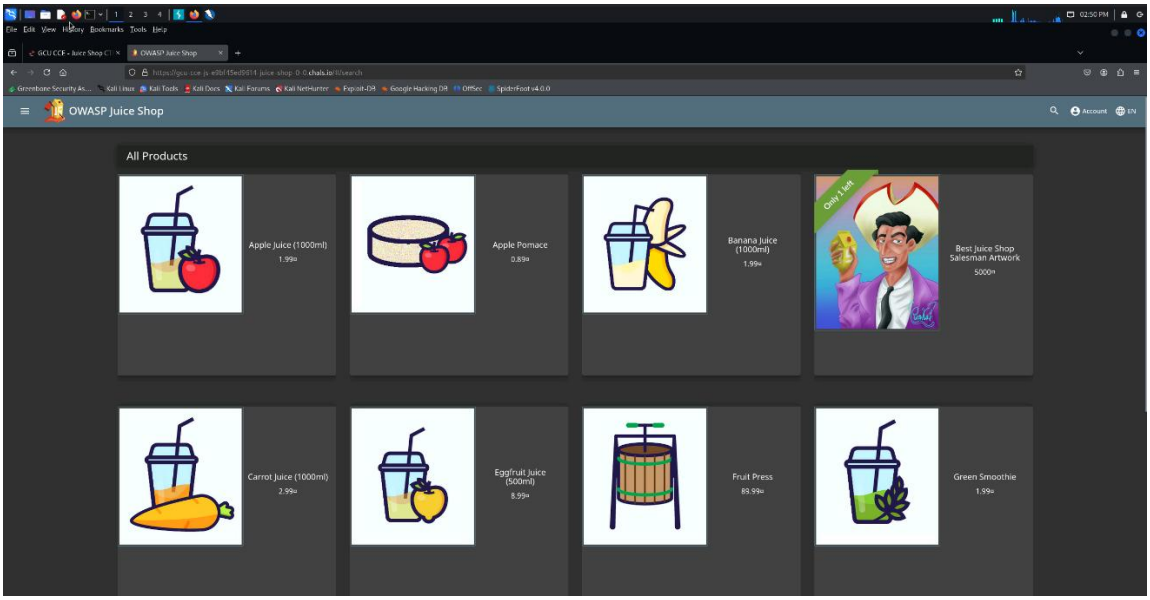
# 8.0 Appendix

**Figure 1. Kali Linux Penetration Testing Environment**



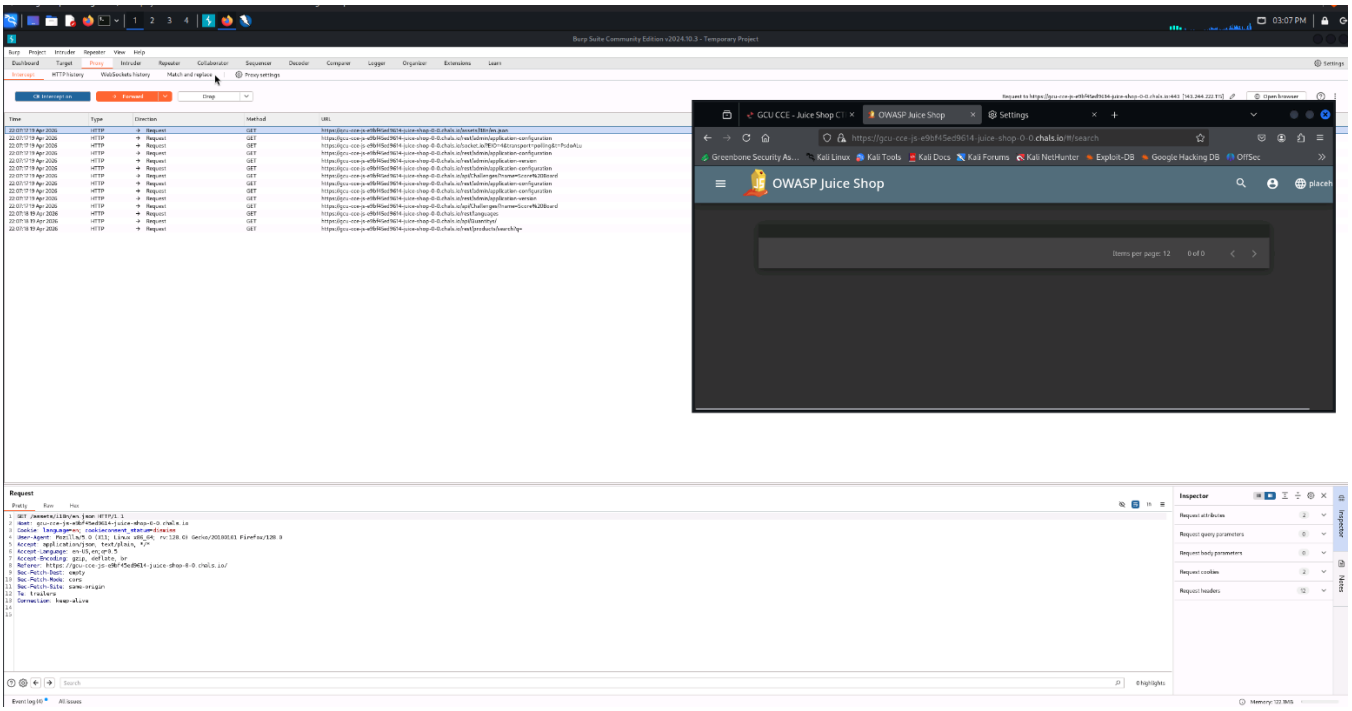

This figure shows the Kali Linux virtual machine used during the assessment. The environment includes tools such as Firefox, Burp Suite, and OWASP ZAP, which were used for reconnaissance, interception, and vulnerability scanning.

**Figure 2. OWASP Juice Shop Target Application Homepage**



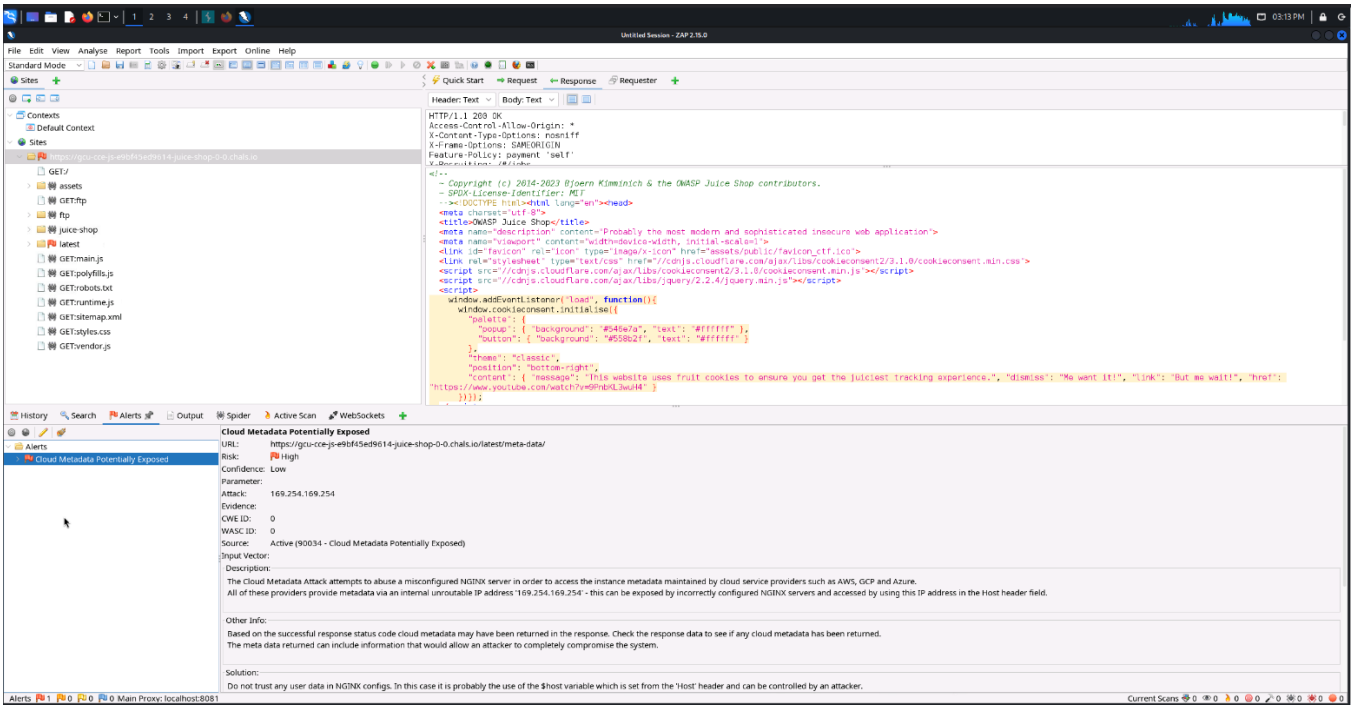

This figure displays the main interface of the OWASP Juice Shop application. It highlights the product listing page, which serves as the primary entry point for user interaction and initial reconnaissance.

**Figure 3. Burp Suite Intercepting Browser Traffic**



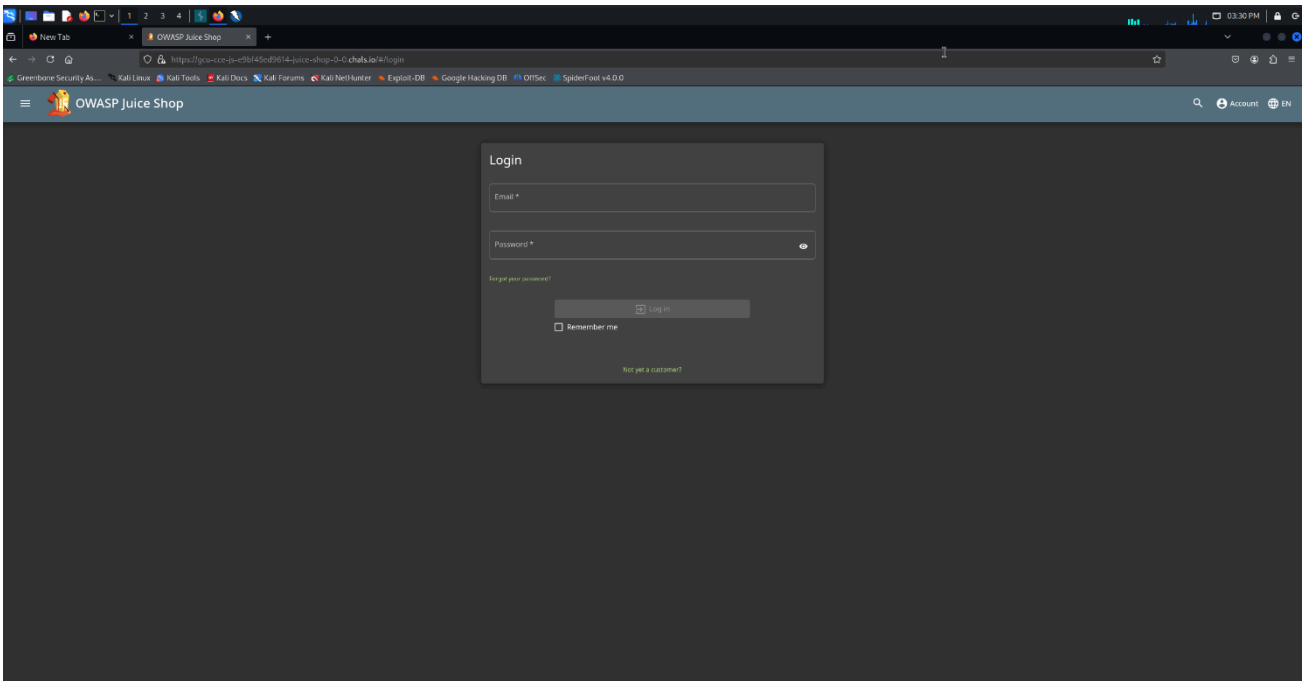

This figure demonstrates Burp Suite capturing HTTP requests between the browser and the target application. It shows intercepted GET requests and headers, which were analyzed and modified during testing to identify vulnerabilities.

**Figure 4. OWASP ZAP Automated Scan Results**



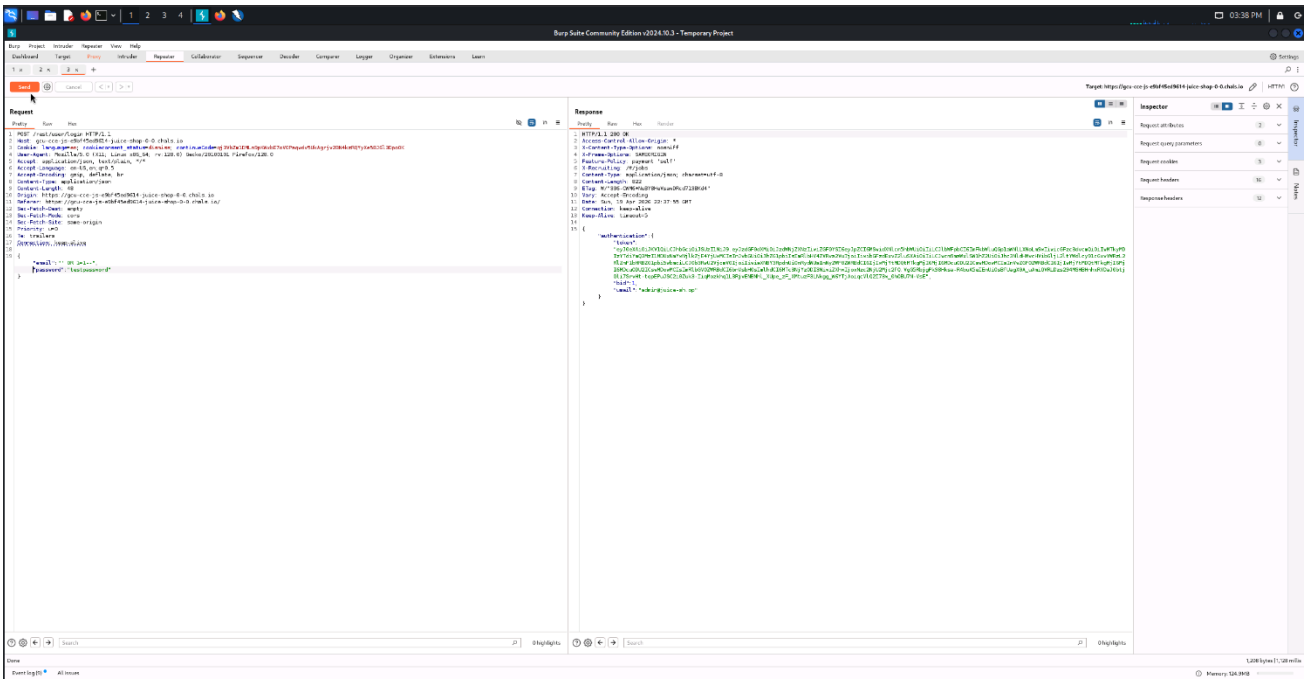

This figure shows the results of an automated vulnerability scan performed using OWASP ZAP. The scan identifies potential security issues, including misconfigurations and exposure risks within the application.

**Figure 5. Juice Shop Login Interface Before Testing**



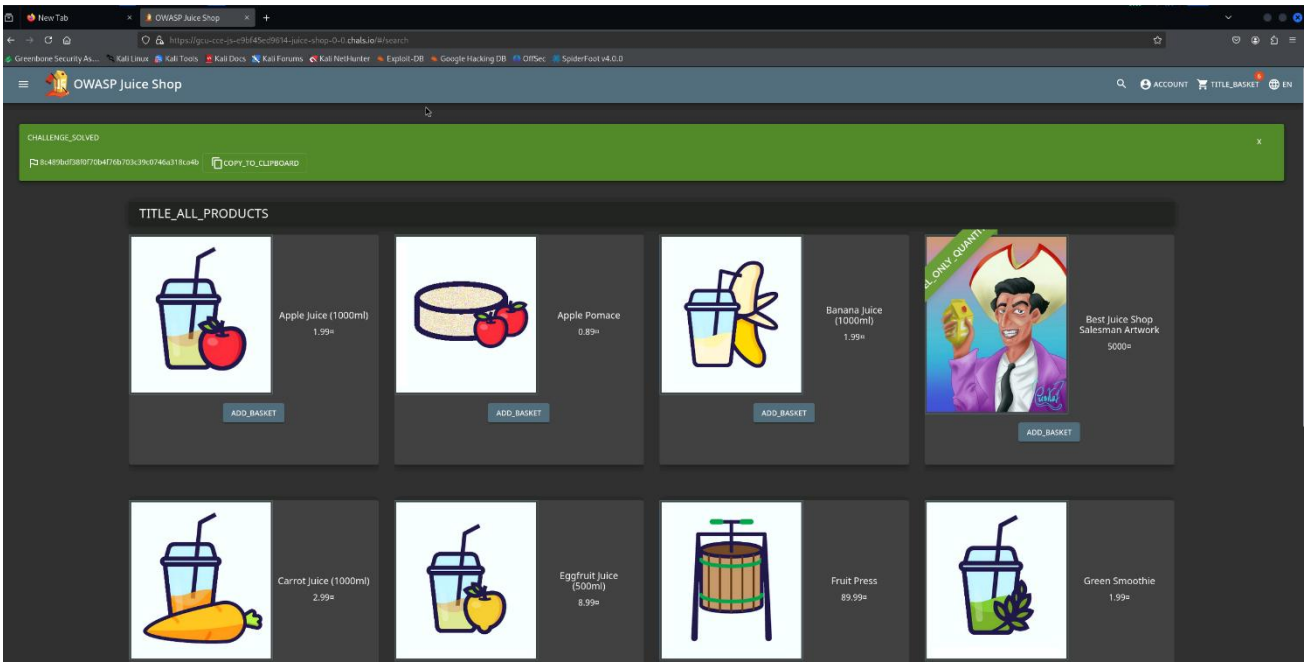

This figure shows the login page of the OWASP Juice Shop application prior to exploitation. The interface requires an email and password, which serves as the entry point for authentication testing.

**Figure 6. Modified Login Request in Burp Suite (SQL Injection Attempt)**



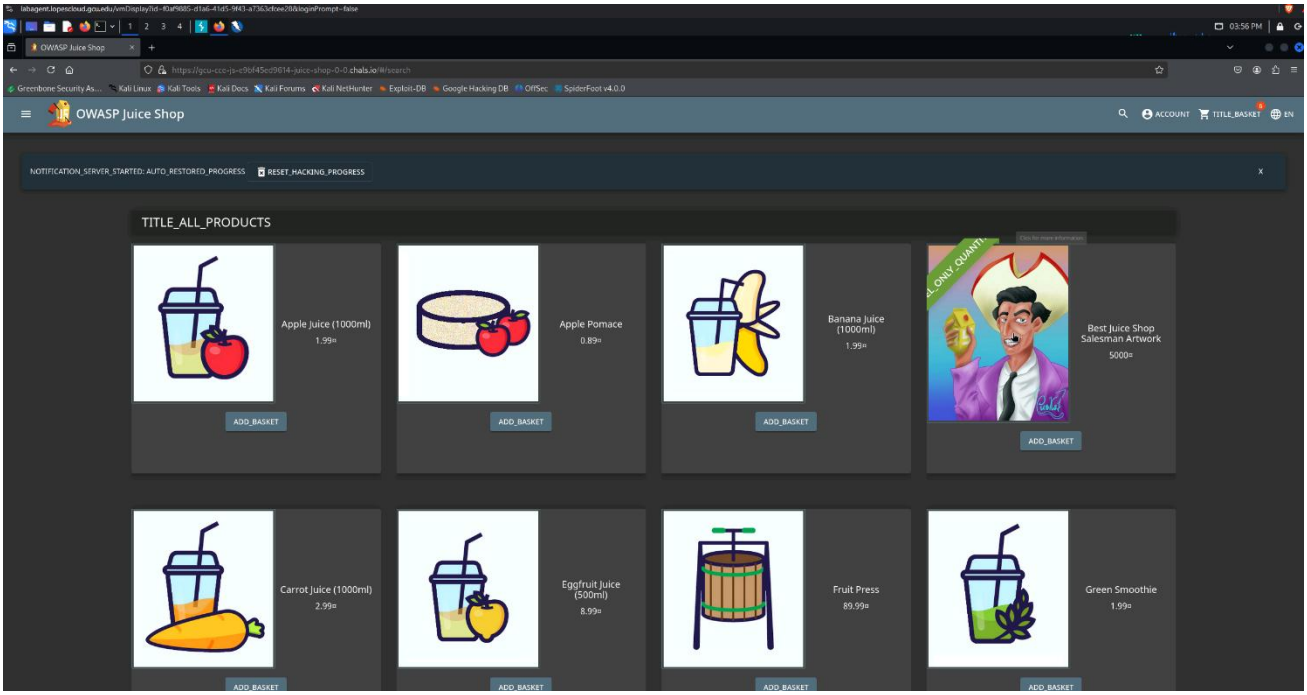
This figure displays a captured and modified HTTP POST request in Burp Suite. The email parameter was altered using an SQL injection payload (' OR 1=1--) to bypass authentication controls.

**Figure 7. Successful Unauthorized Login as Administrator**



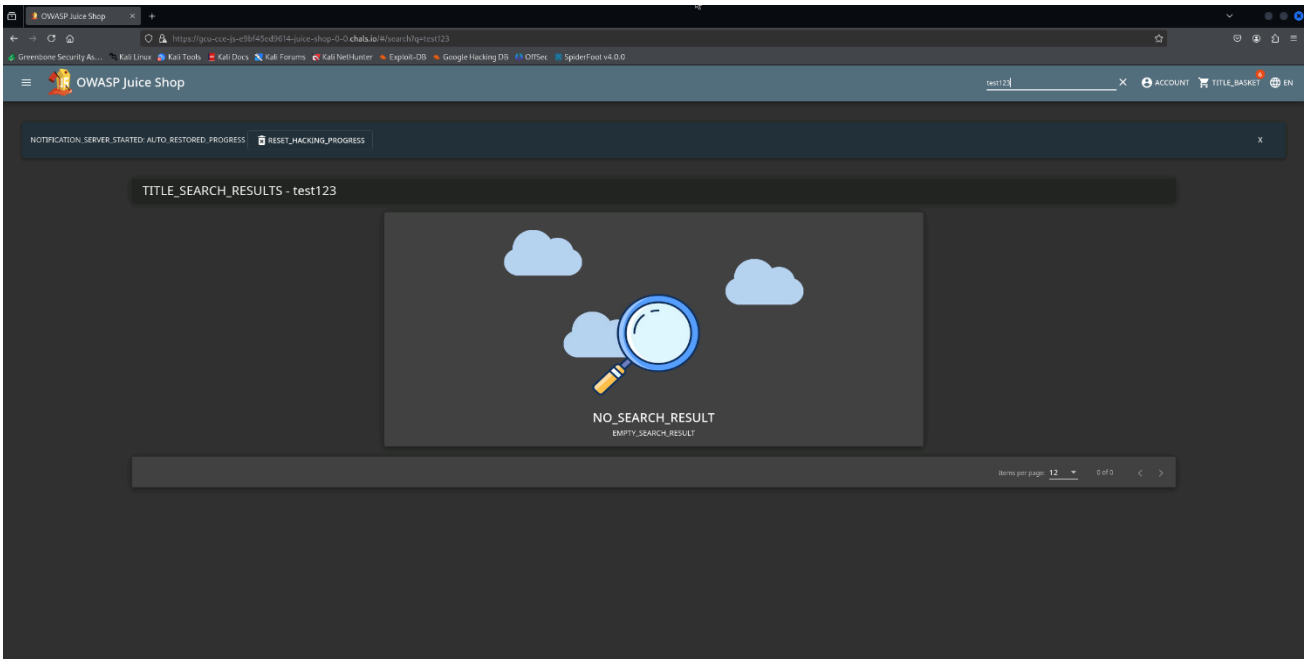

This figure shows the application after successful authentication bypass. The response confirms that administrative access was granted without valid credentials, demonstrating a critical authentication vulnerability.

**Figure 8. Juice Shop Homepage Before XSS Testing**



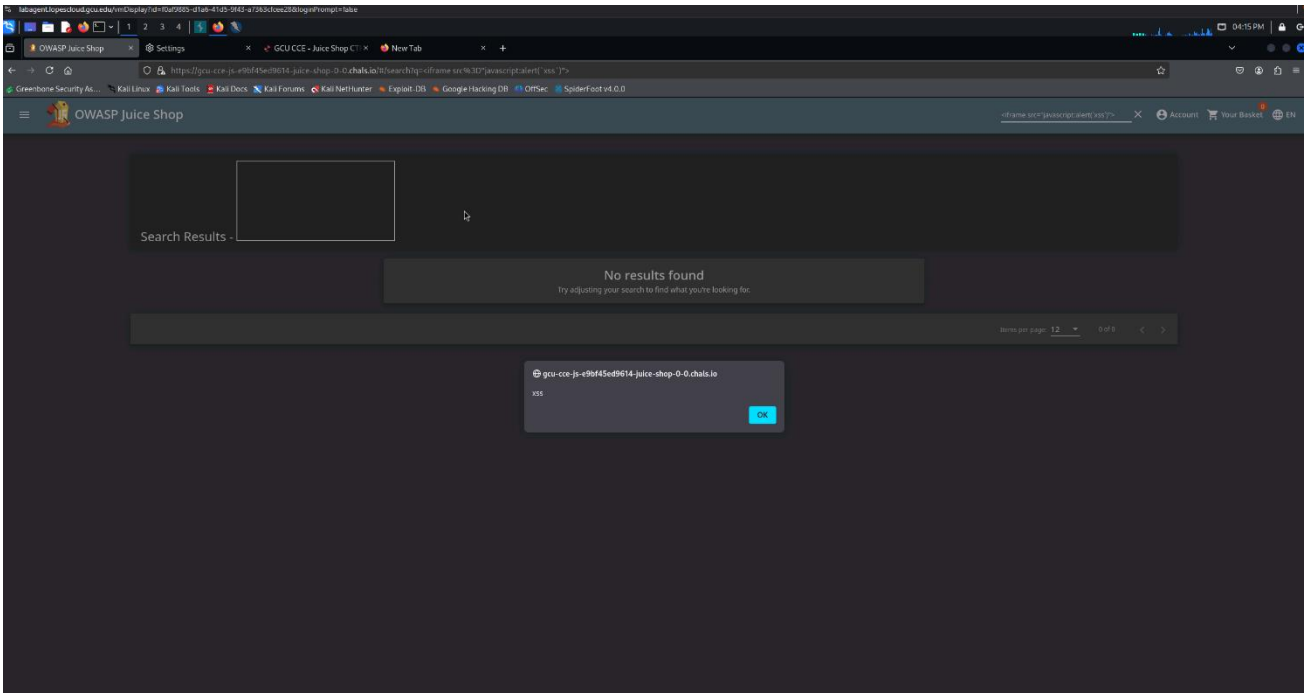

This figure shows the OWASP Juice Shop homepage prior to testing for cross-site scripting vulnerabilities. The search functionality is visible and was identified as a potential input vector.

**Figure 9. User Input Reflected in Search Results**



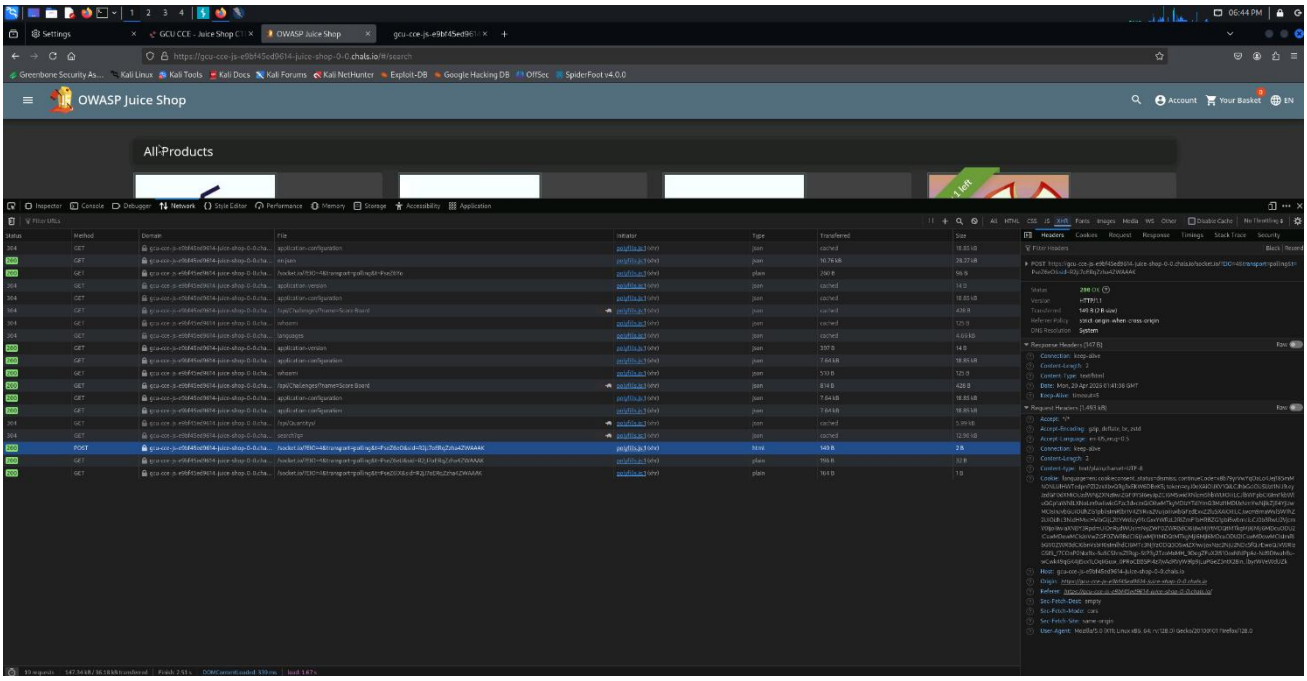

This figure demonstrates how user-supplied input is reflected directly in the search results page. The lack of output sanitization indicates a potential vulnerability to client-side script injection.

**Figure 10. DOM XSS Payload Executed in Browser**



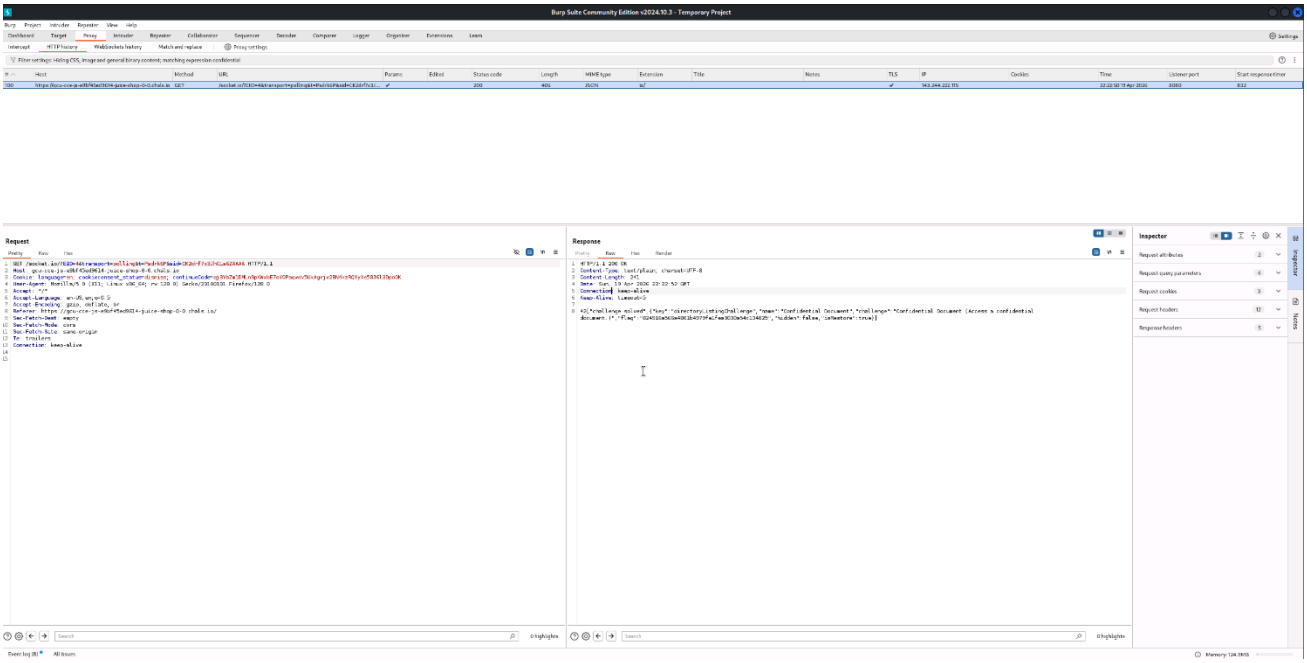

This figure shows the successful execution of a DOM-based XSS payload entered into the search field. The injected JavaScript `(<iframe src="javascript:alert('xss')">)` triggered a browser alert, confirming the vulnerability.

**Figure 11. Network Traffic Revealing Exposed Application Endpoints**



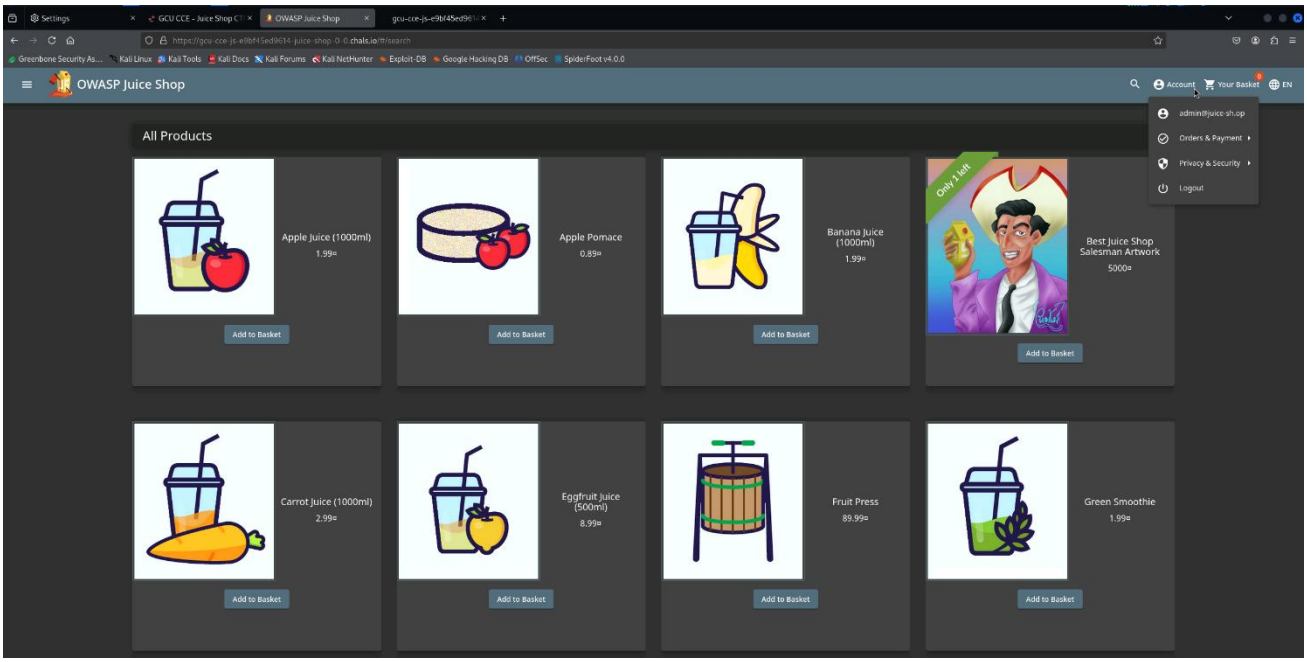

This figure shows browser developer tools capturing network traffic. Multiple internal API endpoints are visible, including those related to application configuration and hidden functionality, indicating potential exposure of sensitive resources.

**Figure 12. Burp Suite Capturing Request to Sensitive Endpoint**



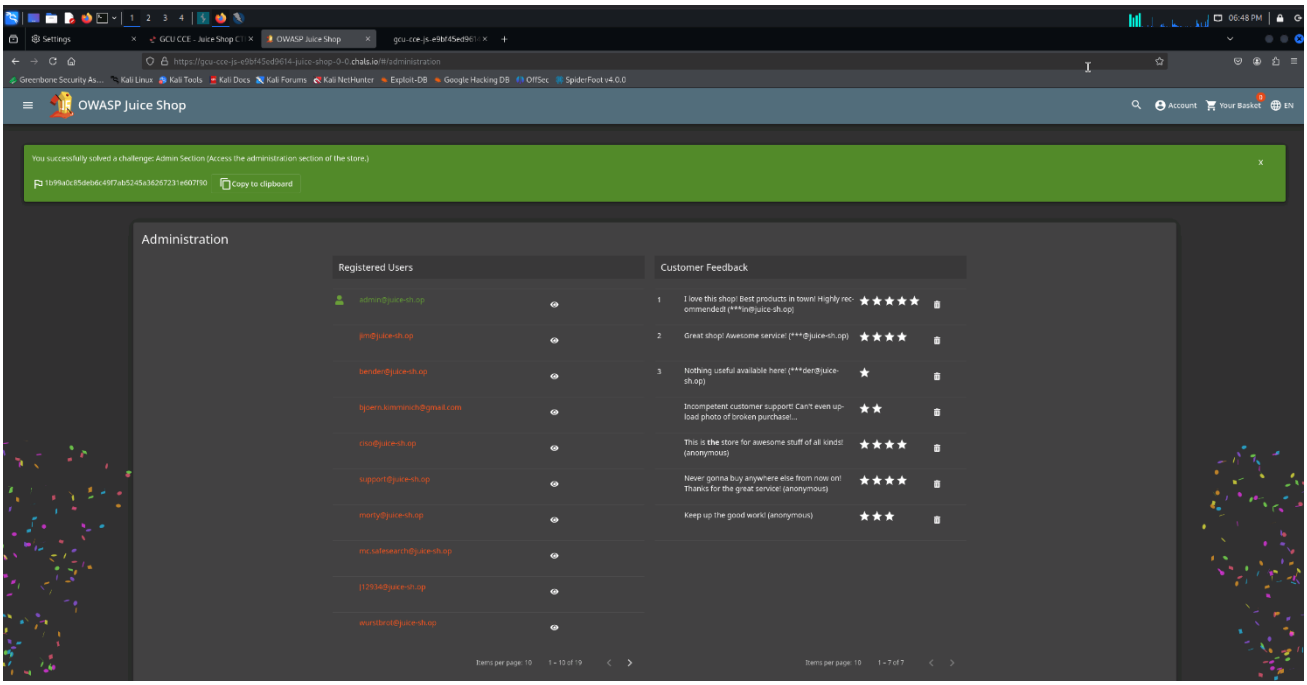

This figure displays a request intercepted in Burp Suite to a backend endpoint associated with the “Confidential Document” challenge. The response reveals sensitive information without requiring proper authentication.

**Figure 13. Successful Authentication Bypass via SQL Injection**



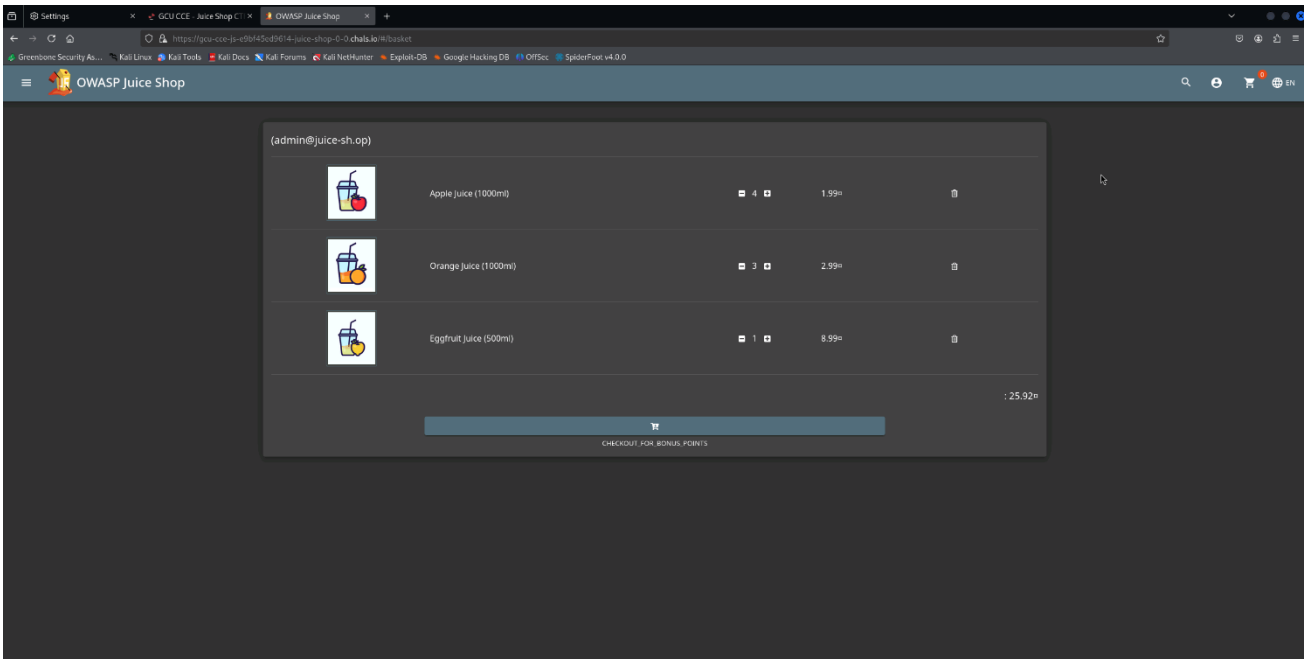

This figure shows the application after a successful login bypass using a crafted SQL injection payload. The user is authenticated as an administrator without valid credentials, confirming a critical authentication flaw.

**Figure 14. Unauthorized Access to Administration Panel**



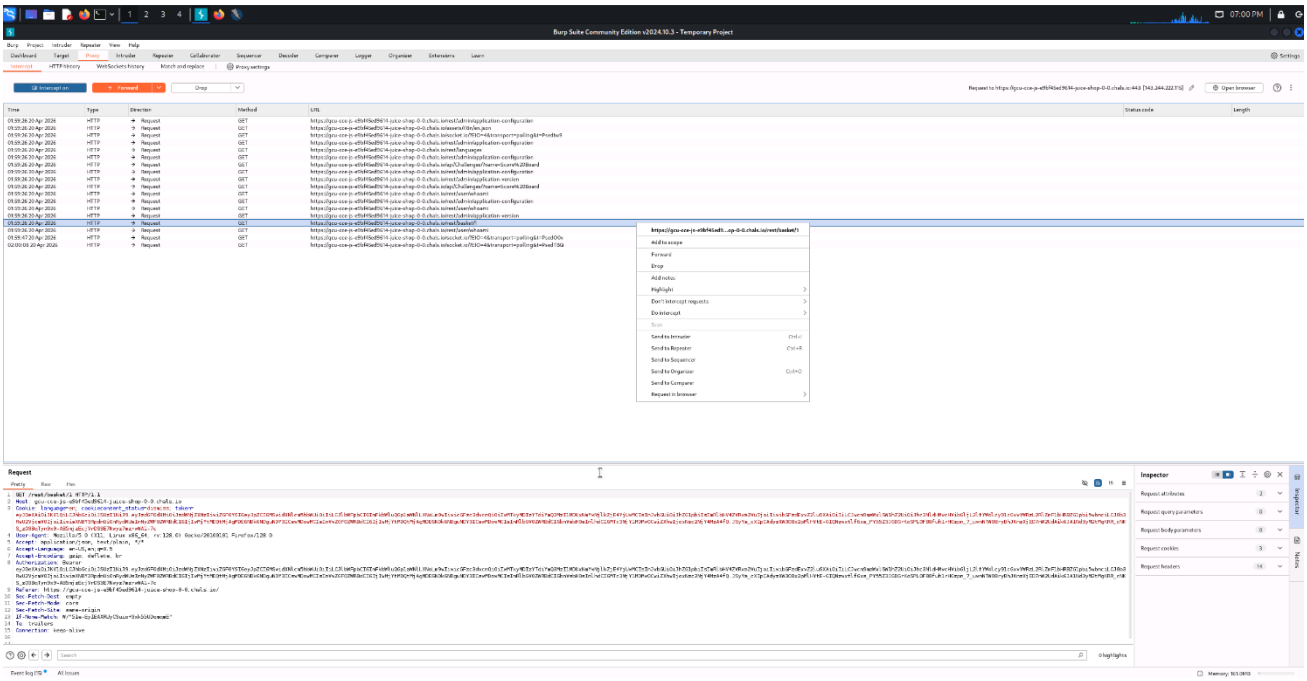

This figure displays the administrative dashboard accessed without proper authorization. Sensitive data, including registered users and customer feedback, is visible, demonstrating the impact of broken access control.

**Figure 15. Adding Item to Basket**



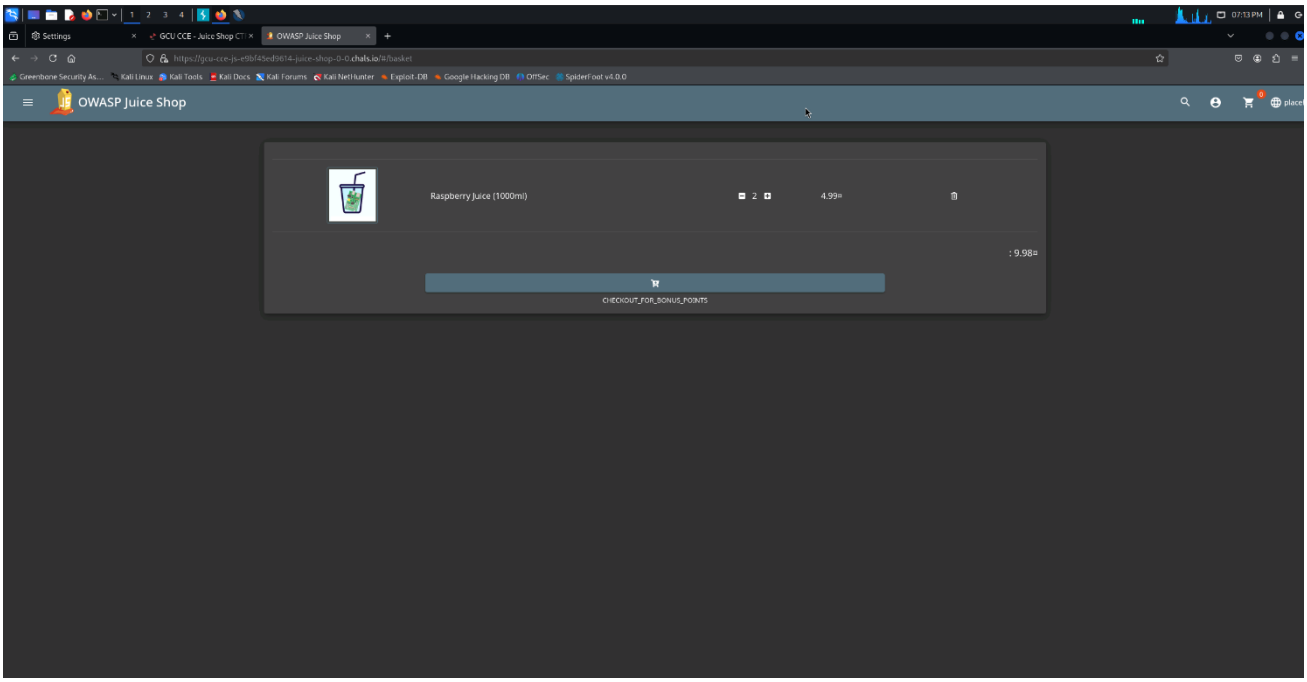

This figure shows a user adding an item to their shopping basket within the application. This action generates a request that can later be intercepted and modified.

**Figure 16. Intercepted Request to User Basket**



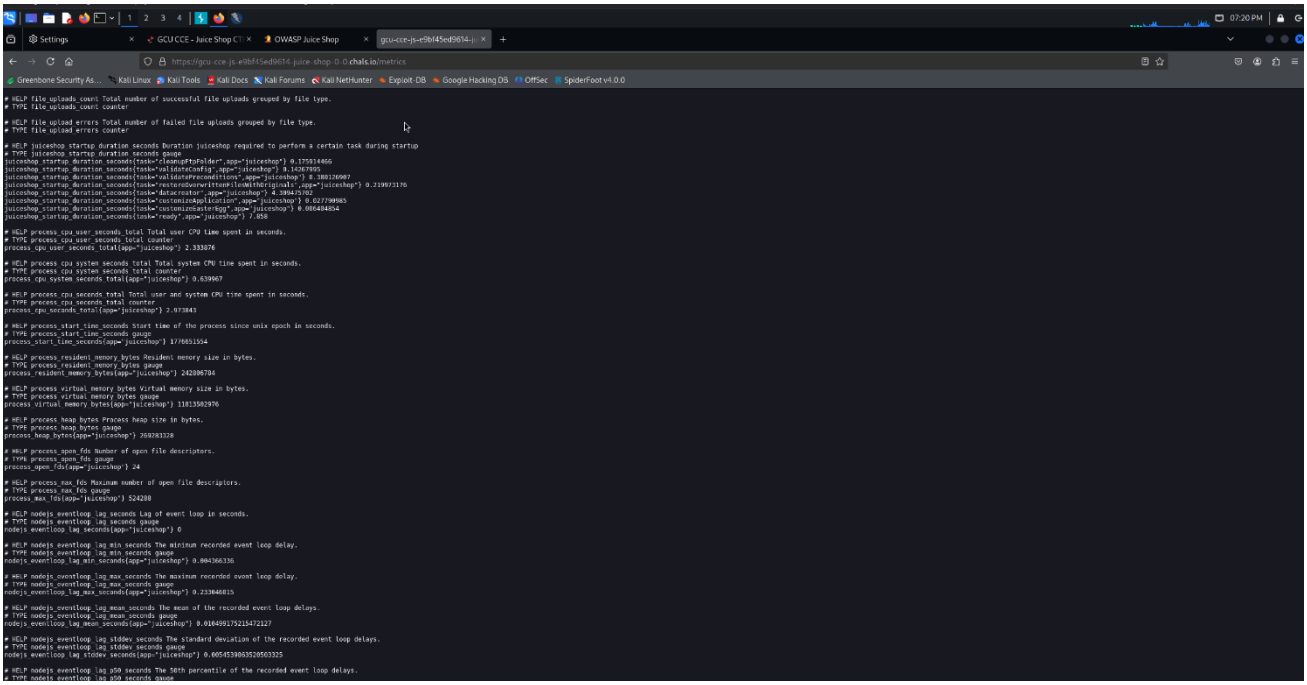

This figure displays a request captured in Burp Suite targeting the basket endpoint. The request includes identifiers that can be manipulated to access other users' data.

**Figure 17. Unauthorized Access to Another User's Basket**



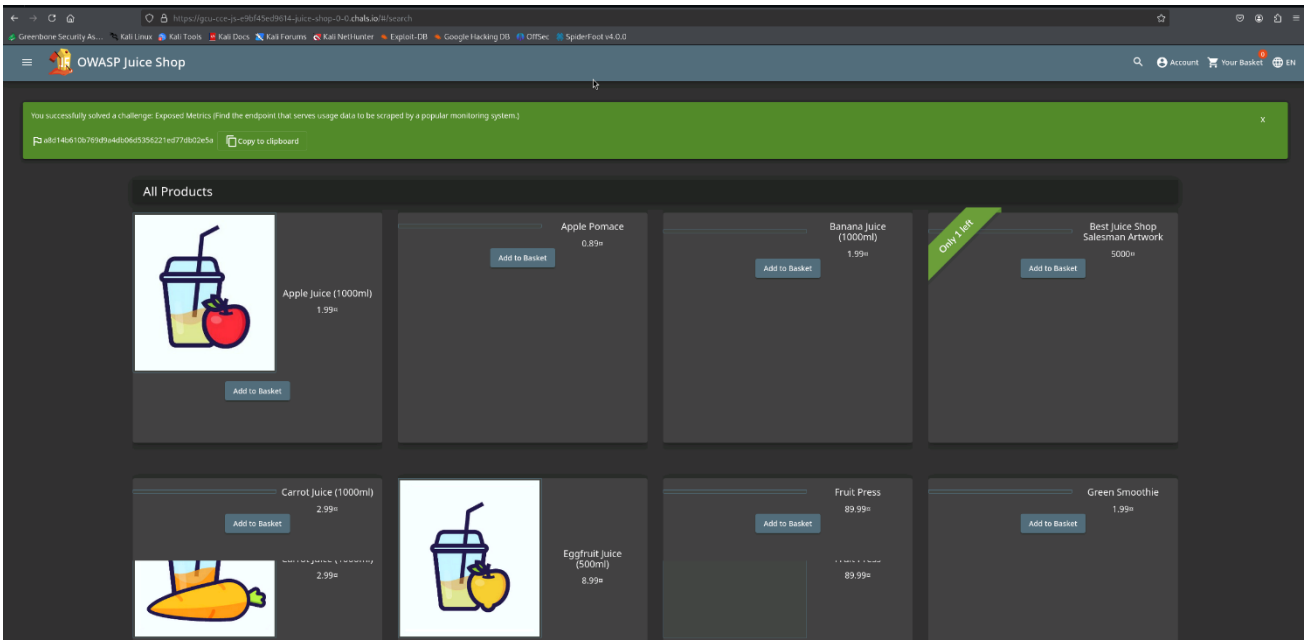

This figure demonstrates successful access to another user's basket by modifying request parameters. The displayed basket contents belong to a different user, confirming a broken access control vulnerability.

**Figure 18. Publicly Accessible Application Metrics Endpoint**



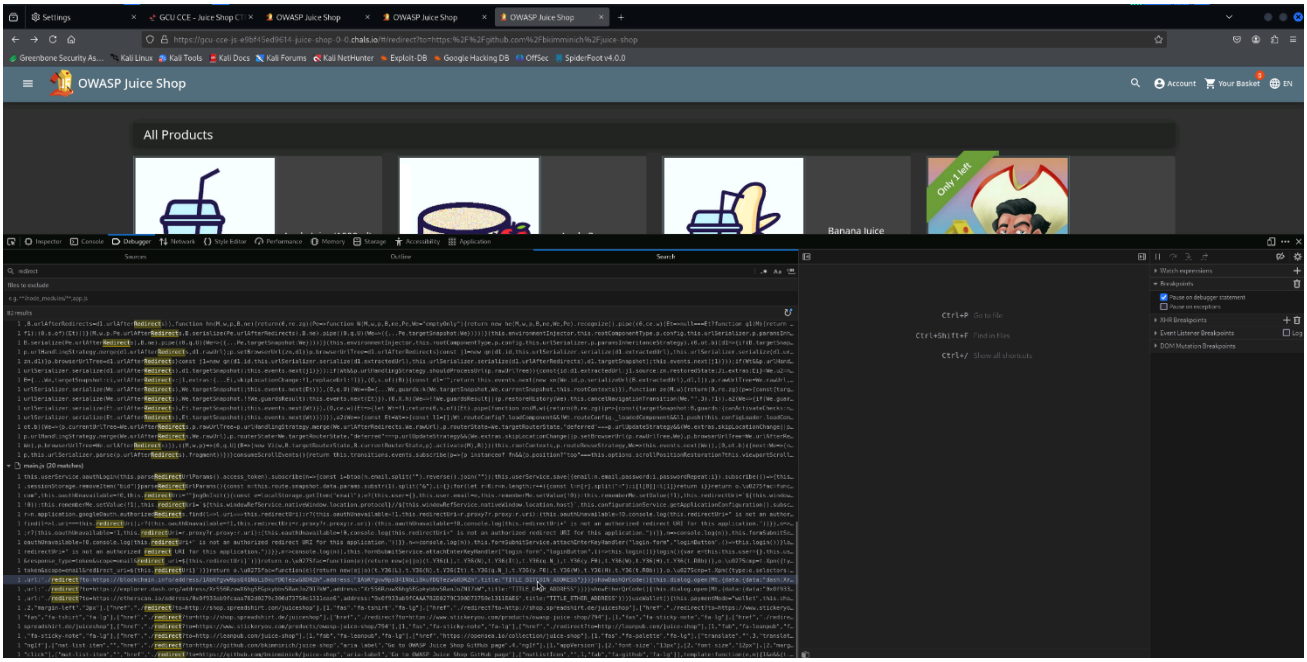
This figure shows the /metrics endpoint being accessed directly through the browser. The endpoint exposes internal application performance and system metrics without requiring authentication.

**Figure 19. Exposed Metrics Challenge Successfully Completed**



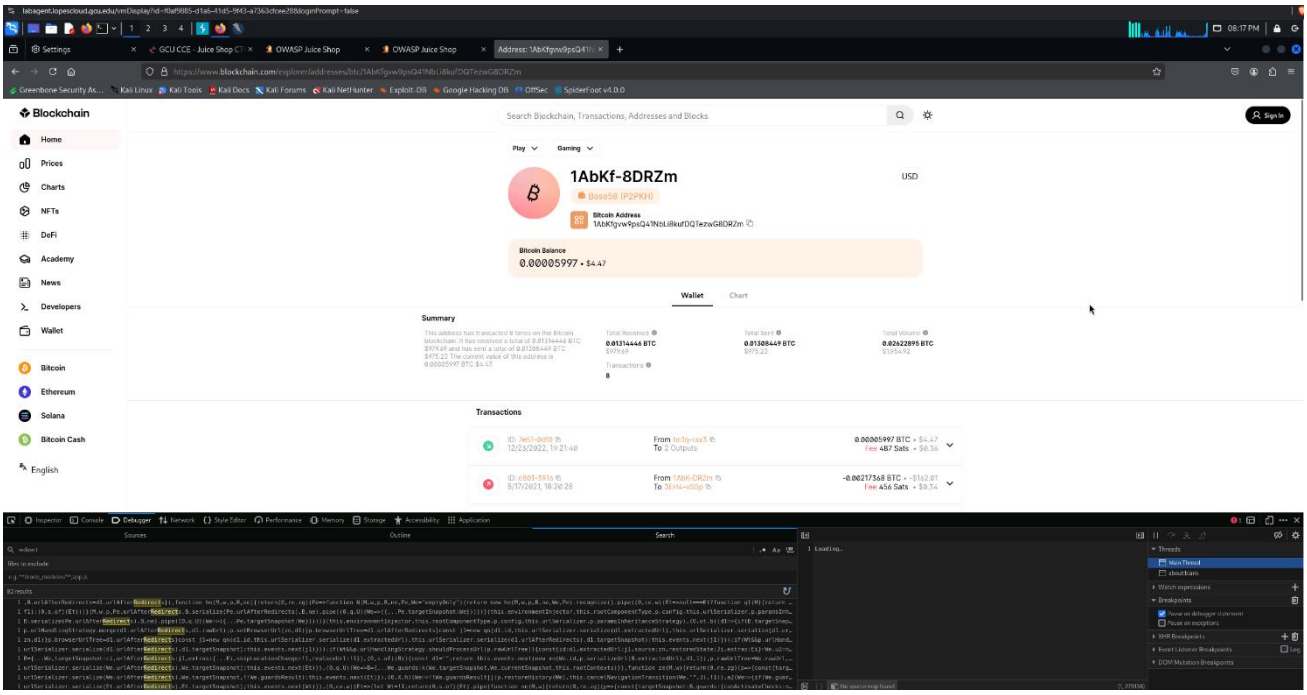

This figure confirms successful discovery and access of the exposed metrics endpoint. The application acknowledges the vulnerability, demonstrating that sensitive monitoring data is publicly accessible.

**Figure 20. Source Code Analysis of Redirect Logic**



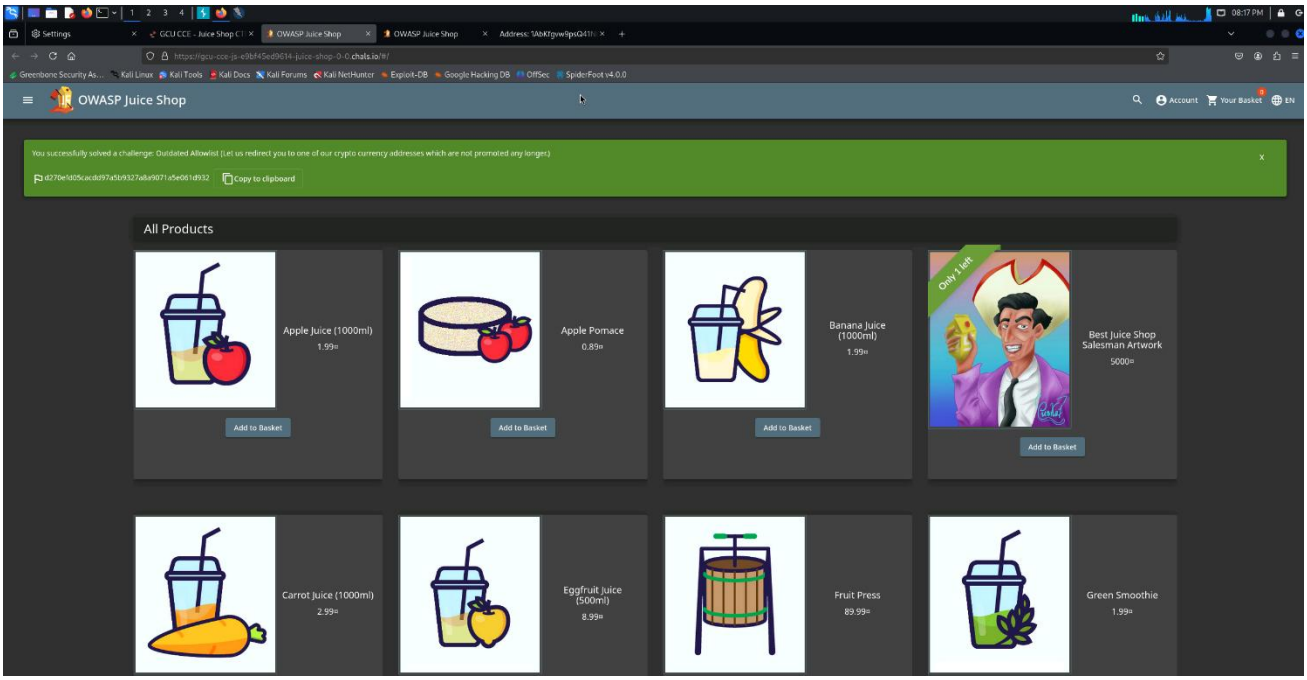

This figure shows the application's client-side source code where redirect logic is implemented. The presence of outdated or improperly validated allowlisted URLs indicates a potential open redirect vulnerability.

**Figure 21. Successful Redirect to Cryptocurrency Address**



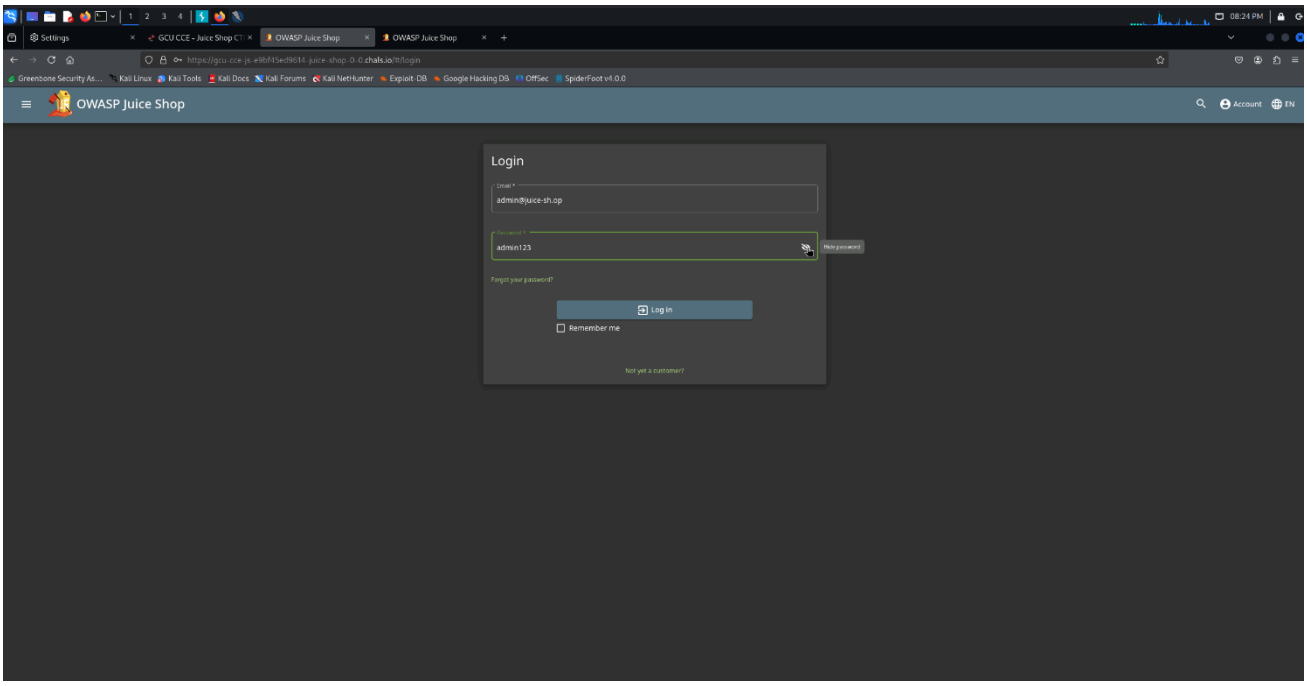

This figure demonstrates a successful redirection to an external cryptocurrency address. The application improperly allows navigation to an unapproved external destination, confirming a failure in allowlist validation.

**Figure 22. Challenge Completion Notification**



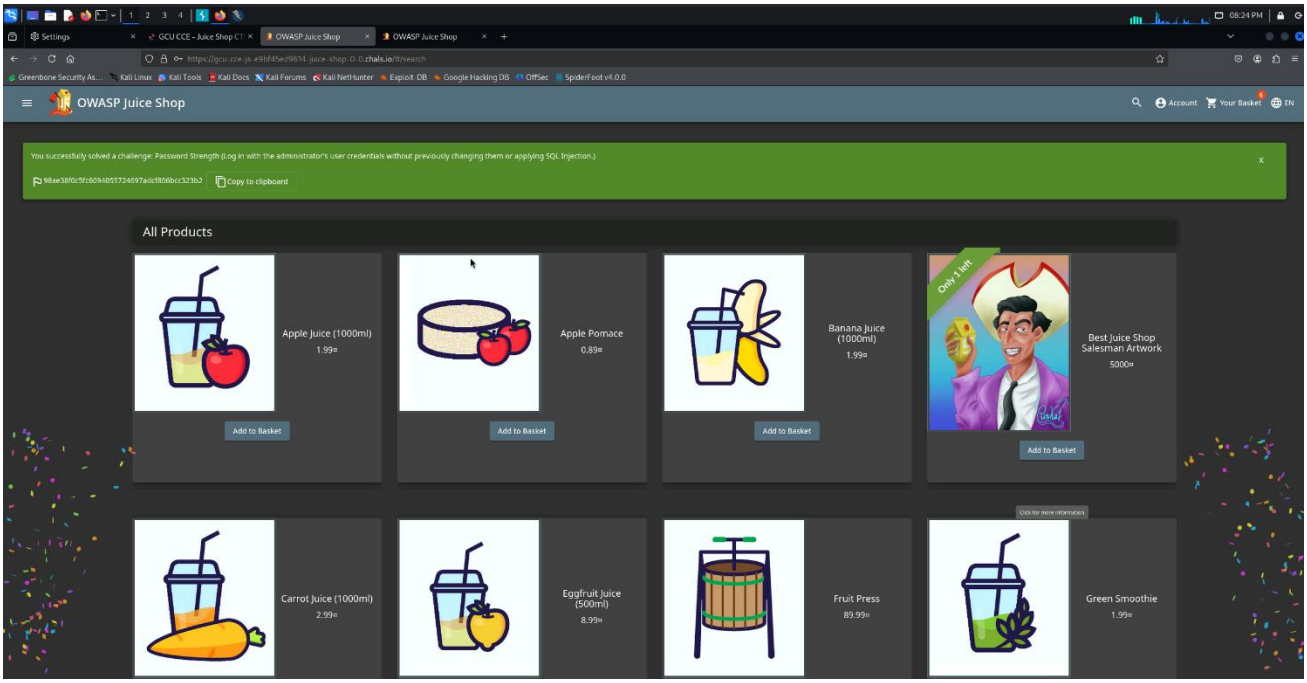

This figure confirms successful exploitation of the outdated allowlist vulnerability. The application acknowledges that a redirect to a non-approved address was achieved.

**Figure 23. Admin Login Using Weak Credentials**



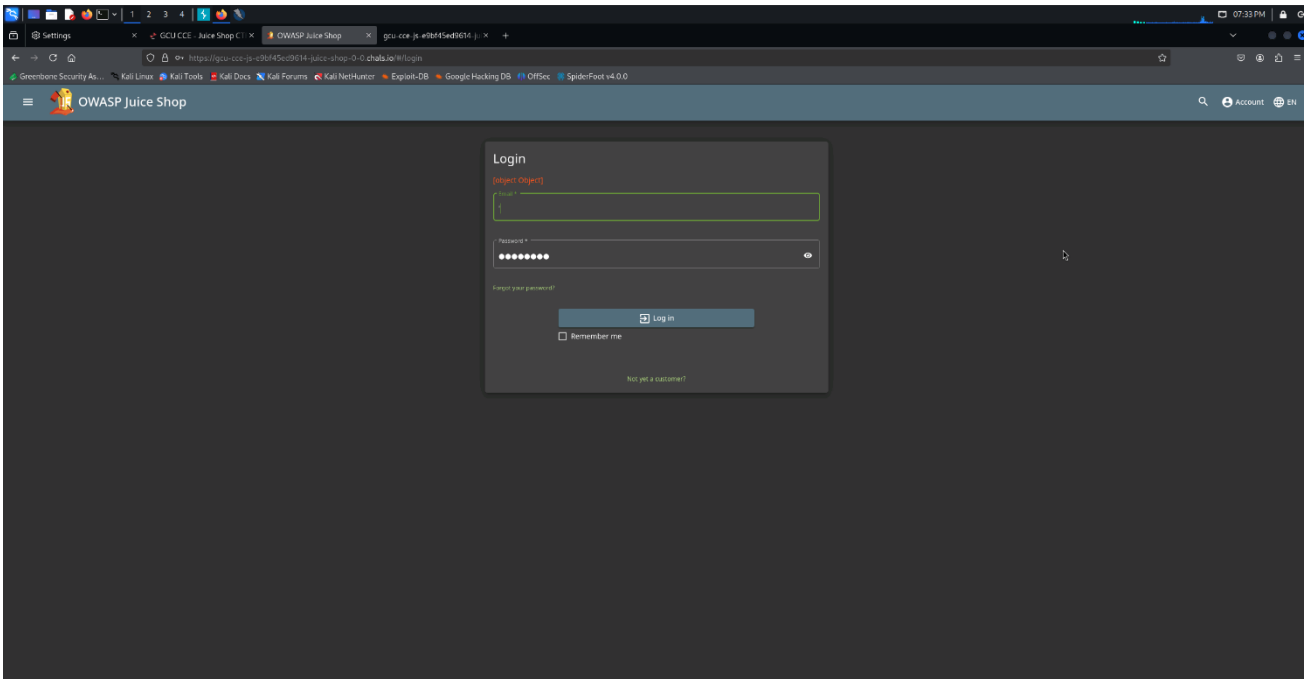

This figure shows the login page where the administrator account is accessed using weak and easily guessable credentials. The password lacks complexity and does not meet secure password standards.

**Figure 24. Successful Authentication and Challenge Completion**



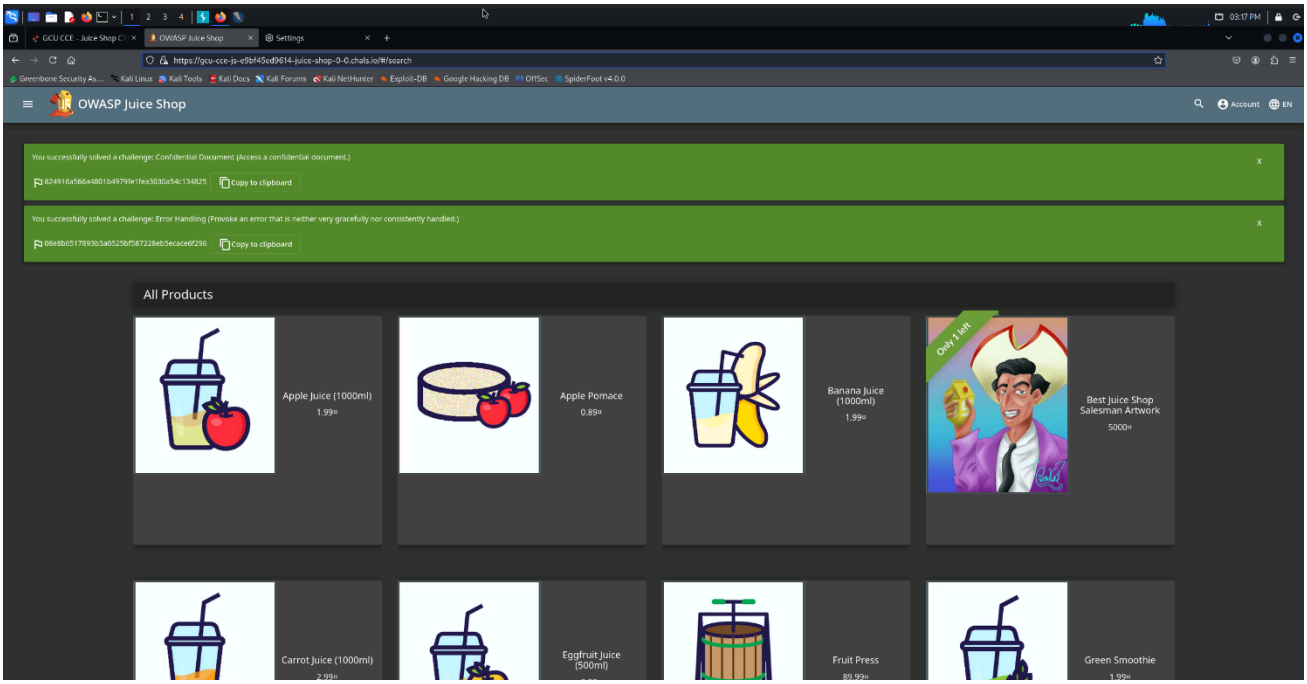

This figure confirms successful login to the administrator account using weak credentials. The application accepts the insecure password and grants access, demonstrating insufficient password policy enforcement.

**Figure 25. Application Exposes Internal Error Object Due to Improper Input Handling**



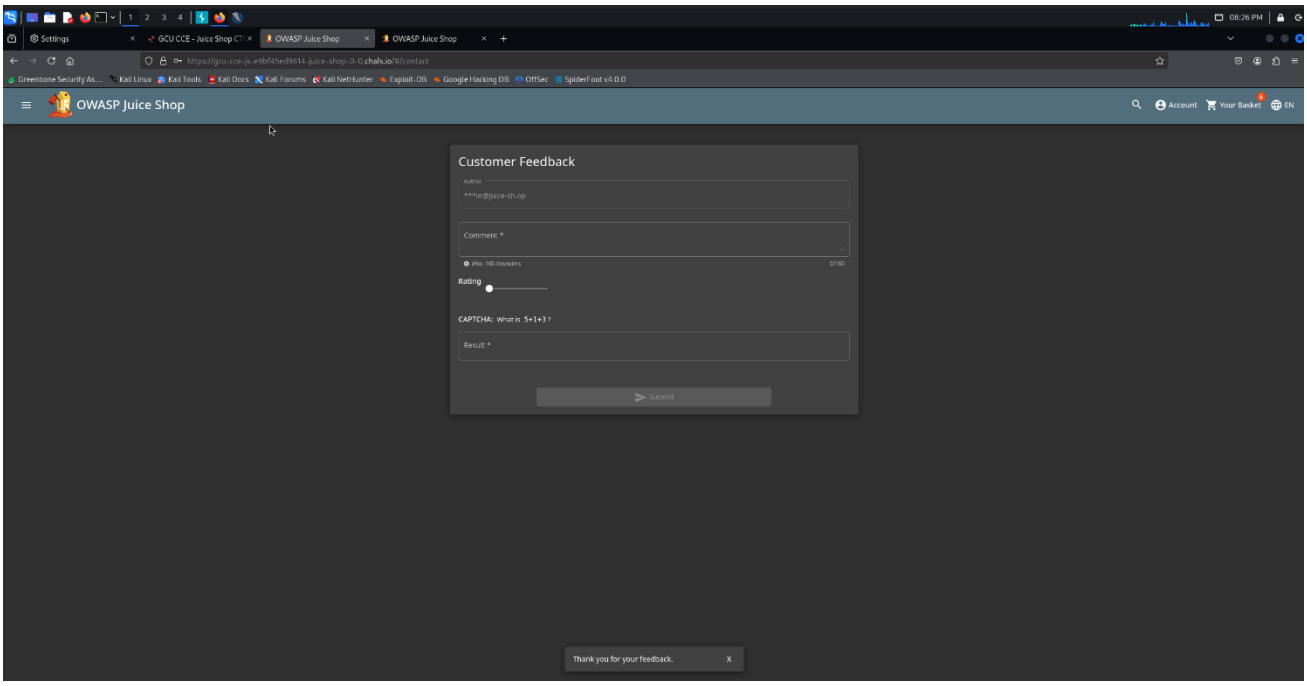

This figure shows the login page displaying an internal error message ([object Object]) after invalid input is submitted. This indicates improper error handling and exposure of internal application behavior to the user.

**Figure 26. Successful Error Handling Challenge Completion**



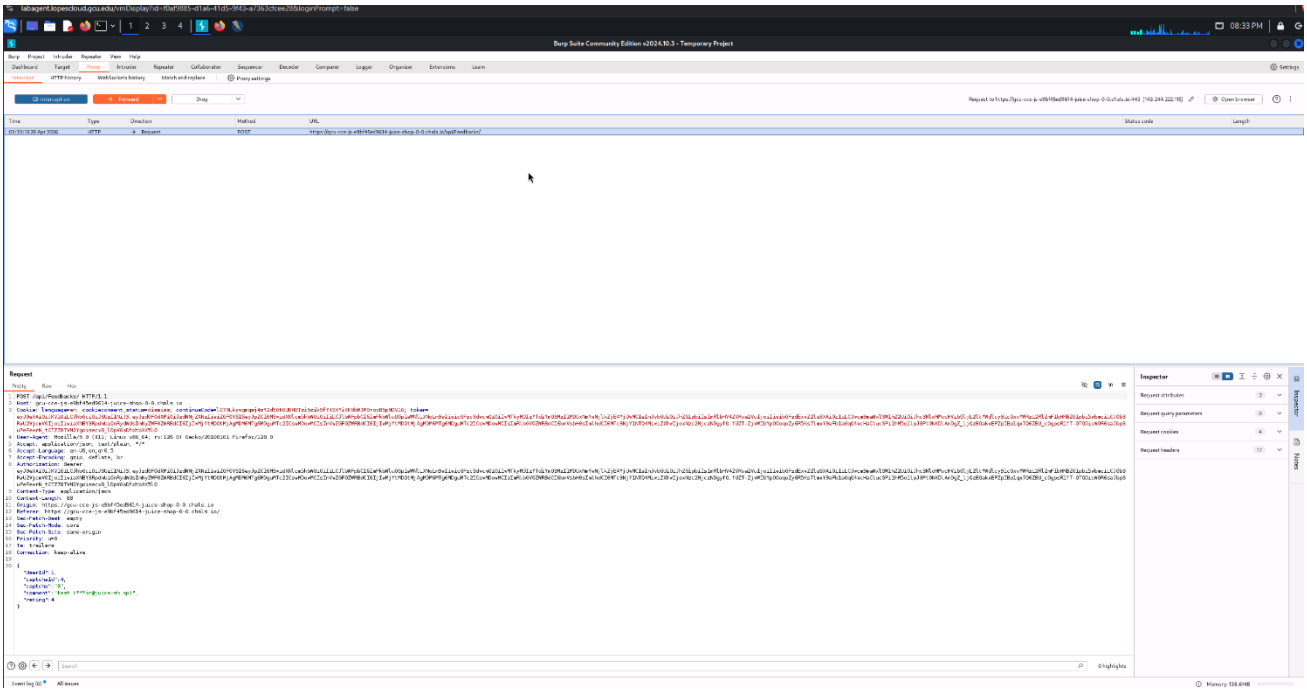

This figure confirms that the error handling vulnerability was successfully triggered and identified by the application. The system displays a success notification after provoking an improperly handled error.

**Figure 27. Customer Feedback Form with CAPTCHA Challenge (Pre-Attack State)**



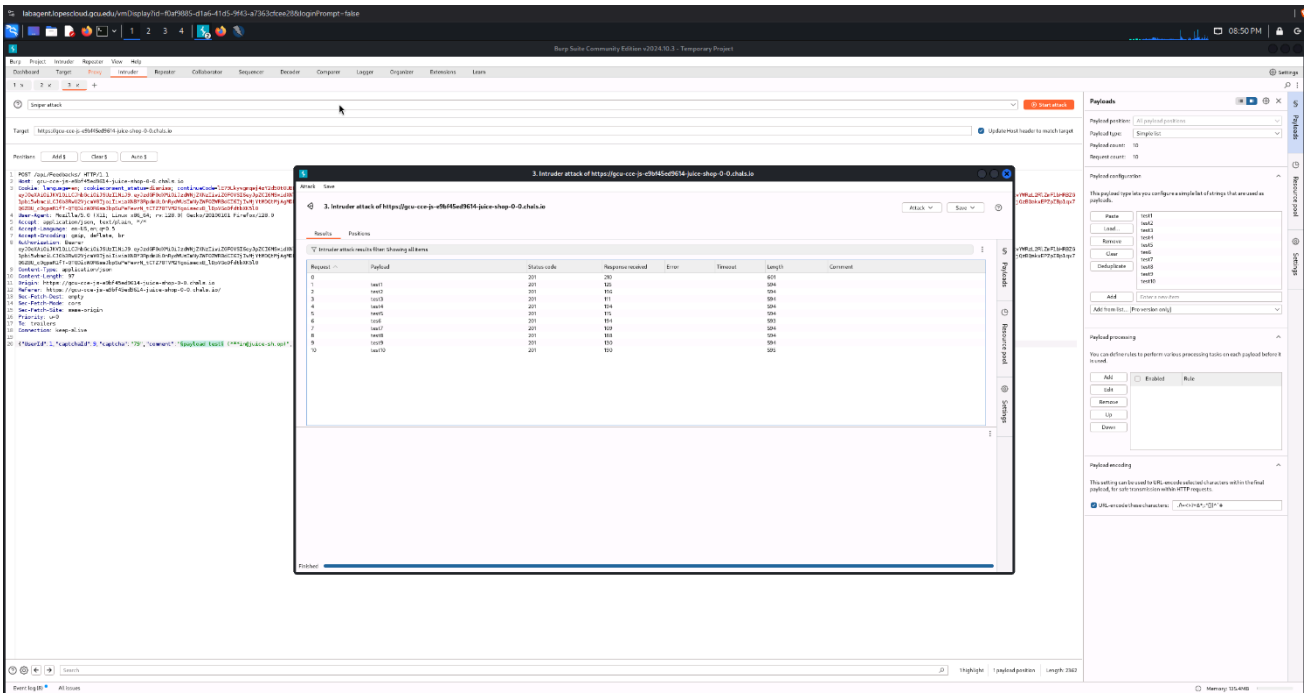

This figure shows the customer feedback form with a CAPTCHA challenge designed to prevent automated submissions.

**Figure 28. Burp Suite Intercept Capturing Feedback Submission Request**



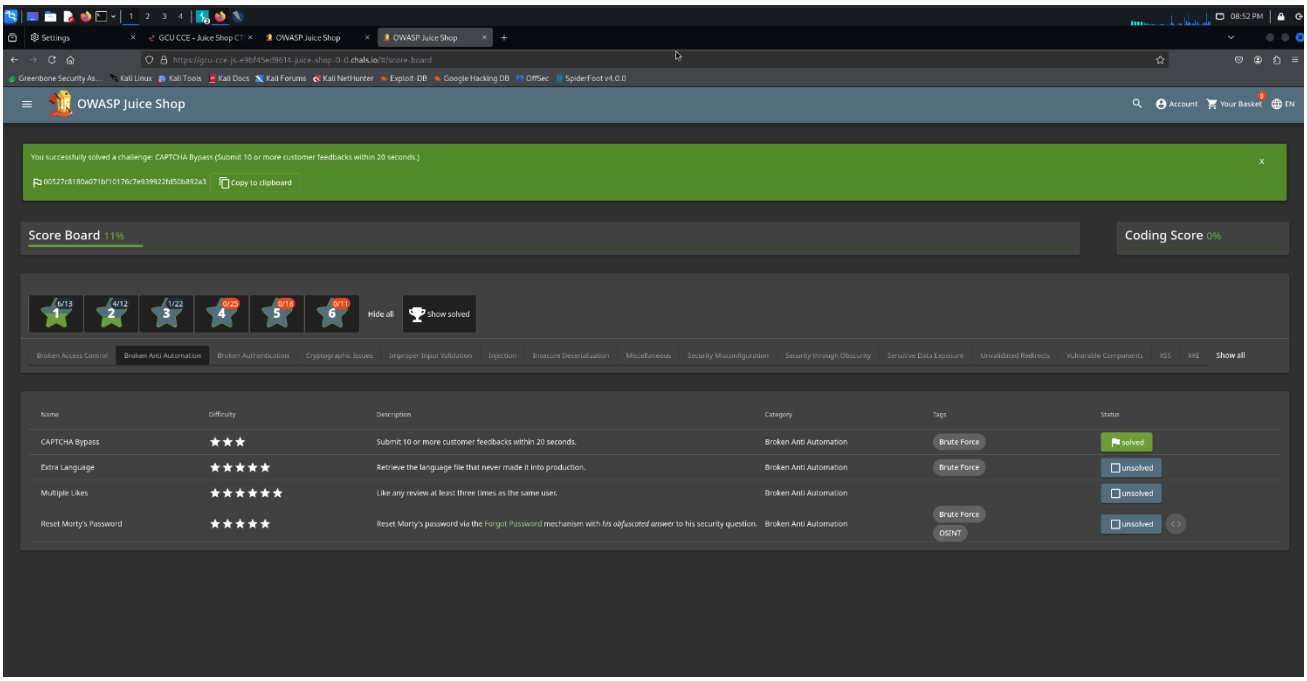
This figure shows Burp Suite intercepting the HTTP POST request submitted from the feedback form, including CAPTCHA and user input parameters.

**Figure 29. Burp Suite Intruder Configured for Automated CAPTCHA Bypass Attack**



This figure shows Burp Intruder configured to automate multiple feedback submissions by manipulating request payloads, effectively bypassing the CAPTCHA mechanism.

**Figure 30. OWASP Juice Shop Scoreboard Confirming CAPTCHA Bypass Challenge Completion**




This figure confirms successful exploitation of the CAPTCHA bypass vulnerability after submitting multiple automated requests within a short time frame.