Web Application Firewall

Practical Assignment 2 of STI - Report

Tiago Ventura – 2019243695 Sancho Simões – 2019217590

Masters in Intelligent Systems University of Coimbra

Coimbra, 24rd of may 2023

Contents

1 Goal

The primary objectives of this project are twofold. First, we aim to delve into the realm of web application security by thoroughly exploring the **OWASP JuiceShop** website, while adhering to the comprehensive guidelines provided by the **Web Security Testing Guide (WSTG)**. This endeavor involves employing powerful tools such as **Kali Linux** and **OWASP ZAP** to conduct rigorous web security testing.

During the initial phase, we will diligently examine the JuiceShop website, meticulously following the WSTG guidelines. Our focus will be on identifying potential vulnerabilities and assessing their exploitability. We will undertake various penetration testing techniques, including **automated** and **active scans**, **fuzz attacks** on the login form, and **manual penetration tests** on logged-in threats. Furthermore, we will configure **OWASP ZAP** to perform active scans in **authenticated areas**. The culmination of these efforts will be the creation of a comprehensive web application security report that documents the identified vulnerabilities and provides insights into their exploitation potential.

In the second phase, we shift our attention to implementing a robust web application firewall (WAF) as a proactive measure to safeguard the JuiceShop website against **application-layer attacks**. Leveraging the power of **ModSecurity reverse proxy**, we will set up an **Apache 2** service to act as the WAF. Our objective is to **monitor, filter, and block HTTP traffic** destined for the JuiceShop, effectively fortifying its security posture. The WAF configuration will be optimized to thwart a wide range of potential attacks.

To evaluate the efficacy of our implemented WAF, we will conduct a series of repeat penetration tests, reproducing the same scenarios from the initial phase. This iterative testing will enable us to gauge the performance of the WAF in detecting and preventing previously identified security issues. The results and observations from these tests will be carefully documented and incorporated into an updated version of the web application security report, highlighting the configurations, test descriptions, and performance outcomes.

In conclusion, this project aims to enhance our understanding of web application security testing methodologies and the implementation of a web application firewall. By immersing ourselves in practical hands-on activities, we aspire to acquire valuable knowledge and skills in the field of web security. Through diligent testing, analysis, and documentation, we aim to contribute to the broader realm of web security practices and fortify the security posture of the JuiceShop website.

2 Introduction

In this project, we embark on a comprehensive exploration of web application security and the implementation of a web application firewall (WAF). Our primary focus is on securing the **OWASP JuiceShop** website, a vulnerable web application, against various application-layer attacks. To accomplish this, we utilize cutting-edge tools and methodologies, adhering to the guidelines outlined in the **Web Security Testing Guide (WSTG)**.

2.1 OWASP JuiceShop

The **OWASP JuiceShop** is a purposely vulnerable web application designed for security testing and educational purposes. It simulates real-world security vulnerabilities and provides an ideal platform for assessing web application security.

2.2 Web Security Testing Guide (WSTG)

The **Web Security Testing Guide (WSTG)** is a comprehensive guide developed by the Open Web Application Security Project (OWASP). It offers detailed guidelines and best practices for conducting web application security tests, covering various aspects of web security.

2.3 Kali Linux

Kali Linux is a powerful operating system specifically designed for penetration testing and ethical hacking. It provides a wide range of tools and resources to aid in comprehensive web security testing.

2.4 OWASP ZAP

OWASP ZAP (Zed Attack Proxy) is an open-source web application security scanner. It is widely used for identifying vulnerabilities and performing various security testing activities, such as automated and active scans.

2.5 Web Application Firewall (WAF)

A **web application firewall (WAF)** is a security solution that monitors, filters, and blocks HTTP traffic to protect web applications from malicious attacks. It acts as a barrier between the application and external threats, enhancing the overall security posture.

3 Web Application Security Testing

The initial phase of our project focuses on thorough web application security testing. Following the WSTG guidelines, we perform a series of tests to identify vulnerabilities and assess their exploitability.

- Perform an **automated scan** to identify common security issues automatically.
- Conduct an **active scan** to explore the effectiveness of various security policies.
- Utilize relevant **add-ons** to enhance testing capabilities and maximize threat identification.
- Execute a **Fuzz attack** on the login form to uncover potential vulnerabilities.
- Perform **manual penetration tests** to identify threats that arise after user authentication.
- Configure **OWASP ZAP** to perform active scans in **authenticated areas** of the web application.

The outcomes of these tests will be documented in a comprehensive web application security report, highlighting identified vulnerabilities and their potential exploitation methods.

4 Web Application Firewall (WAF)

In the second phase of our project, we deploy a web application firewall (WAF) to enhance the security of the JuiceShop website. The WAF, based on **ModSecurity** reverse proxy, will effectively monitor, filter, and block HTTP traffic destined for the JuiceShop, providing an additional layer of protection against application-layer attacks.

The WAF (Web Application Firewall) configuration involved setting up a new machine with the Apache2 service. Apache2 was configured in reverse proxy mode. In the file /etc/apache2/sites-enabled/000-default.conf, the IP address of the target machine was specified.

```
<VirtualHost *:*>
    ProxyPreserveHost On

    # Servers to proxy the connection, or;
    # List of application servers:
    # Usage:
    # ProxyPass / http://[IP Addr.]:[port]/
    # ProxyPassReverse / http://[IP Addr.]:[port]/
    # Example:
    ProxyPass / http://10.0.0.2:3000/
    ProxyPassReverse / http://10.0.0.2:3000/

    ServerName localhost
</VirtualHost>
```

Figure 1: Reverse Proxy.

After configuring Apache, the security mode was enabled by adding the "mod_security" module. In the configuration file /etc/modsecurity/modsecurity.conf, the SecRuleEngine directive was set to "On" to enable filtering and blocking of attacks.

```
# -- Rule engine initialization ----------------------------------------------

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
#SecRuleEngine DetectionOnly

SecRuleEngine On

# -- Request body handling ---------------------------------------------------
```

Figure 2: SecRuleEngine.

Web Application Firewalls are an important security measure that helps protect web applications from various types of attacks. They analyze incoming and outgoing HTTP traffic, detect malicious patterns or behaviors, and block or filter out potentially harmful requests. WAFs add an additional layer of defense to the application, complementing other security measures such as secure coding practices and regular vulnerability assessments.

By configuring Apache2 as a reverse proxy with mod_security, the WAF can intercept incoming requests, inspect them for potential threats, and take appropriate actions based on the defined rules. The mod_security module provides a flexible and customizable framework for defining security policies and rulesets to protect against known attack vectors, such as SQL injection, cross-site scripting (XSS), or file inclusion vulnerabilities.

The specific configuration mentioned involves enabling the SecRuleEngine directive in the modsecurity.conf file, which activates the rule engine and enables the WAF to actively filter and block malicious requests. Fine-tuning the WAF rulesets and policies based on the application's specific requirements and known attack patterns is essential for maximizing security while minimizing false positives.

Overall, the WAF configuration using Apache2 and mod_security enhances the security posture of the web application by actively monitoring and protecting against common web-based attacks. It adds an extra layer of defense to help mitigate the risk of unauthorized access, data breaches, and other security incidents.

5 Architecture

This section provides an overview of the architecture of the project, including the network structure, servers, and services involved.

5.1 Network Structure

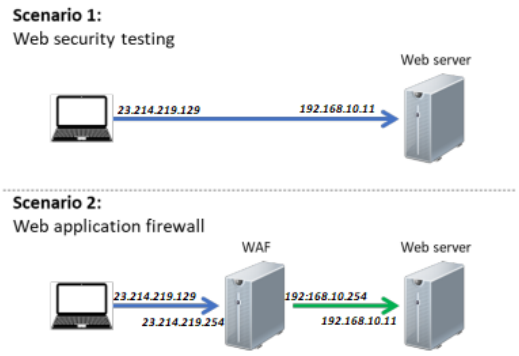

Figure 3: Architeture.

In both scenarios, we utilized three virtual machines (VMs). In the first scenario, the WAF machine acted solely as a router, forwarding requests to the Juice Shop. However, in the second scenario, the same machine served as a reverse proxy, functioning as both a WAF and an intermediary router between the client (attacker) and the Juice Shop.

This setup allowed for a clear separation of functionalities, with each machine serving a distinct purpose. The use of three VMs provided the flexibility needed for vulnerability detection and WAF testing in both scenarios.

By designating specific roles to each VM, we ensured effective isolation of functions. This segregation facilitated targeted analysis and testing of the WAF's capabilities, while also providing a controlled environment for assessing vulnerabilities in the Juice Shop application.

The VM structure proved advantageous, allowing for seamless adaptation to different testing scenarios. Whether focusing on vulnerability detection or WAF evaluation, the setup provided a reliable framework for conducting thorough assessments and enhancing the overall security of the system.

5.2 Servers

The servers utilized in the project are presented, specifying their roles and responsibilities within the architecture.

The server configuration consists of three main components:

1. **Attacker/Pentester:** This server represents the machine used by the attacker or pentester. It is running KaliLinux and OWASP ZAP, and its IP address is `23.214.219.129` (interface `enp0s10`).

2. **Web Application Firewall:** The server acts as a reverse proxy and web application firewall. It is responsible for filtering and protecting incoming traffic. It has two IP addresses: `23.214.219.254` (interface `enp0s9`) and `192.168.10.254` (interface `enp0s10`).

3. **JuiceShop Web Application Server:** This server hosts the JuiceShop web application. It is accessible at IP address `192.168.10.1` (interface `enp0s10`).

5.3 Services

The services employed in the project are outlined, emphasizing their functionalities and interactions with other components.

Table 1: Services

Service Name	**Endpoint**	Status	**Version**
Apache HTTP Server	http://localhost:80	Running	2.4.46
MySQL Server	localhost:3306	Running	8.0.23
OW JuiceShop	http://localhost:3000	Running	14.5.1

Apache HTTP Server: The Apache HTTP Server is a widely used web server software. In this project, it is responsible for handling HTTP requests and serving web content. The server is currently running on the local machine (localhost) and is accessible through the endpoint http://localhost:80. The version being used is 2.4.46.

MySQL Server: The MySQL Server is a popular relational database management system. It is utilized in the project to store and manage the application's data. The server is running on the local machine, and its endpoint is set to localhost:3306. The current version being used is 8.0.23.

OW JuiceShop: OW JuiceShop is an intentionally vulnerable web application used for security testing and learning purposes. It simulates real-world security vulnerabilities and provides an environment to assess and enhance web application security. The JuiceShop service is currently running on the local machine, and its endpoint is http://localhost:3000. The version being used is 14.5.1.

These services work together to support the functionality and security of the project. The Apache HTTP Server handles incoming HTTP requests, while the MySQL Server manages the storage and retrieval of data. The OW JuiceShop application serves as the target for security testing, enabling the identification and resolution of vulnerabilities.

Overall, the effective functioning and coordination of these services are crucial for the smooth operation of the project and ensuring a secure and reliable web application environment.

6 Web Application Security Testing

This section focuses on the web application security testing phase, where various testing techniques are applied to identify vulnerabilities and weaknesses in the web application.

6.1 Information Gathering

When testing a web application, it is important to gather information about it to better understand its vulnerabilities and potential risks. In this regard, several steps were taken to explore the JuiceShop application and conduct comprehensive testing.

Firstly, an information search was performed to gather relevant details about JuiceShop. This initial research provided valuable insights into the application before proceeding to the testing phase.

To further assess the web application, a process called fingerprinting was employed. This involved identifying the specific web application framework or technologies used in its development. By analyzing the application's responses, headers, and other identifiable characteristics, it was possible to determine the framework or CMS employed. This information helps in understanding the underlying components and potential vulnerabilities associated with the chosen framework.

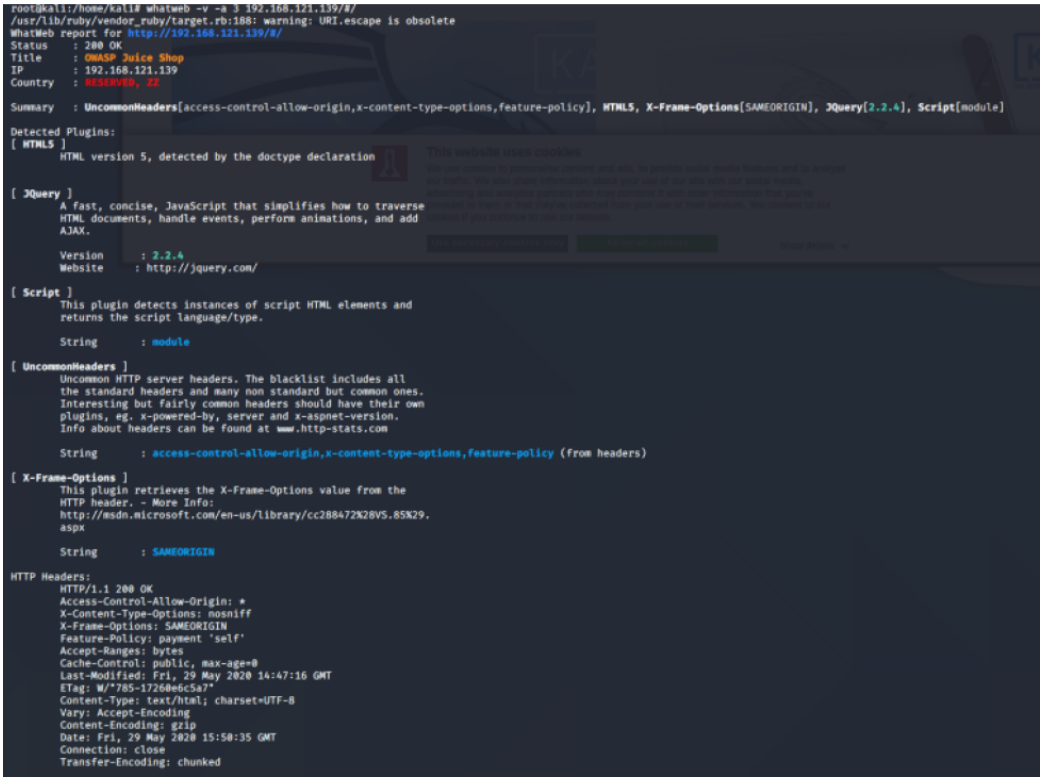

Figure 4: Fingerprint.

Additionally, the webpage content was thoroughly reviewed to identify any potential information leakage. During the development of a website, developers may inadvertently leave behind comments, detailed information, or metadata in the code. These elements can contain sensitive or confidential information that should not be exposed to the public. By using security testing tools like ZAP, the webpage content was analyzed to detect any instances of information leakage. Alerts were generated to highlight the presence of sensitive information, signaling the need for immediate attention and appropriate actions to mitigate the associated risks.

Medium (High)	Trace.axd Information Leak
Description	The ASP.NET Trace Viewer (trace.axd) was found to be available. This component can leak a significant amount of valuable information.
URL	http://192.168.121.139/assets/public/trace.axd
Method	GET
Evidence	HTTP/1.1 200 OK
URL	http://192.168.121.139/assets/trace.axd
Method	GET
Evidence	HTTP/1.1 200 OK
URL	http://192.168.121.139/trace.axd
Method	GET
Evidence	HTTP/1.1 200 OK
Instances	3
Solution	Consider whether or not Trace Viewer is actually required in production, if it isn't then disable it. If it is then ensure access to it requires authentication and authorization.
Reference	https://msdn.microsoft.com/en-us/library/bb386420.aspx https://msdn.microsoft.com/en-us/library/wwh16c6c.aspx https://www.dotnetperls.com/trace
CWE Id	215
WASC Id	13
Source ID	1

Medium (High)	.env Information Leak
Description	One or more .env files seems to have been located on the server. These files often expose infrastructure or administrative account credentials, API or APP keys, or other sensitive configuration information.
URL	http://192.168.121.139/.env
Method	GET
Evidence	HTTP/1.1 200 OK
URL	http://192.168.121.139/assets/.env
Method	GET
Evidence	HTTP/1.1 200 OK
URL	http://192.168.121.139/assets/public/.env
Method	GET
Evidence	HTTP/1.1 200 OK

Figure 5: Information Leak.

By conducting these activities, valuable insights were gained regarding the JuiceShop application. The search engine reconnaissance, fingerprinting, and webpage content review helped uncover potential vulnerabilities, exposure of sensitive information, and areas requiring mitigation. The findings from these steps are essential in strengthening the overall security of the web application, enabling necessary remediation measures to be implemented and reducing the risk of unauthorized access or compromise of sensitive data.

6.2 Configuration and Deployment Management Testing

During the testing process of the JuiceShop web application, several important steps were taken to gather information, assess vulnerabilities, and identify potential risks.

One aspect of the testing involved examining the HTTP methods used by the website. By executing a specific command, it was determined that the TRACE method was not active on the site. This method, if enabled, could allow attackers to access cookies through JavaScript, potentially leading to session hijacking and unauthorized access. Additionally, attempting to use the TRACE method or the DELETE method resulted in redirection to the home page, indicating that the site was protected against these types of attacks.

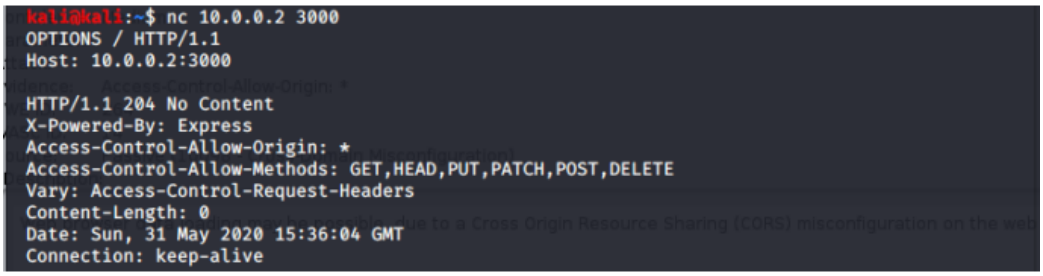

Figure 6: HTTPS.

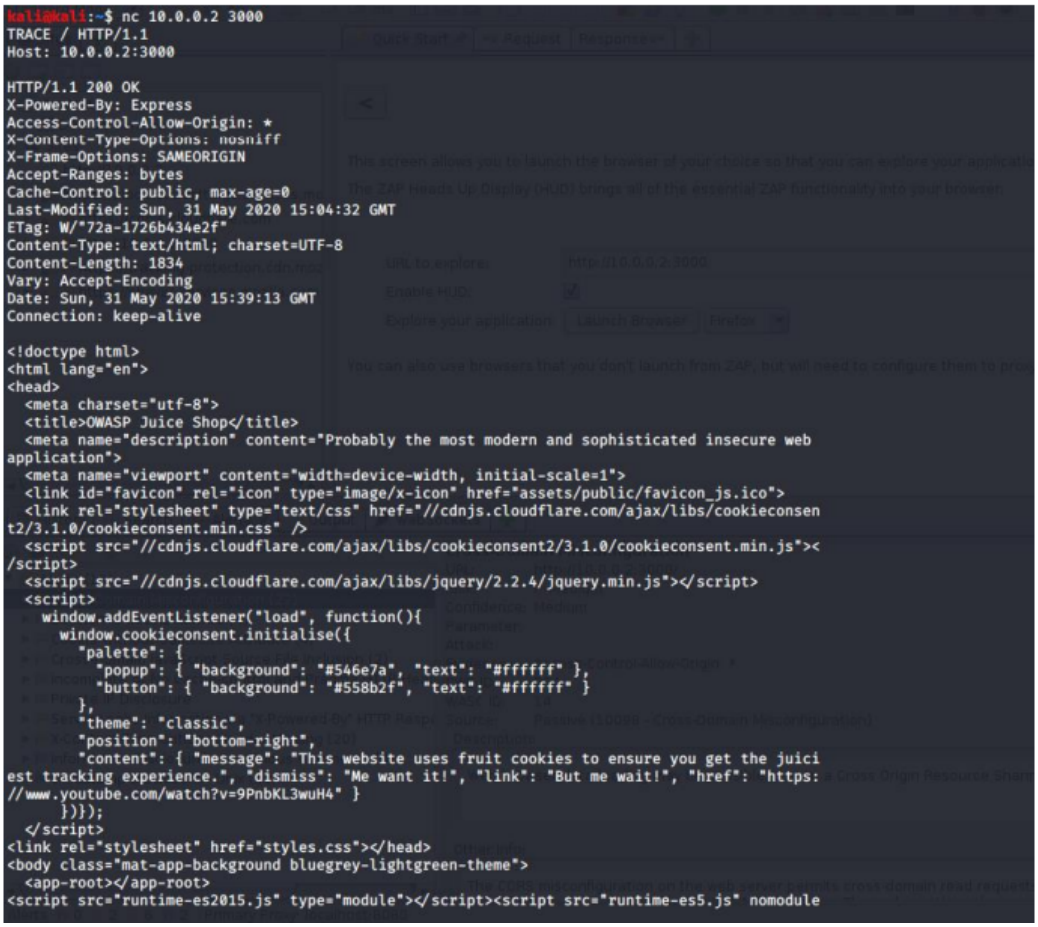

Figure 7: TRACE.

Figure 8: DELETE.

Another area of focus was reviewing old backup and unreferenced files for sensitive information. Although most files on a web server are managed directly by the server, there is a possibility of forgotten files containing important infrastructure and system credentials. The security testing tool, ZAP, raised an alert regarding this potential vulnerability.

Figure 9: Old Backup.

The JuiceShop web application was also found to have a vulnerability related to Cross-Origin Resource Sharing (CORS). Improper configuration of the web server can leave it susceptible to cross-domain attacks. ZAP highlighted this vulnerability through an alert, indicating that the necessary authentication settings were visible to all. This could potentially enable attackers to launch brute-force attacks against user accounts.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link id="favicon" rel="icon" type="image/x-icon" href="assets/public/favicon_js.ico">
<link rel="stylesheet" type="text/css" href=
"//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.css" />
<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>
  window.addEventListener("load", function(){
    window.cookieconsent.initialise({
```

Figure 10: CORS.

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Range: items 0-12/13
Content-Type: application/json; charset=utf-8
Content-Length: 1792
ETag: W/"700-ycJVzFi6IEobKr/v2WhCzQvJKlA"
Vary: Accept-Encoding
Date: Sun, 31 May 2020 12:44:47 GMT
Connection: keep-alive
```

```
{"status":"success","data":[{"id":1,"question":"Your eldest siblings middle name?","createdAt":
"2020-05-31T12:42:08.133Z","updatedAt":"2020-05-31T12:42:08.133Z"},{"id":2,"question":
"Mother's maiden name?","createdAt":"2020-05-31T12:42:08.133Z","updatedAt":"2020-05-31T12:42:08.133Z"},{
"id":3,"question":"Mother's birth date? (MM/DD/YY)","createdAt":"2020-05-31T12:42:08.135Z","updatedAt":
"2020-05-31T12:42:08.135Z"},{"id":4,"question":"Father's birth date? (MM/DD/YY)","createdAt":
"2020-05-31T12:42:08.135Z","updatedAt":"2020-05-31T12:42:08.135Z"},{"id":5,"question":
"Maternal grandmother's first name?","createdAt":"2020-05-31T12:42:08.135Z","updatedAt":
```

Figure 11: Access Control.

Additionally, the testing process involved evaluating the security of the cloud storage used by the application. Cloud services provide convenient storage and accessibility for web applications. However, improper access control settings can result in the exposure of confidential information, data breaches, or unauthorized access. In the case of JuiceShop, ZAP identified a vulnerability related to the cloud storage configuration.

High (Low)	Cloud Metadata Potentially Exposed
Description	The Cloud Metadata Attack attempts to abuse a misconfigured NGINX server in order to access the instance metadata maintained by cloud service providers such as AWS, GCP and Azure. All of these providers provide metadata via an internal unroutable IP address '169.254.169.254' - this can be exposed by incorrectly configured NGINX servers and accessed by using this IP address in the Host header field.
URL	http://192.168.121.139/latest/meta-data/
Method	GET
Attack	169.154.169.254
Instances	1

Figure 12: Cloud.

By uncovering and addressing these vulnerabilities and risks, the overall security posture of the JuiceShop web application can be strengthened, mitigating potential threats and protecting sensitive data.

6.3 Identity Management Testing

During the testing of the JuiceShop web application, various aspects of identity management were examined to ensure secure user registration and protect against potential vulnerabilities.

Two types of users were identified on the JuiceShop site: administrators and regular users. Role definitions play a crucial role in determining the access levels and permissions granted to different user types, allowing for appropriate privileges and restrictions to be enforced.

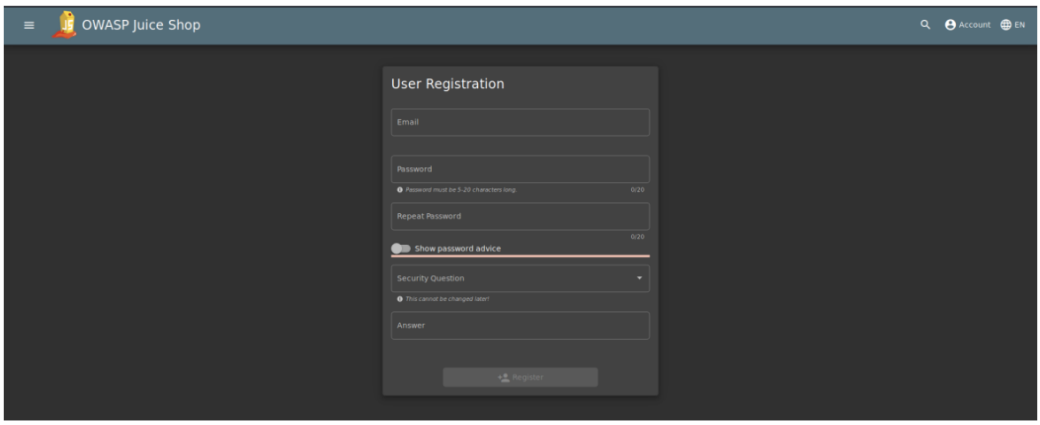

Figure 13: Juice Shop Registration.

The user registration process was evaluated to assess its alignment with business and security requirements. It was confirmed that anyone could register to access the application, and registrations were automatically granted without human review. Multiple registrations were allowed with different email addresses. However, users were not able to register for different roles or permissions. The process required proof of identity, but registered identities were not verified.

To prevent attackers from obtaining a list of registered users, it was important to ensure that the application responded uniformly to different login failures. In the case of JuiceShop, it was observed that the error message "Invalid email or password" was displayed regardless of whether the entered email did not correspond to a valid email address or if the password was incorrect. This consistent error message made it more challenging for attackers to differentiate between valid and invalid user accounts.

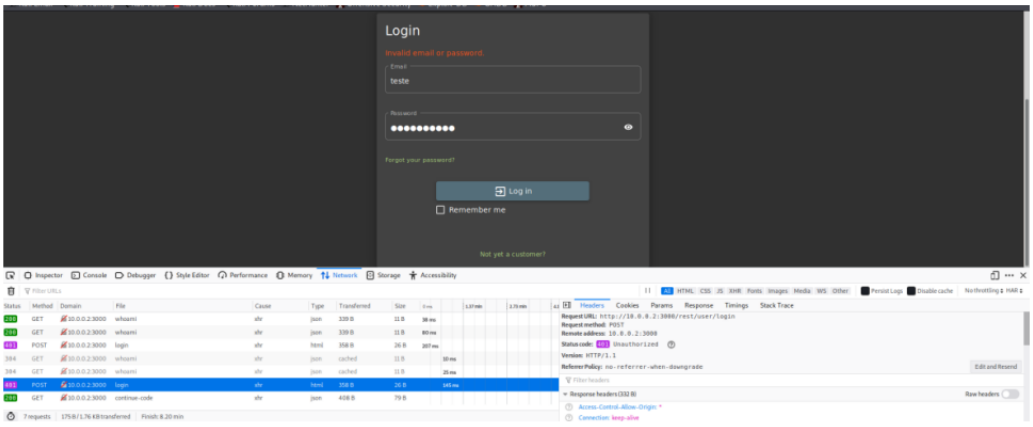

Figure 14: Username Failure.

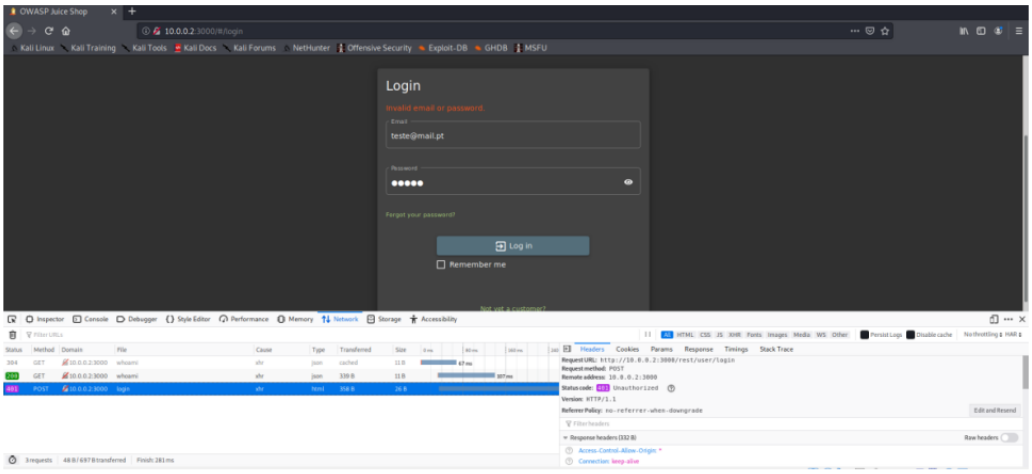

Figure 15: Password Failure.

Another aspect of identity management testing involved evaluating the username policy. JuiceShop required a specific structure for account names, including the presence of the "@" character and at least one alphabetic character before and after it. During the registration process, if an email that already existed was entered, a message requesting a valid email address was displayed. Additionally, a message was presented if the account name structure was not followed. These measures aimed to maintain a stronger username policy and prevent attackers from easily enumerating registered user accounts.

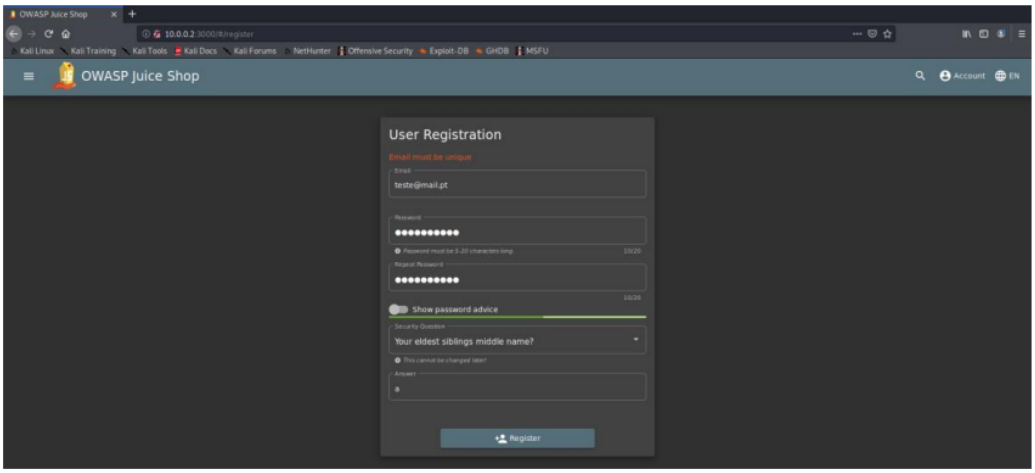

Figure 16: Username Policy 1.

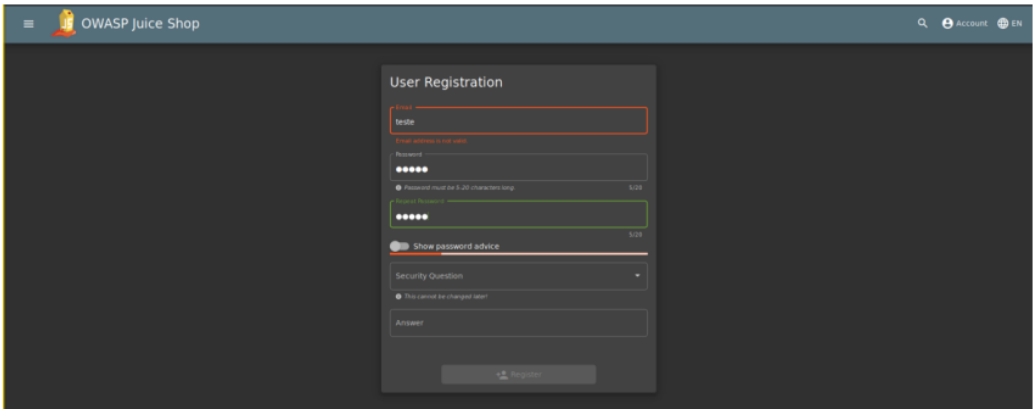

Figure 17: Username Policy 2.

By conducting thorough identity management testing and addressing any identified weaknesses or vulnerabilities, the JuiceShop web application can enhance its security measures and ensure the confidentiality and integrity of user identities throughout the registration and account management processes.

6.4 Authentication Testing

During the authentication testing phase, several aspects of the JuiceShop web application's authentication mechanisms were evaluated to identify potential vulnerabilities and strengthen security measures.

One of the findings was that user credentials were transmitted over an unencrypted channel. When conducting a manual exploration using OWASP ZAP, it was observed that user credentials were visible in the request history. This lack of encryption exposes the credentials to potential interception by attackers who may be monitoring the network. Such a vulnerability can lead to identity hijacking and unauthorized access to user accounts.

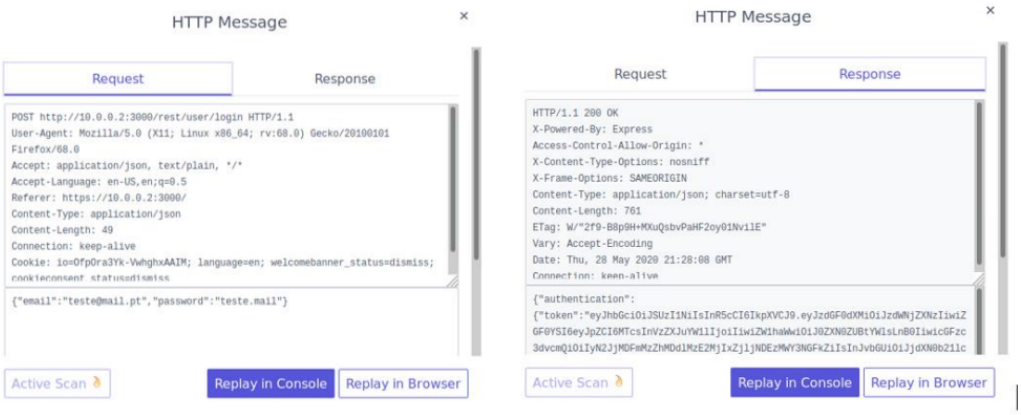

Figure 18: User Data Exposed.

Another vulnerability identified was related to default credentials. Through the detection of SQL injection vulnerabilities using ZAP, a specific user's username was obtained. By using brute-force techniques, the user's password, "admin123," was discovered. While the username itself may not be a default value, the simplicity and standardization of the password pose a significant security risk. It is crucial to eliminate default or easily guessable credentials to prevent unauthorized access.

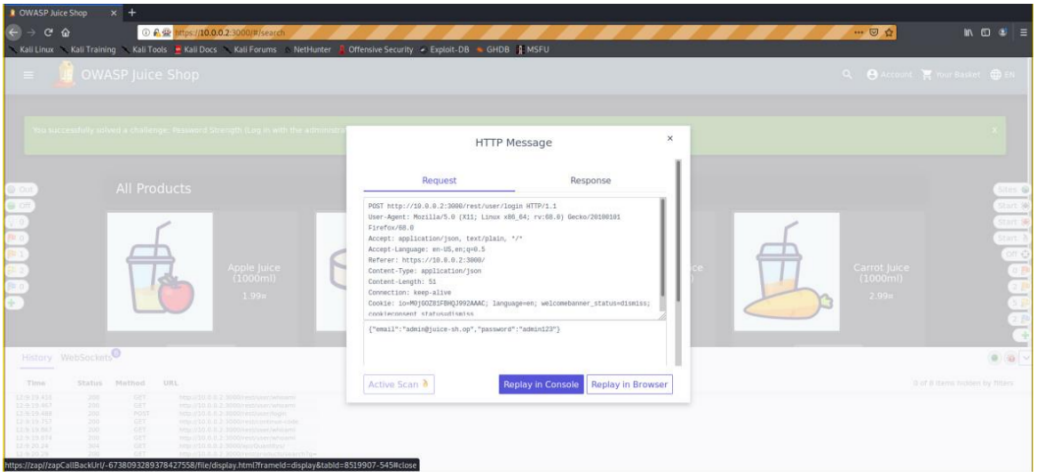

Figure 19: Credentials.

The JuiceShop website also exhibited a weak lockout mechanism. After multiple consecutive failed login attempts with an incorrect password, it was found that the site did not impose any restrictions on subsequent login attempts. This lack of protection against brute-force attacks allows attackers to try multiple passwords until they gain access to a valid user account. Implementing a strong lockout mechanism can mitigate this vulnerability.

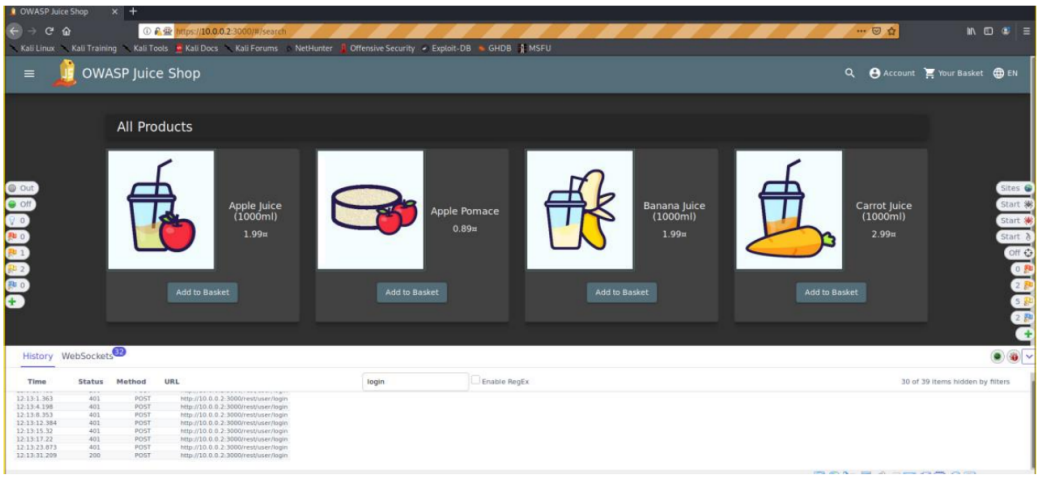

Figure 20: Blocking Mechanism.

In terms of password security, JuiceShop did not enforce a stringent password policy. Although users were advised to create strong passwords during registration, they were not restricted from choosing weak passwords. This lack of enforcement increases the risk of password-based attacks, such as brute-force or dictionary attacks. Implementing a robust password policy that includes complexity requirements and restrictions on commonly used or easily guessable passwords is essential to enhance security.

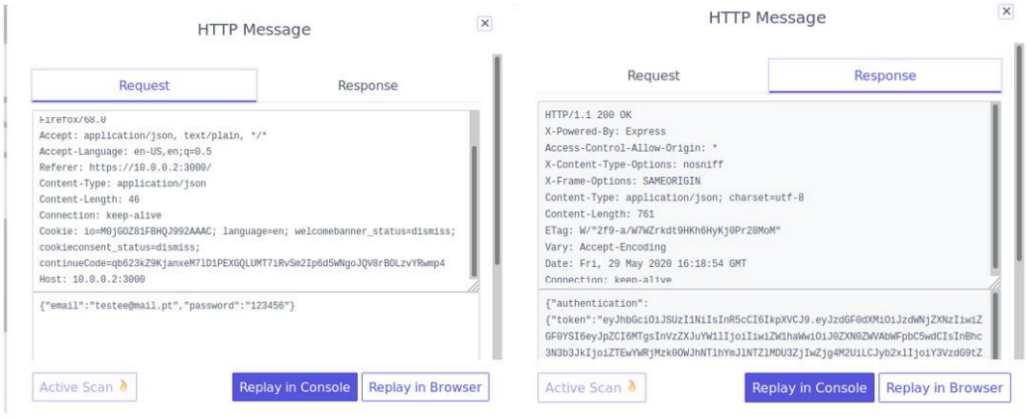

Figure 21: Password Policy.

Additionally, the security questions in JuiceShop were found to have weak answers. Attackers can exploit this weakness by systematically attempting various names or dates to guess the answers and gain unauthorized access to user accounts. Strengthening the security questions and ensuring that the answers are sufficiently secure can mitigate this vulnerability.

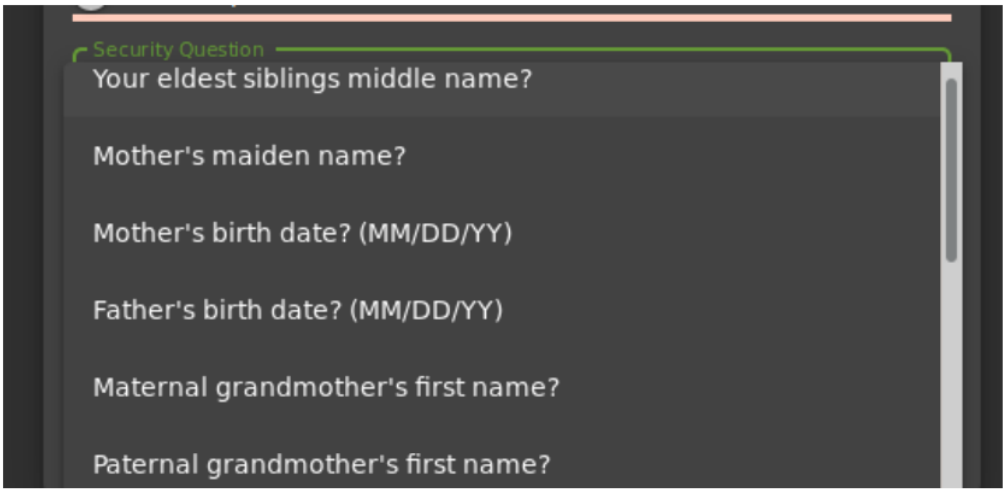

Figure 22: Security Questions.

Addressing these authentication vulnerabilities is crucial to safeguard user credentials and prevent unauthorized access. By implementing encryption for transmitting credentials, eliminating default credentials, improving lockout mechanisms, enforcing strong password policies, and strengthening security question answers, the JuiceShop web application can enhance its authentication security and protect user accounts from potential attacks.

6.5 Authorization Testing

During the authorization testing phase, a fuzzing attack was performed on a known user with a simple password. The objective was to assess the effectiveness of the authorization mechanisms and identify any potential vulnerabilities.

The results obtained from the fuzzing attack using ZAP revealed the following:

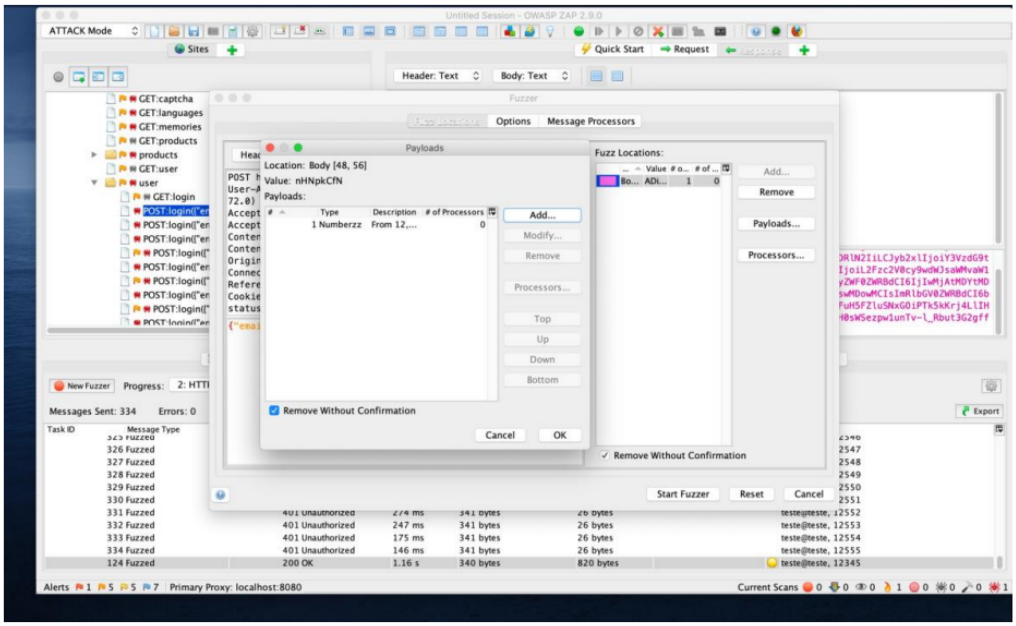

Figure 23: Fuzz.

By conducting thorough authorization testing, it is possible to evaluate the effectiveness of access control mechanisms, identify vulnerabilities, and ensure that users are only granted the appropriate privileges within the application. This helps prevent unauthorized access, maintain data confidentiality, and protect against potential malicious activities.

6.6 Session Management Testing

During the session management testing phase, several aspects related to session handling and security were evaluated. The following findings were observed:

Cookie Attributes: The analysis of cookie attributes revealed that certain attributes, such as path and HttpOnly, were properly configured. However, a warning was raised indicating that the SameSite attribute was not set to "strict." This attribute helps protect against cross-site request forgery (CSRF) attacks by preventing cookies from being sent with cross-site requests.

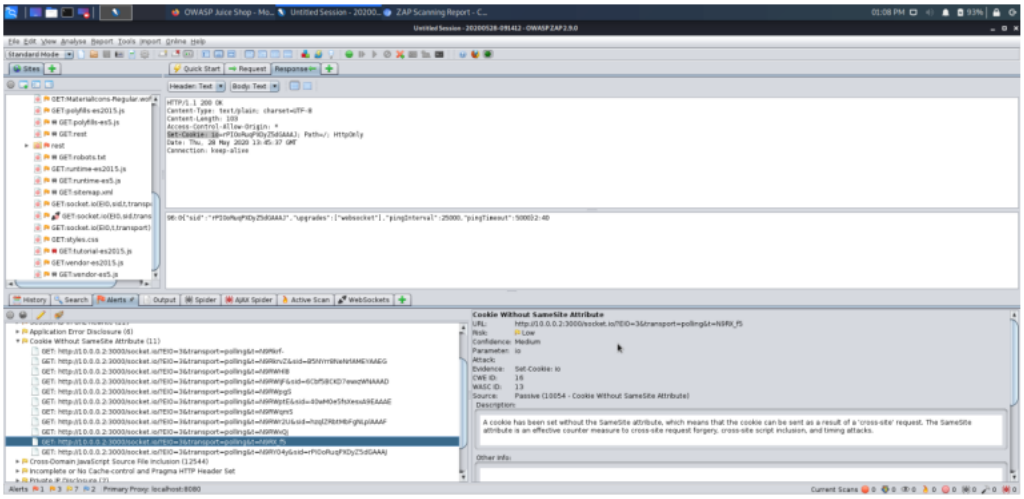

Figure 24: Cookie Attributes 1.

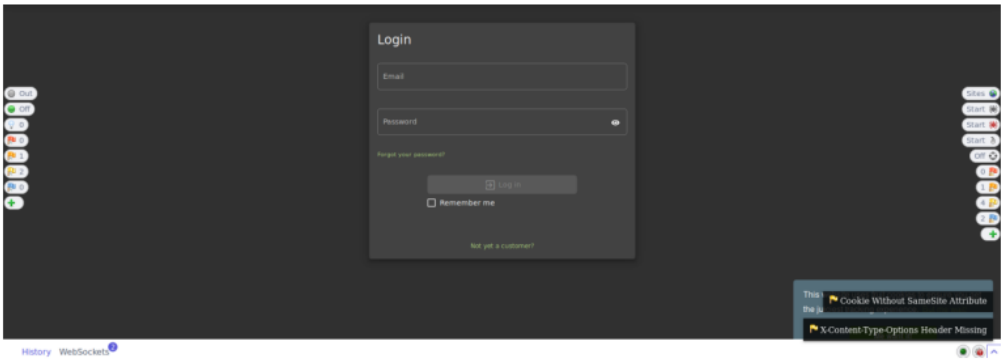

Figure 25: Cookie Attributes 2.

Session Fixation: No session fixation vulnerabilities were identified. The application properly invalidates the existing session ID before authenticating a user and assigns a new session ID upon successful authentication. This mitigates the risk of attackers exploiting session IDs associated with other users.

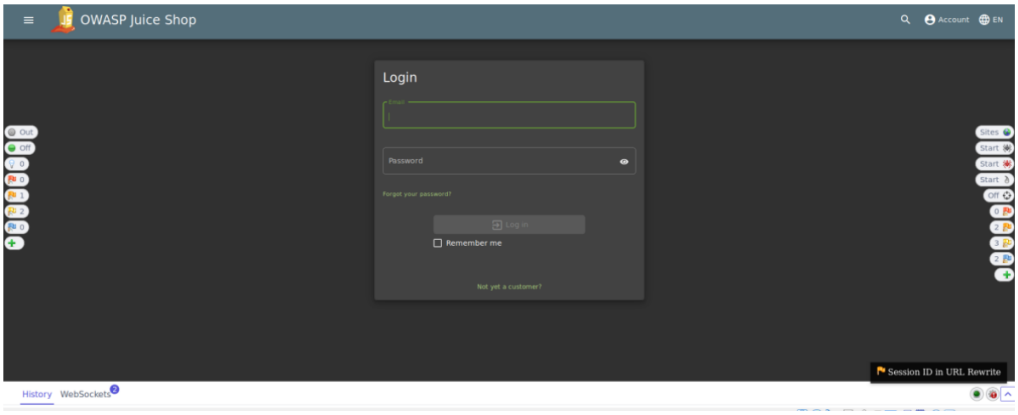

Figure 26: Session Fixation.

Exposed Session Variables: The authentication process generates tokens that are associated with each user's session. However, it was observed that these tokens are not transmitted over an encrypted channel, potentially exposing them to attackers. Despite this, the GET and POST requests during authentication did not reveal any vulnerabilities, indicating that proper protection measures are in place to prevent session exposure.

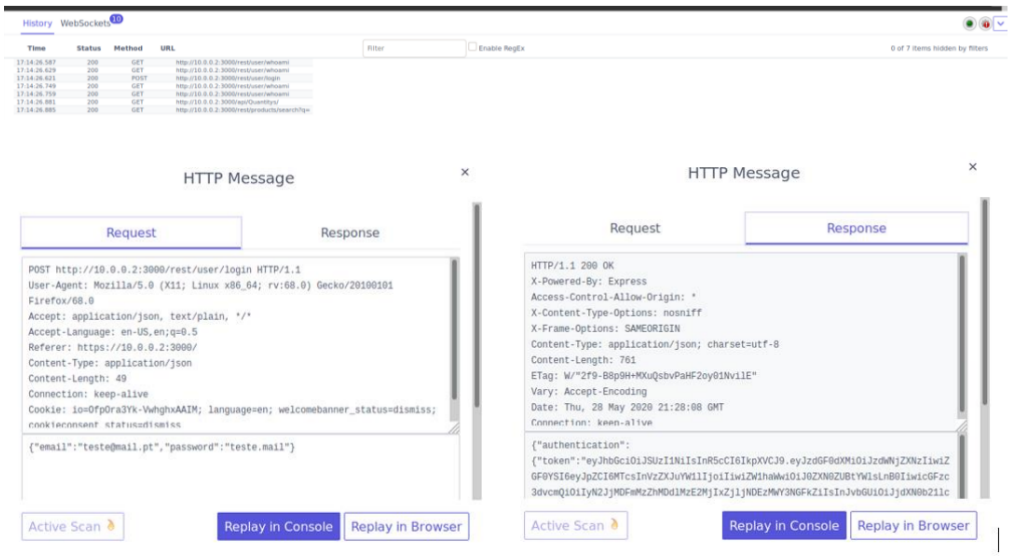

Figure 27: Exposed Session Variables.

Cross-Site Request Forgery (CSRF): The ZAP report detected two instances of CSRF vulnerabilities. CSRF attacks involve tricking authenticated users into unknowingly performing unintended actions on a web application. These vulnerabilities need to be addressed to prevent potential exploitation and ensure the integrity of user actions.

High (Medium)	Anti CSRF Tokens Scanner
Description	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 order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf. CSRF attacks are effective in a number of situations, including: * The victim has an active session on the target site. * The victim is authenticated via HTTP auth on the target site. * The victim is on the same local network as the target site. CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.
URL	http://192.168.121.139/vendor-es5.js
Method	GET
Evidence	<form>
URL	http://192.168.121.139/vendor-es2015.js
Method	GET
Evidence	<form>
Instances	2

Figure 28: CSRF.

By conducting thorough session management testing, potential weaknesses in session handling and security measures can be identified. Addressing these vulnerabilities is crucial to protecting user sessions, preventing unauthorized access, and mitigating risks associated with session fixation and CSRF attacks.

6.7 Input Validation Testing

During the input validation testing phase, the presence of SQL injection vulnerabilities was detected. By using SQL injection as the email during the login process and providing any password, it was observed that the application authenticated the user as the administrator by retrieving the first entry from the Users table.

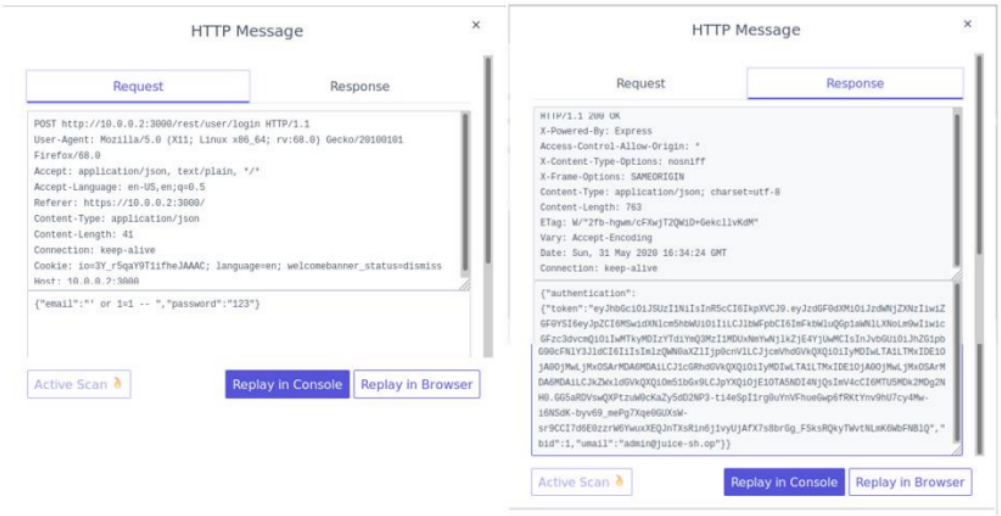

Figure 29: SQL Injection.

SQL injection is a common web application vulnerability that occurs when user-supplied input is not properly validated or sanitized before being included in SQL queries. Attackers can exploit this vulnerability by injecting malicious SQL code, which can lead to unauthorized access, data breaches, or manipulation of the underlying database.

In this case, the application failed to adequately validate and sanitize the user-supplied input, allowing the SQL injection attack to succeed. Proper input validation and sanitization techniques, such as using parameterized queries or prepared statements, should be implemented to prevent SQL injection vulnerabilities.

Addressing SQL injection vulnerabilities is crucial to ensure the security and integrity of the application's data and prevent unauthorized access. By implementing robust input validation practices, developers can mitigate the risk of SQL injection attacks and enhance the overall security posture of the application.

6.8 Testing for Error Handling

During the error handling testing phase, it was observed that the application exhibits non-standard error handling behavior. When a request for a non-existent page is made, instead of returning a proper 404 error response, the application redirects the user to the JuiceShop's homepage. This behavior indicates that the application is designed to hide error details from the user.

Proper error handling is essential for providing a secure and user-friendly experience. Error messages should be informative enough for users to understand the issue, while avoiding the disclosure of sensitive information that could be exploited by attackers. Additionally, returning appropriate HTTP status codes, such as 404 for page not found or 405 for method not allowed, helps to convey the nature of the error to both users and automated systems.

In the case of the PUT method, it was expected to encounter a 405 error indicating that the method is not allowed for the requested resource. However, the application's error handling mechanism overrides this standard behavior and redirects the user to the homepage instead. This deviation from expected error responses could potentially introduce confusion and hinder the identification of underlying issues.

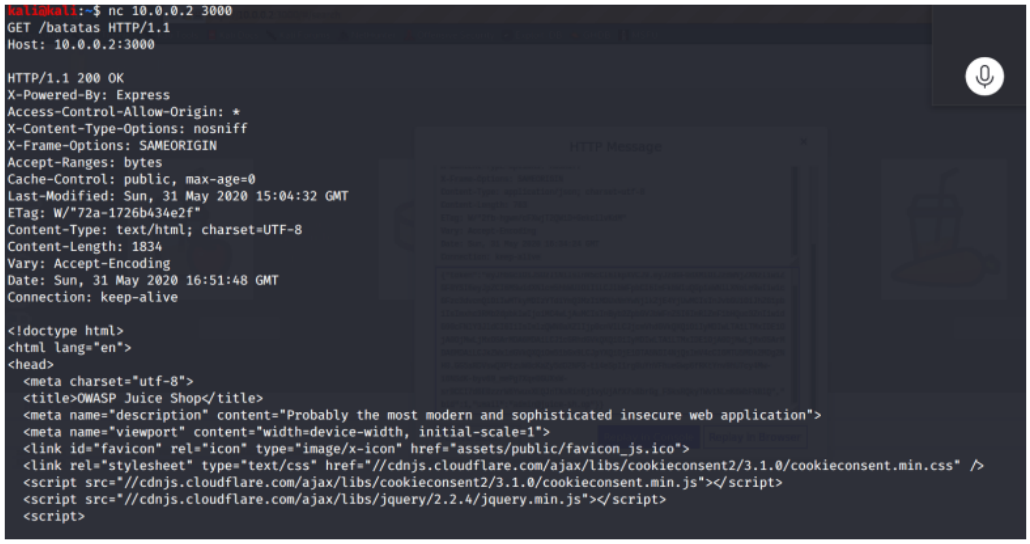

Figure 30: Error Handling 1.

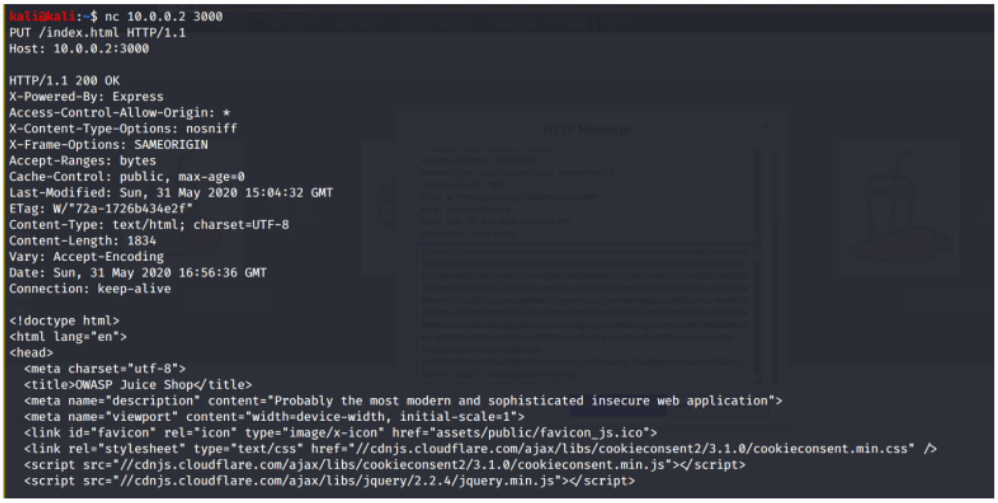

Figure 31: Error Handling 2.

To improve error handling, the application should adhere to industry best practices. This includes returning standard HTTP status codes, providing clear and informative error messages to users without divulging sensitive details, and following consistent error handling patterns throughout the application.

By implementing proper error handling mechanisms, developers can enhance the usability, security, and resilience of the application, ensuring that users receive meaningful error feedback and that potential vulnerabilities or misconfigurations are not obscured.

6.9 Testing for Weak Cryptography

Weak cryptography can lead to data breaches and unauthorized access. Testing for weak cryptography vulnerabilities is crucial to ensure the security of data and prevent unauthorized access. Although no specific tests were performed in this case, several possibilities exist to identify and address potential weaknesses. Here are some common approaches:

- **Cipher Strength Analysis:** Evaluate the strength of cryptographic algorithms and protocols used in the system. This involves assessing their resistance against known cryptographic attacks and vulnerabilities. For example, if the system uses outdated or deprecated algorithms such as DES or MD5, it is important to identify and replace them with more secure alternatives like AES or SHA-256.

- **Key Management Assessment:** Examine the key generation, storage, and distribution processes within the system. Weaknesses in key management can significantly compromise the cryptographic strength. Testing may involve analyzing key generation algorithms, checking for proper key length, assessing key storage mechanisms (e.g., secure key stores or hardware security modules), and verifying secure key exchange protocols.

- **Randomness Analysis:** Cryptographic operations often rely on random numbers for various purposes, such as key generation and initialization vectors. Testing should include assessing the quality of random number generation mechanisms within the system. Weak or predictable random number generation can lead to cryptographic vulnerabilities.

- **Encryption Implementation Review:** Analyze how encryption is implemented within the system. This involves examining encryption libraries, APIs, and code snippets to ensure they are used correctly and securely. Common mistakes to watch out for include improper initialization vector (IV) usage, weak key derivation functions, or insecure modes of operation.

- **Cryptographic Protocol Analysis:** If the system relies on cryptographic protocols (e.g., SSL/TLS, IPsec), review their implementation and configurations. Look for potential vulnerabilities in protocol versions, cipher suites, certificate validation, and negotiation processes.

- **Cryptanalysis Techniques:** If there are specific concerns or suspicions about the strength of the cryptography used, more advanced techniques like

cryptanalysis can be employed. Cryptanalysis involves analyzing the cryptographic algorithms and attempting to identify weaknesses or vulnerabilities through mathematical analysis.

6.10 Business Logic Testing

Insecure business logic can lead to unauthorized access and data breaches. Although no specific tests were performed for business logic vulnerabilities in this case, it is important to recognize the significance of testing and securing the business logic within a system. Insecure business logic can result in unauthorized access and data breaches, making it crucial to address potential vulnerabilities in this area. Therefore, we present briefly below some suggestions for tests to be performed over the business logic of the application in question.

- **Input Validation:** Validate and sanitize user inputs to prevent common vulnerabilities like SQL injection, cross-site scripting (XSS), and command injection. Test the application with various input scenarios, including boundary cases and malicious inputs.
- **Authorization and Access Control:** Review the system's authorization mechanisms and access control policies to ensure they are correctly implemented. Test different user roles and permissions to verify that access is properly restricted and unauthorized actions are prevented.
- **Business Workflow Testing:** Analyze the system's business workflows and test them to identify potential security gaps or logical flaws. This can involve creating test cases that simulate different scenarios and ensure that the desired business logic is enforced correctly.
- **Session Management:** Assess the session management mechanisms within the system to prevent session hijacking or session fixation attacks. Test session handling, expiration, and termination procedures to ensure they align with security best practices.
- **Error Handling:** Evaluate how the system handles errors and exceptions. Test for scenarios where errors might expose sensitive information or provide attackers with valuable insights into the system's behavior.
- **Data Validation and Integrity:** Verify that all incoming and outgoing data is properly validated and that integrity checks are performed to prevent data

tampering. Test for data manipulation scenarios to ensure the system can detect and reject unauthorized modifications.

- **Business Rules Compliance:** Check if the system enforces the specified business rules correctly. Validate that the implemented logic aligns with the intended behavior and that all relevant conditions and constraints are properly evaluated and enforced.

It is important to note that these suggestions provide a starting point for business logic testing. The specific approach may vary depending on the system's complexity, business requirements, and the level of access and information available for testing. Consulting with security professionals and following industry best practices is recommended when conducting business logic testing.

6.11 Client Side Testing

During client-side testing, WebSocket responses were captured by ZAP during a manual scan. These responses aim to verify the persistence of the connection.

WebSockets provide a bidirectional communication channel between a client (usually a web browser) and a server. Unlike traditional HTTP requests, WebSocket connections remain open, allowing real-time data transfer between the client and the server.

By capturing and analyzing WebSocket responses, it is possible to assess the behavior and integrity of the client-server communication. The responses can reveal important information such as the status of the connection, the data being transmitted, and any potential vulnerabilities or misconfigurations.

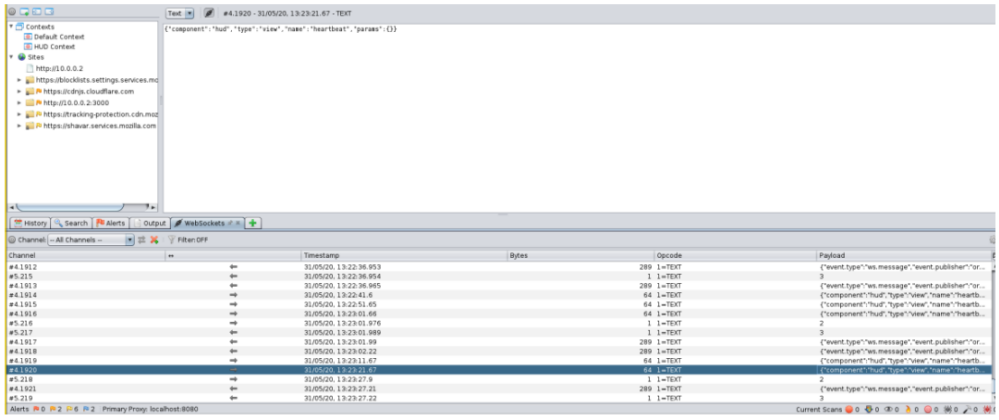

Figure 32: WebSockets.

Client-side testing is crucial for evaluating the security and functionality of the application from the user's perspective. It involves assessing how the application handles client-side technologies such as JavaScript, AJAX requests, cookies, local storage, and WebSockets. This testing ensures that the application interacts correctly with these components, maintains data integrity, and prevents client-side vulnerabilities such as cross-site scripting (XSS) or cross-site request forgery (CSRF).

By examining the WebSocket responses captured during the scan, testers can gain insights into the application's real-time communication and identify any abnormalities or potential security risks. This information can be used to address any issues and improve the overall security and performance of the application.

Overall, client-side testing plays a crucial role in assessing the behavior and security of an application from the user's perspective, and WebSocket responses are an essential component of this evaluation.

6.12 Vulnerability Overview

The OWASP Juice Shop application exhibits vulnerabilities across multiple security categories, including injection attacks, Cross-Site Scripting (XSS), broken authentication and session management, broken access control, security misconfiguration, insecure cryptographic storage, and insufficient logging and monitoring.

Injection attacks: The application is susceptible to SQL injection, NoSQL injection, and command injection attacks, which can allow malicious actors to manipulate or execute unauthorized commands within the application's database or system.

Cross-Site Scripting (XSS): The application is vulnerable to both reflected and stored XSS attacks. These vulnerabilities enable attackers to inject malicious scripts into the application, potentially leading to unauthorized access, data theft, or other malicious activities.

Broken Authentication and Session Management: The application has weaknesses related to authentication and session management. These include weak password policies, insecure session management, and the potential for session fixation attacks, where an attacker can hijack a user's session.

Broken Access Control: The application exhibits vulnerabilities in access control mechanisms. This includes insecure direct object references, vertical and horizontal privilege escalation, and broken access control. These flaws can allow unauthorized users to access restricted resources or perform actions beyond their authorized privileges.

Security Misconfiguration: The application suffers from security misconfigurations, such as default credentials, insecure file and directory permissions, and insecure SSL/TLS configurations. These misconfigurations expose the application to potential unauthorized access or data breaches.

Insecure Cryptographic Storage: The application stores sensitive information, such as passwords and credit card data, in an insecure manner. This can potentially lead to unauthorized access and data compromise.

Insufficient Logging and Monitoring: The application lacks proper logging and monitoring capabilities, making it challenging to detect and respond to security incidents effectively. This deficiency hampers the ability to identify and mitigate potential threats or attacks in a timely manner.

7 Web Application Security Firewall

This section focuses on the implementation and configuration of a web application firewall (WAF) to enhance the security of the web application.

In the provided illustration, we can see the alerts generated during an automated attack without any web application firewall (WAF) protection in place. These alerts indicate potential security vulnerabilities and malicious activities targeting the web application.

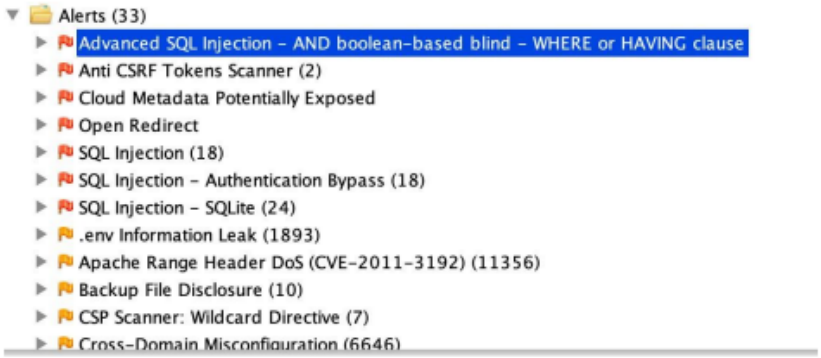

Figure 33: Alerts / WAF Disabled.

The effectiveness analysis of the WAF involved conducting another automated attack after implementing the WAF. The purpose was to assess whether the WAF successfully detected and mitigated the attack attempts, reducing the number of alerts and potential security risks.

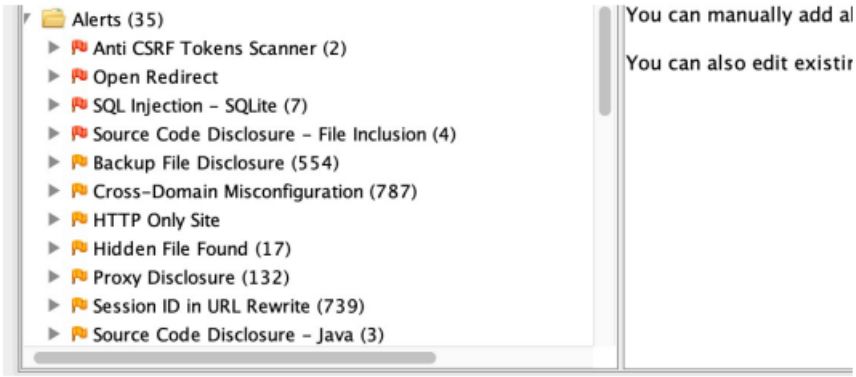

Figure 34: Alerts / WAF Enabled.

By analyzing the alerts generated during the attack with the WAF enabled, it allows for evaluating the WAF's ability to identify and respond to different types of attacks. The alerts serve as indicators of potentially malicious activities, such as suspicious request patterns, known attack signatures, or abnormal behavior.

The effectiveness of a WAF can be assessed based on several factors, including the accuracy of attack detection, the ability to block or filter malicious requests, and the capability to minimize false positives and negatives. False positives occur when legitimate requests are mistakenly identified as malicious, while false negatives happen when actual attacks go undetected.

During the analysis, it is important to review the nature of the alerts, such as the types of attacks detected (e.g., SQL injection, cross-site scripting, etc.), the severity levels assigned to the alerts, and the actions taken by the WAF to mitigate the attacks. Additionally, the comparison between the number of alerts before and after the implementation of the WAF can provide insights into its effectiveness in reducing potential security risks.

The WAF effectiveness analysis helps in fine-tuning the WAF configuration, adjusting rule sets, and refining security policies. It enables the security team to strengthen the application's defense mechanisms by actively mitigating known attack vectors and adapting to emerging threats.

Overall, the WAF effectiveness analysis provides valuable information about the WAF's ability to protect the web application against various attack vectors. It assists in improving the application's security posture, enhancing incident response capabilities, and ensuring a more robust defense against potential security breaches.

Alerts	Level
SQL Injection Authentication Bypass	High and Filtered
SQL Injection in Login Forms	High and Filtered
Advanced SQL Injection and Boolean – Based blind - WHERE or HAVING clause	High and Filtered
Cloud metadata potentially exposed	High and Filtered
.env information leak	Before, Medium and Information Level,now only Information Level
Trace .axd information leak	Before, Medium and Information Level,now only Information Level
Apache Range Header DoS (CVE-2011-3192)	Medium and Filtered
ELMAH Information Leak	Medium for Information
CSP Scanner: Wildcard Directive	Medium and Filtered

In the provided table, we can identify the vulnerabilities that were filtered by the Web Application Firewall (WAF). The WAF effectively detected and mitigated these vulnerabilities, providing an additional layer of security for the web application.

To demonstrate the effectiveness of the WAF in filtering SQL Injection attacks, a test was conducted. An attempt was made to access a JuiceShop account using a SQL Injection attack. However, due to the presence of the WAF, the login was not allowed, indicating that the WAF successfully recognized and blocked the malicious attempt.

Figure 35: Refused Login.

By examining the server logs of Apache, it is evident that the WAF was able to identify the SQL Injection attack. This highlights the proactive nature of the WAF in monitoring incoming requests, analyzing their patterns, and applying security rules to detect and prevent common attack techniques.

Figure 36: Apache Error Log.

Figure 37: modsec_audit Log.

The ability of the WAF to filter out SQL Injection attacks is crucial in safeguarding the web application's database from unauthorized access, data breaches, and potential manipulation. SQL Injection attacks involve exploiting vulnerabilities in the application's input validation mechanisms to execute malicious SQL statements, which can lead to data leakage, data loss, or even complete system compromise.

The logs generated by the Apache server provide valuable insights into the WAF's effectiveness and its ability to detect and block specific attack patterns. These logs can be reviewed and analyzed to gain a better understanding of the types of attacks mitigated by the WAF and to fine-tune its configuration for optimal protection.

Overall, the successful identification and blocking of SQL Injection attacks by the WAF demonstrate its role as a critical security component in defending against common web application vulnerabilities. The WAF acts as a gatekeeper, continuously monitoring and analyzing incoming traffic, filtering out potentially malicious requests, and ensuring the integrity and security of the web application and its underlying data.

We have consolidated the previous tests into a single section for the sake of simplicity and because some vulnerabilities were not possible to fix. By grouping the tests together, we aim to provide a comprehensive overview of the security assessment conducted on the web application.

It is important to note that despite our best efforts, certain vulnerabilities could not be fully addressed or remediated due to limitations in the application or the technologies used. These unresolved vulnerabilities may pose ongoing risks to the security of the web application.

However, by combining the findings from the various tests, we have gained valuable insights into the overall security posture of the application. This consolidated approach allows us to identify patterns, prioritize remediation efforts, and develop a more holistic understanding of the potential risks and areas for improvement.

While it is ideal to address all vulnerabilities identified during the assessment, it is essential to acknowledge the challenges and constraints that may prevent immediate fixes. It is recommended to establish a risk management strategy that includes ongoing monitoring, regular security updates, and proactive measures to mitigate the impact of unresolved vulnerabilities.

In summary, despite some remaining vulnerabilities that were not feasible to fix, the consolidation of previous tests provides a comprehensive view of the web application's security status. This consolidated approach allows for a better understanding of the overall risk landscape and enables stakeholders to make informed decisions regarding the application's security and future improvements.

8 Conclusion

In conclusion, the implementation of a Web Application Firewall (WAF) has proven to be an effective security measure in our project. The WAF demonstrated its capability to intercept and block harmful attacks, particularly SQL Injection, which poses a significant threat to web applications by exploiting input validation vulnerabilities.

By actively monitoring and analyzing incoming requests, the WAF successfully identified and mitigated known attack patterns, safeguarding the application from unauthorized access, data breaches, and manipulation. This is especially crucial when considering the potential risks associated with compromising sensitive data, such as administrator login credentials and the integrity of database tables.

While the WAF excelled in protecting against SQL Injection attacks, it's important to note that vulnerabilities related to the implementation of the web page require code-level modifications rather than relying solely on a firewall. Nevertheless, the WAF proved its value by significantly reducing the number of vulnerabilities and alerts detected during automatic attack simulations conducted with the WAF enabled between ZAP and JuiceShop.

Furthermore, the WAF's ability to actively analyze and filter incoming SQL queries played a vital role in maintaining the integrity and security of the application's data. By identifying suspicious patterns and blocking potentially harmful queries, the WAF ensured that only valid and secure SQL statements reached the database.

It's important to recognize that the WAF should not be considered a standalone solution, but rather one element of a comprehensive security strategy. Combining the WAF with other security measures such as secure coding practices, regular vulnerability assessments, and patch management will provide a stronger defense against a wide range of threats.

In summary, the implementation of the Web Application Firewall has proven to be an effective security measure, enhancing the overall security posture of the web application. By mitigating SQL Injection attacks and reducing vulnerabilities, the WAF has contributed significantly to the protection of sensitive data and the prevention of unauthorized access. Continued attention to security best practices and a multi-layered approach will ensure the ongoing resilience and protection of the web application against evolving threats in the future.

References

[1] OWASP. (2021). OWASP Juice Shop. Retrieved from `https://owasp.org/www-project-juice-shop/`

[2] Halfond, W. G. J., Viegas, J., Orso, A., Manolios, P., & Liblit, B. (2006). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis* (pp. 165-175).

[3] Saleh, M., Alazab, M., Zhou, Y., & Zhang, Y. (2019). A comprehensive survey on cross-site scripting: Attack vectors, impacts, and countermeasures. *Journal of Network and Computer Applications*, 144, 1-23.

[4] Granjal, J. (2018). *Segurança Prática em Sistemas e Redes com Linux*. FCA - Editora de Informática.