

Jurassic Island Web Application Penetration Testing Report PWH Capstone

PUBLIC USE

This document is public use as considered by the owner. The document herein contains unprivileged information regarding a web application that is being hosted locally, intended for testing. Beyond this page, the report will emulate a real penetration testing report, including the verbiage and footer notes. Please keep in mind this document is considered public and not confidential.

Date: April 25, 2024

Table of Contents

Table of Contents.....	2
Confidentiality Statement.....	4
Disclaimer.....	4
Contact Information.....	4
Assessment Overview	5
Assessment Components.....	5
Web Application Penetration Test.....	5
Severity Ratings.....	6
Risk Factors	6
Likelihood.....	6
Impact	6
Scope.....	7
Scope Exclusions	7
Client Allowances.....	7
Executive Summary.....	8
Scoping and Time Limitations	8
Testing Summary	8
Tester Notes and Recommendations.....	9
Key Strengths and Weaknesses	9
Vulnerability Summary.....	10
Technical Findings.....	11
Web Application Penetration Test.....	11
Finding WPT-001: Broken Object Level Authorization (BOLA) / Insecure Direct Object Reference (IDOR) (Critical)	11
Finding WPT-002: Weak Cryptographic Secret (Critical)	13
Finding WPT-003: Race Condition Present (Critical).....	14
Finding WPT-004: Broken Access Control (High)	16
Finding WPT-005: Cross-Site Scripting (XSS/JavaScript Injection) Possible (High)	18
Finding WPT-006: No Content Security Policy (CSP) Set (High)	20
Finding WPT-007: Lack of Multi-Factor Authentication (MFA) Support (Moderate)	21

Finding WPT-008: Outdated Software Libraries (Moderate).....	22
Finding WPT-009: No Cross-Site Request Forgery (CSRF) Headers or Tokens (Moderate)	24
Finding WPT-010: Lack of Rate Limiting (Low).....	26
Finding WPT-011: Lack of TLS/SSL Encryption (Informational).....	28
Additional Scans and Reports	29

Confidentiality Statement

This document is the exclusive property of Jurassic Island and SixSecure. This document contains proprietary and confidential information regarding the security state of Jurassic Island's web application. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Jurassic Island and SixSecure. Jurassic Island may share this information with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment period and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for full evaluation of all security controls. SixSecure prioritized the assessment to identify the weakest security controls and vulnerabilities an attacker may exploit. SixSecure recommends conducting similar assessments on at least an annual basis by internal or third-party assessors to ensure the continued success of security controls.

Contact Information

Name	Title	Contact Information
<i>Jurassic Island</i>		
N/A	N/A	N/A
<i>SixSecure</i>		
Ryan Sapone	Course Taker	Email: N/A

Assessment Overview

From April 23, 2024 through April 25, 2024, Jurassic Island engaged SixSecure to evaluate the security posture of its web application compared to industry best practices, which included a penetration test against the Jurassic Island web application and API components. All testing performed was based on the *NIST SP 800-115 Technical Guide to Information Security Testing and Assessment*, *OWASP Web Application Security Testing Guide*, and customized testing frameworks.

The phases of a web application penetration test include the following:

- Reconnaissance – collecting information about the target web application, its infrastructure, and potential vulnerabilities to inform the testing process
- Scanning – actively probing the web application for vulnerabilities using manual and automated tools to discover open ports, identify services, and detect common security issues
- Vulnerability Assessment – analyzing collected data to identify specific vulnerabilities within the web application, assessing the security of its code, configuration settings, and third-party components
- Exploitation – attempting to exploit identified vulnerabilities to gain unauthorized access or manipulate the application's behavior, validating the severity of issues and assessing potential impact
- Reporting – compiling findings into a comprehensive report with detailed descriptions of identified vulnerabilities, their potential impact, and recommendations for remediation

Assessment Components

Web Application Penetration Test

A web application penetration test is a strategic assessment aimed at identifying and mitigating potential security risks in an online platform. This involves a thorough evaluation of the application's architecture, code, functionality, and security features. The goal is to simulate a real-world cyber threat, such as unauthorized access or data breaches in order to ensure the web application is resilient against attacks. The results provide a clear understanding of potential vulnerabilities and actionable insights to enhance the security of the web application and safeguard sensitive data to maintain users' trust.

Severity Ratings

The following table defines levels of severity and the corresponding CVSS score range that are used throughout this document to assess vulnerability and risk impact:

Severity	CVSS V4 Score Range	Definition
Critical	9.0 – 10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0 – 8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0 – 6.9	Vulnerabilities exist but are not exploitable, or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.0 – 3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Risk Factors

Risk is measured by two factors: likelihood and impact.

Likelihood

Measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the availability of tools, attacker skill level, and client environment.

Impact

Measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Scope

Assessment	Details
Web Application Penetration Test	Jurassic Island web application

Scope Exclusions

There were no scope exclusions mentioned prior to the engagement.

Client Allowances

Jurassic Island provided SixSecure the following allowances:

- A full local copy of the web application via Docker files

Executive Summary

SixSecure evaluated the Jurassic Island web application over the course of 48 hours, performing reconnaissance, scanning, vulnerability assessment, and exploitation. SixSecure found that there were numerous security flaws within the web application that had a potential for loss of profit, reading other users' purchase histories, broken business functionality, and weak cryptography. The web application does not follow industry best practices and guidelines for security, namely having outdated and vulnerable programming libraries, weak encryption secrets, and a lack of injection prevention mechanisms.

Although some attacks were not possible or had minimal impacts, these underlying security issues should be addressed as soon as possible in order to reduce the web application's overall risk. This penetration test revealed a total of three (3) critical severity issues, three (3) high severity issues, three (3) medium severity issues, and one (1) low severity issue.

Scoping and Time Limitations

The scope and time of the engagement was limited to the Jurassic Island web application and related API components, and within the span of 48 total hours. There was no scoping direction given prior to the engagement.

Testing Summary

SixSecure initially built the Docker container from the provided files in order to spin up the Jurassic Island web server and components locally. Once spun up, we tested the application to understand its functionality. After creating and verifying a user account, we noticed that the verification link had an ID of 1. We then tested by creating a second user account, which had the verification ID of 2, signaling a very low entropy UUID.

We authenticated to the application and tested the functionality after being logged in. The main thing we noticed was the ability to add items to your basket, and then proceed to checkout. All of this traffic was proxied through to Burp Suite, which is where we noticed the request for purchase history when in your basket. Seeing an *id* parameter, we later tested for both SQL injection as well as IDOR, confirming that IDOR was present. We also looked at the JSON web token, which was tested for mass assignment and other secrets issues, which yielded two separate issues:

1. the secret for the token was too weak and was compromised
2. no cookie was needed to view purchase histories

Because we did not locate an administrative portal or functionality, mass assignment testing did not yield any further compromise, nor did compromising the token's secret key.

We continued testing the application's input fields, which did not yield any avenues for cross-site scripting (XSS). However, we did find two issues during this testing, the first being that we could inject HTML and JavaScript into the */basket* page. This injection allowed us to pop an alert box confirming XSS, but also injecting HTML, opening an avenue for clickjacking and other social engineering attacks. The other issue tested was a race condition within the */checkout* page. It was found that a race condition vulnerability allowed us to obtain products without payment.

Other issues were noted and tested for throughout the duration of the engagement, including rate limiting, multi-factor authentication, and server-side template injection among others. The overall state of the application is in poor condition, and we recommend retesting at a later point.

Tester Notes and Recommendations

The testing results of this engagement show that there are many changes needed to the web application. There is a high risk to confidentiality, availability, and integrity for users' data, along with a high potential to lose the company profits based on the race condition. Numerous other, smaller issues are also present, where we would recommend frequent retesting of the web application.

Key Strengths and Weaknesses

The following highlights the key strengths of the Jurassic Island web application identified during the assessment:

1. No SQL injection was found possible
2. Error messages not overly verbose
3. No information leakage in HTML source code

The following highlights the key weaknesses of the Jurassic Island web application identified during the assessment:

1. Broken business logic in verifying user accounts
2. Information leakage via BOLA/IDOR vulnerability
3. Outdated software libraries in use
4. Lack of CSP and anti-CSRF tokens
5. Weak JWT secret
6. Broken access control when viewing purchase history

Vulnerability Summary

Critical	High	Moderate	Low	Informational
3	3	3	1	1

Finding	Severity	Explanation
Web Application Penetration Test		
<i>WPT-001: Broken Object Level Authorization (BOLA) / Insecure Direct Object Reference (IDOR)</i>	Critical	Unauthorized users can access restricted functionalities or sensitive data due to insufficient access controls
<i>WPT-002: Weak Cryptographic Secret</i>	Critical	The JWT uses a weak and easily compromised secret
<i>WPT-003: Race Condition Present</i>	Critical	Allows a malicious user to add more items to the basket during checkout, obtaining free products
<i>WPT-004: Broken Access Control</i>	High	Users are not required to verify user accounts, and users do not need a JWT to see purchase histories
<i>WPT-005: Cross-Site Scripting (XSS/JavaScript Injection) Possible</i>	High	One web page allows JavaScript injection into the URL, allowing malicious JavaScript
<i>WPT-006: No Content Security Policy (CSP) Set</i>	High	The lack of a CSP helps allow XSS and other attacks without restricting which content should be trusted
<i>WPT-007: Lack of Multi-Factor Authentication (MFA)</i>	Moderate	MFA is not implemented, reducing the overall security of the application
<i>WPT-008: Outdated Software Libraries</i>	Moderate	Numerous vulnerabilities found with the application's frameworks and languages
<i>WPT-009: No Cross-Site Request Forgery (CSRF) Headers or Tokens</i>	Moderate	Absence of CSRF headers and tokens makes the application's forms susceptible to CSRF attacks
<i>WPT-010: Lack of Rate Limiting</i>	Low	Absence of rate limiting measures allows attackers to perform brute force attacks with minimal resistance
<i>WPT-011: Lack of TLS/SSL Encryption</i>	Informational	The application is run locally, but there is no encryption mechanism for data

Technical Findings

Web Application Penetration Test

Finding WPT-001: Broken Object Level Authorization (BOLA) / Insecure Direct Object Reference (IDOR) (Critical)

Description:	Users are able to view the purchase histories of other users.
Risk:	Likelihood: Very High Impacts: Confidentiality

Description

The initial email verification upon registering had a link with a unique user ID (UUID) of 1. Registering for a second account had the UUID of 2. We also saw a *GET* request to the */purchase-history* page with an *id* parameter anytime we visited the */basket* page for our current user, and after modifying the request we saw that we could access the purchase history of another user.

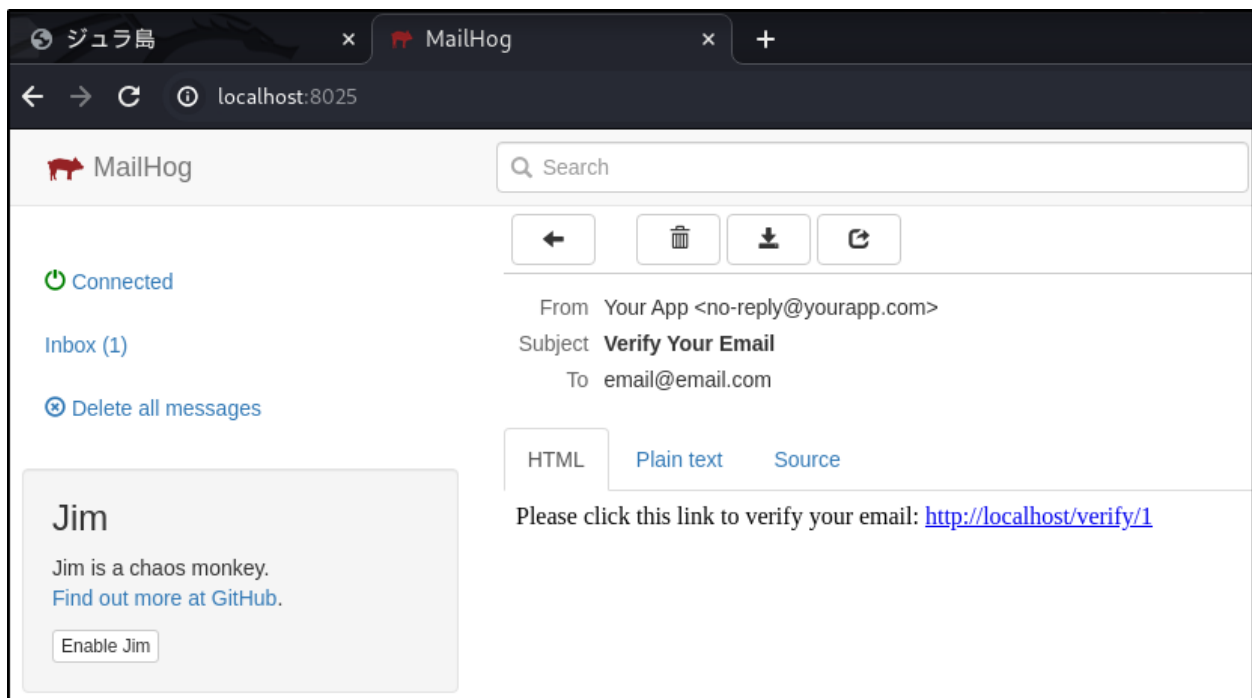


Figure 1 - email verification with the UUID of 1

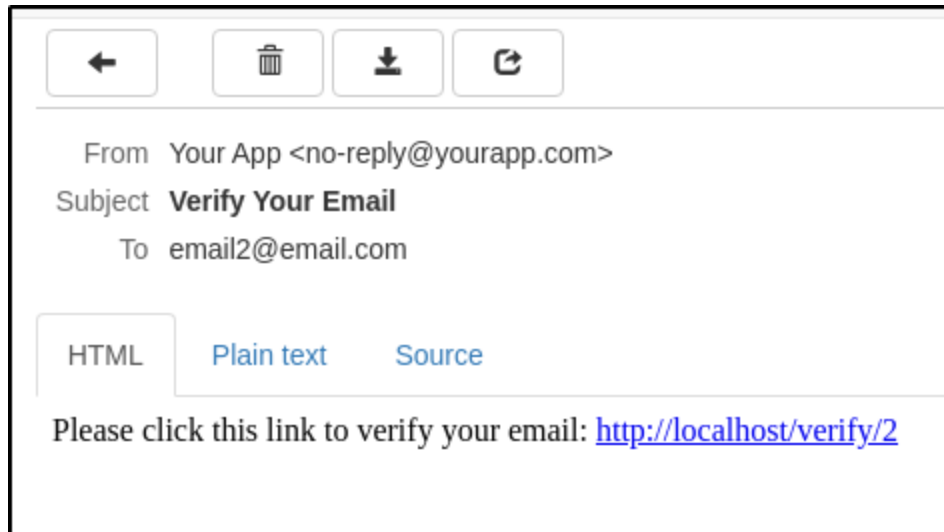


Figure 2 - second user's UUID of 2

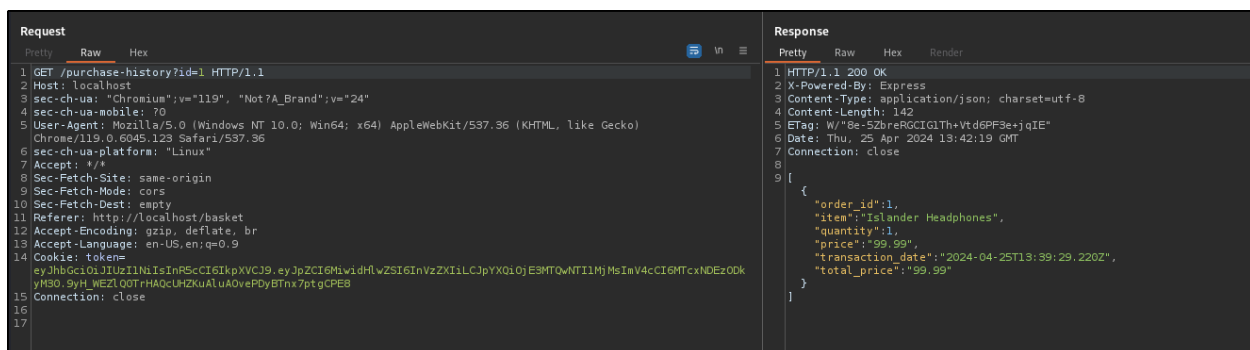


Figure 3 - modifying the GET request's id parameter to see another user's history

Remediation

Ensure proper access controls are in place to prevent unauthorized users from performing actions that are not intended. Implement stronger authorization for the API endpoints to only allow a user to read their own data.

- <https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/>

Finding WPT-002: Weak Cryptographic Secret (Critical)

Description:	The session cookie is a JSON Web Token with a weak secret key, allowing an attacker to modify or create forged tokens.
Risk:	Likelihood: Very High Impacts: Confidentiality, Integrity, Availability

Description

While testing the JWT, we were able to break the secret encryption key to alter and create new tokens. Because of limited administrator functionality, mass assignment testing was not performed.

```
(kali@OCTANE)-[~/Tools/jwt_tool]
$ ./jwt_tool.py 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiOvePDyBTnx7ptgCPE8' -C -d /usr/share/wordlists/seclists/SecLists-mas



Version 2.2.6 @ticarpi

Original JWT:
[+] secret is the CORRECT key!
You can tamper/fuzz the token contents (-T/-I) and sign it using:
python3 jwt_tool.py [options here] -S hs256 -p "secret"
```

Figure 4 - using a local tool to identify the encryption key for the JWT

Remediation

Generate a longer and more cryptographically secure key and perform periodic key rotations to mitigate JWT secret attacks.

- <https://cwe.mitre.org/data/definitions/1390.html>

Finding WPT-003: Race Condition Present (Critical)

Description:	A vulnerability allows an attacker to add additional items to the basket at the same time as checkout, allowing an attacker to obtain products without payment.
Risk:	Likelihood: Very High Impacts: Integrity, Availability

Description

Capturing the POST request to checkout with all of the billing information and sending numerous POST requests to `/shop/basket/add` at the same time exploits this race condition. Similar testing was done to the `/shop/basket/update` endpoint but the race condition was not present there.

Purchase History			
Order ID: 6 - Total: \$179.94 (4/25/2024)			
Item	Price	Quantity	Subtotal
Explorer Tshirt	\$29.99	24	\$719.76

Figure 5 - obtaining \$719.76 worth of product for \$179.94

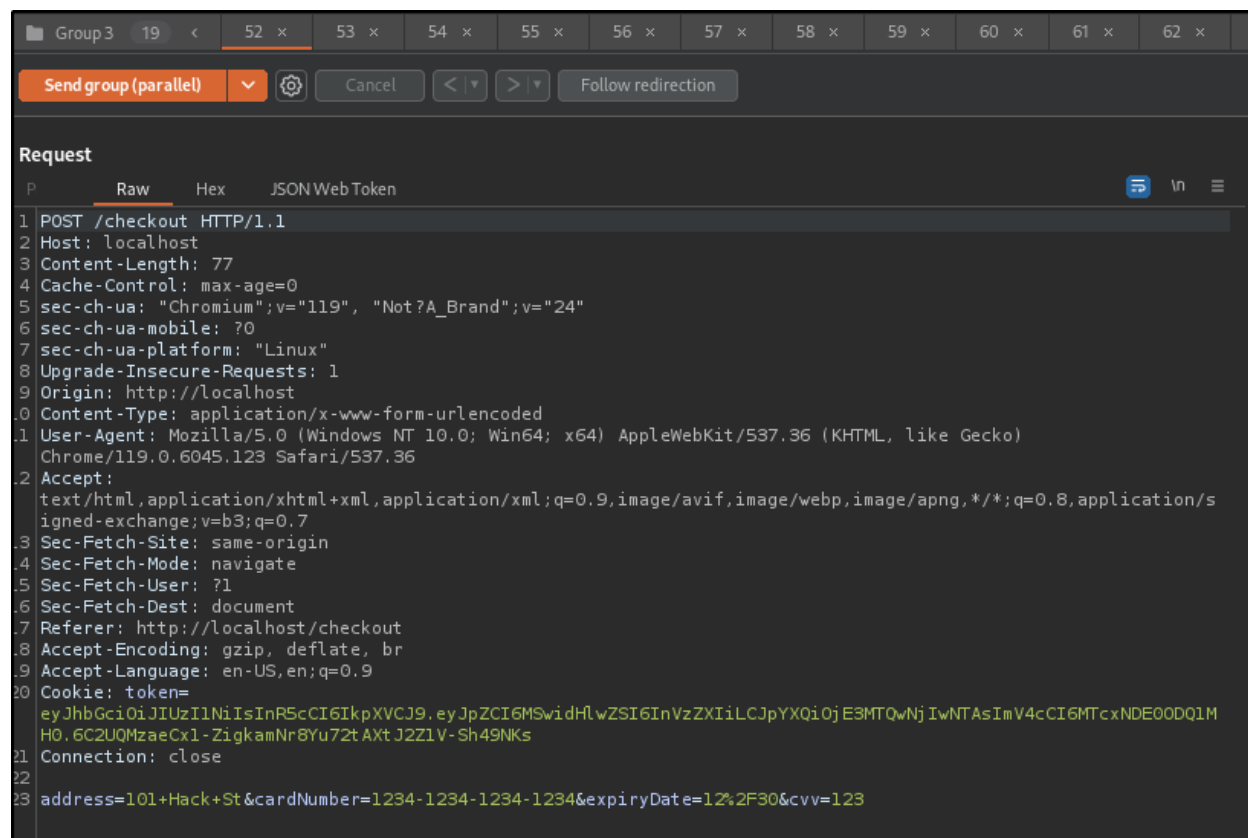


Figure 6 - the POST request to complete checkout, being sent in parallel

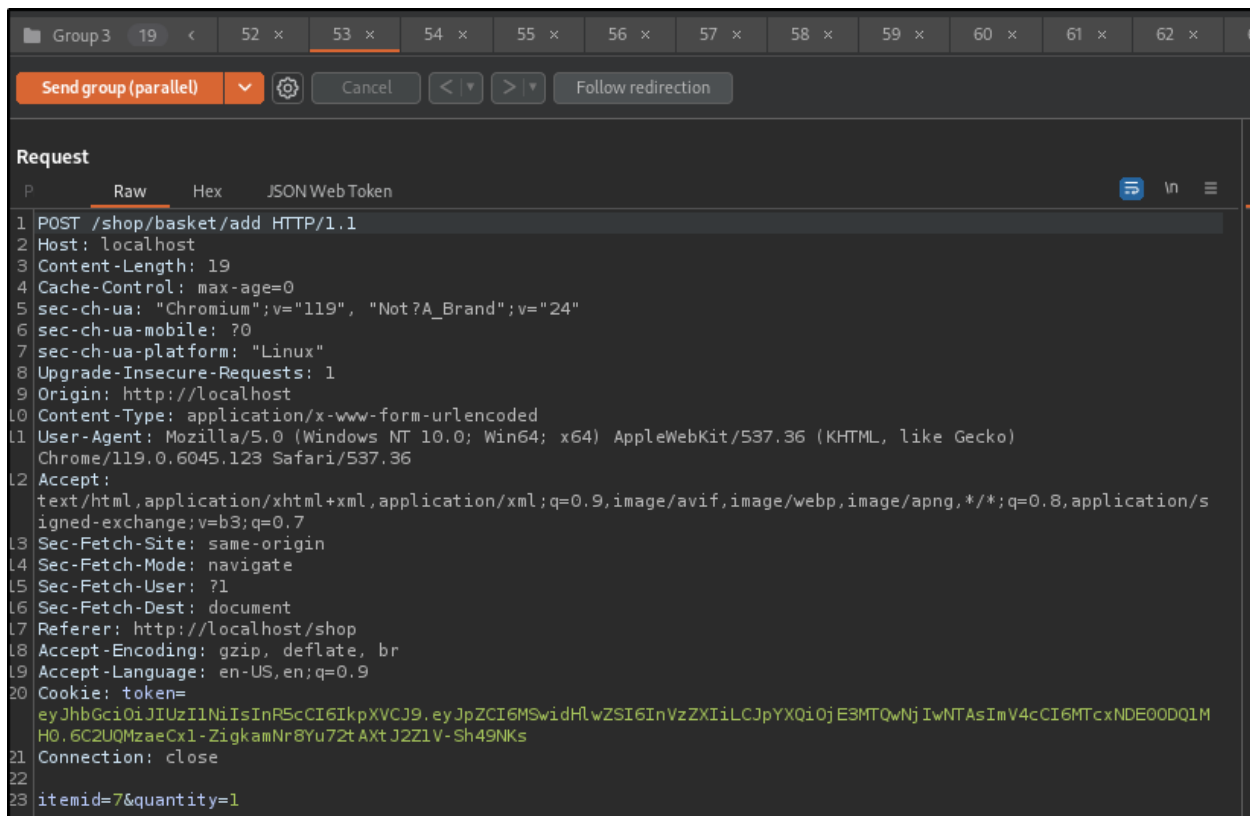


Figure 7 - the POST requests to add more items to the basket, being sent in parallel with the checkout request in figure 6

Remediation

Implement a locking mechanism similar to other transactional queries to ensure that the basket and the checkout are mutually exclusive functions. Additionally, it's best to validate the cart's contents before finalizing the checkout to ensure information integrity.

- <https://cwe.mitre.org/data/definitions/367.html>
- <https://cwe.mitre.org/data/definitions/362.html>

Finding WPT-004: Broken Access Control (High)

Description:	User accounts do not need to be verified after registration. Additionally, a session cookie is not needed in order to see purchase histories when combined with <i>WPT-001</i> .
Risk:	Likelihood: High Impacts: Confidentiality

Description

After registering for an account, you can log right in without needing to verify your email. Ensure that this is intended behavior for the application. A session cookie is provided upon authenticating with the web application; however, this cookie is not needed to make a request to */purchase-history*.

Request	Response
<pre>1 GET /purchase-history?id=1 HTTP/1.1 2 Host: localhost 3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24" 4 sec-ch-ua-mobile: ?0 5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.123 Safari/537.36 6 sec-ch-ua-platform: "Linux" 7 Accept: */* 8 Sec-Fetch-Site: same-origin 9 Sec-Fetch-Mode: cors 10 Sec-Fetch-Dest: empty 11 Referer: http://localhost/basket 12 Accept-Encoding: gzip, deflate, br 13 Accept-Language: en-US,en;q=0.9 14 If-None-Match: W/"2-19Fw4VU07kr8CvBl4zaMCqXZ0w" 15 Connection: close 16 17</pre>	<pre>1 HTTP/1.1 200 OK 2 X-Powered-By: Express 3 Content-Type: application/json; charset=utf-8 4 Content-Length: 142 5 ETag: W/"8e-5ZbreRGCIG1Th+Vtd6PF3e+jqIE" 6 Date: Thu, 25 Apr 2024 14:37:05 GMT 7 Connection: close 8 9 { "order_id":1, "item":"Islander Headphones", "quantity":1, "price":"99.99", "transaction_date":"2024-04-25T13:39:29.220Z", "total_price":"99.99" }</pre>

Figure 8 - no cookie while accessing user 1's purchase history

Request	Response
<pre>1 GET /purchase-history?id=2 HTTP/1.1 2 Host: localhost 3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24" 4 sec-ch-ua-mobile: ?0 5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.123 Safari/537.36 6 sec-ch-ua-platform: "Linux" 7 Accept: */* 8 Sec-Fetch-Site: same-origin 9 Sec-Fetch-Mode: cors 10 Sec-Fetch-Dest: empty 11 Referer: http://localhost/basket 12 Accept-Encoding: gzip, deflate, br 13 Accept-Language: en-US,en;q=0.9 14 If-None-Match: W/"2-19Fw4VU07kr8CvBl4zaMCqXZ0w" 15 Connection: close 16 17</pre>	<pre>1 HTTP/1.1 200 OK 2 X-Powered-By: Express 3 Content-Type: application/json; charset=utf-8 4 Content-Length: 416 5 ETag: W/"1a0-dSBKge8PWjY2qXo6SKUk25lbrMc" 6 Date: Thu, 25 Apr 2024 14:37:29 GMT 7 Connection: close 8 9 [{ "order_id":2, "item":"JuraJima Baseball Cap", "quantity":1, "price":"19.99", "transaction_date":"2024-04-25T14:36:51.061Z", "total_price":"89.96" }, { "order_id":2, "item":"Island Keyring", "quantity":1, "price":"9.99", "transaction_date":"2024-04-25T14:36:51.061Z", "total_price":"89.96" }, { "order_id":2, "item":"Explorer Tshirt", "quantity":2, "price":"29.99", "transaction_date":"2024-04-25T14:36:51.061Z", "total_price":"89.96" }]</pre>

Figure 9 - no cookie while accessing user 2's purchase history

Remediation

Implement a robust account verification process that requires users to verify their email addresses or phone numbers upon registration to prevent unauthorized access. Additionally, enforce proper access controls to ensure that purchase histories and other sensitive information are only accessible to authenticated and authorized users. Remediate *WPT-001*.

- https://owasp.org/Top10/A01_2021-Broken_Access_Control/
- <https://cwe.mitre.org/data/definitions/862.html>
- <https://cwe.mitre.org/data/definitions/863.html>

Finding WPT-005: Cross-Site Scripting (XSS/JavaScript Injection) Possible (High)

Description:	XSS was found on the <code>/basket</code> page, enabling a wide range of potential attack vectors.
Risk:	Likelihood: High Impacts: Confidentiality, Integrity, Availability

Description

XSS/JavaScript injection was found to affect the `/basket` page, particularly at the link:
`{webserver}/basket?{error/success}={XSS_payload}`

This was tested with both the success and error parameters, with a valid payload:
`<script>alert(1)</script>`

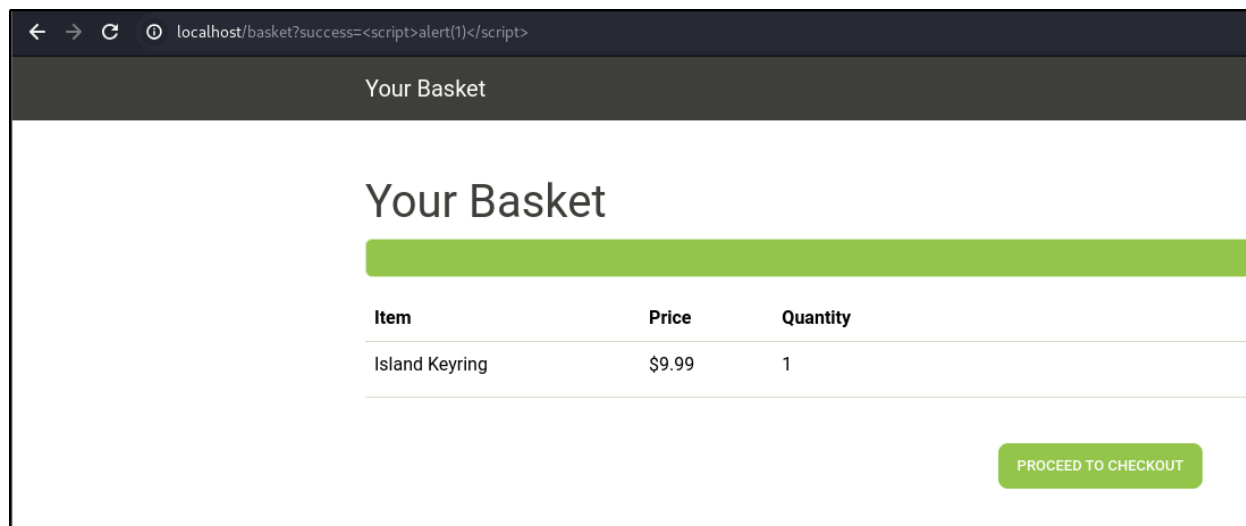


Figure 10 - testing the XSS payload

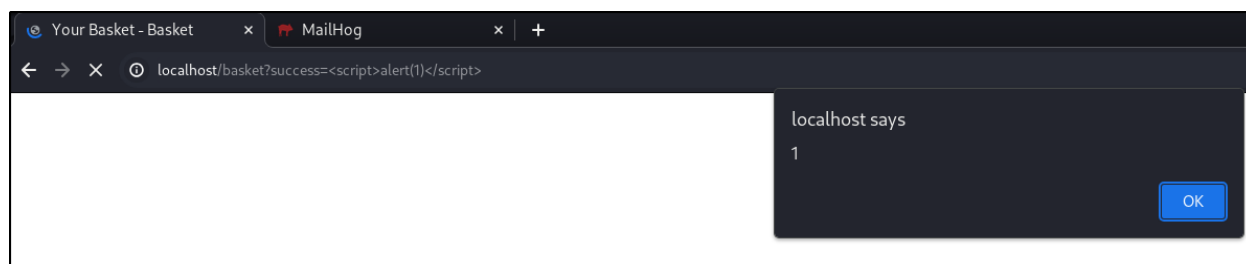


Figure 11 - XSS verified with the success parameter

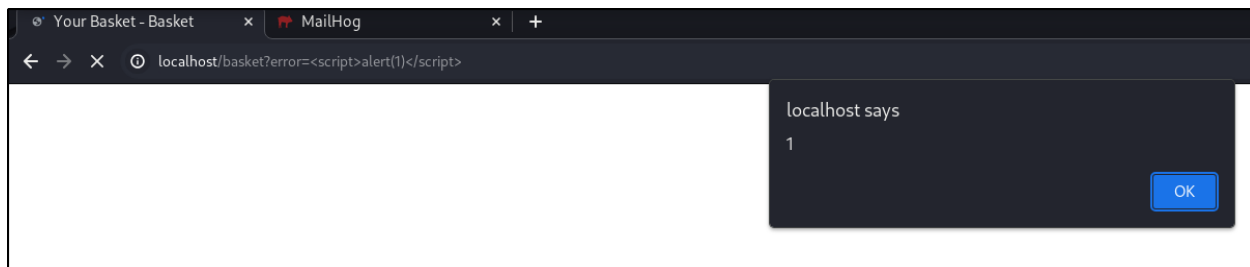


Figure 12 - XSS verified with the error parameter

Remediation

Implement input validation and output encoding to sanitize user inputs on the `/basket` page, preventing malicious JavaScript injection attacks. Utilize CSP headers to restrict the sources of executable scripts, reducing the impact of potential XSS attacks. Regularly update and patch all web application components to mitigate known vulnerabilities and enhance overall security against XSS vulnerabilities.

- <https://cwe.mitre.org/data/definitions/79.html>
- <https://cwe.mitre.org/data/definitions/81.html>
- <https://cwe.mitre.org/data/definitions/86.html>

Finding WPT-006: No Content Security Policy (CSP) Set (High)

Description:	The lack of a CSP allows XSS and other attacks to potentially be available, such as in <i>WPT-005</i> .
Risk:	Likelihood: Very High Impacts: Confidentiality, Integrity, Availability

Description

OWASP's ZAP tool scans for security issues while navigating the web application. A number of alerts were populated, with the most critical being the lack of a CSP. In combination with *WPT-005* and *WPT-008*, this should be resolved as soon as possible.

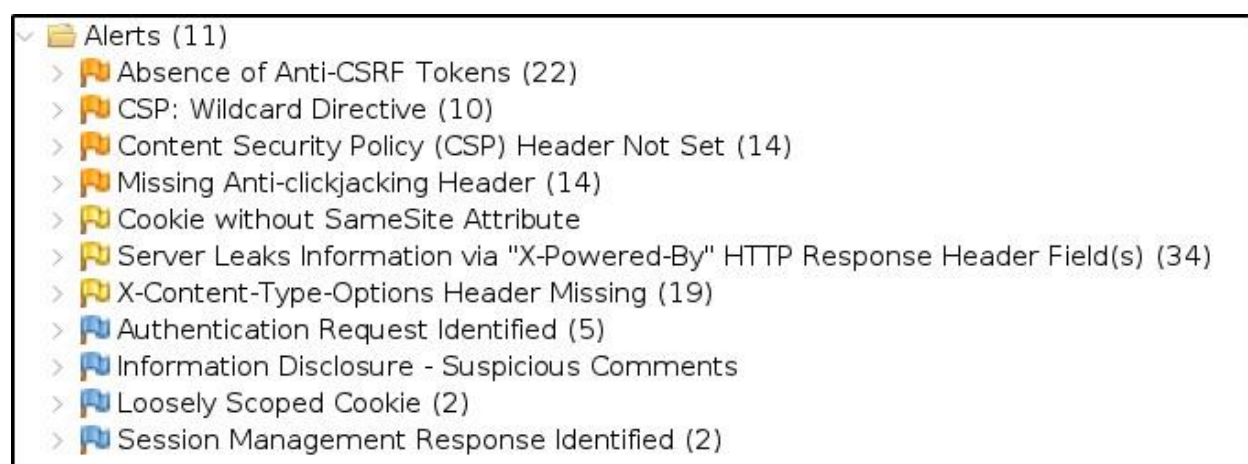


Figure 13 - alerts from OWASP ZAP tool, including the

Remediation

A CSP is a security mechanism that reduces XSS and other attacks by restricting which resources a page can load, including scripts. Implementing a robust CSP and appropriate headers will reduce the likelihood of XSS, clickjacking, and other data injection attacks.

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
- <https://portswigger.net/web-security/cross-site-scripting/content-security-policy>
- <https://cwe.mitre.org/data/definitions/358.html>

Finding WPT-007: Lack of Multi-Factor Authentication (MFA) Support (Moderate)

Description:	The web application does not currently support MFA functionality.
Risk:	Likelihood: High Impacts: Confidentiality, Integrity, Availability

Description

The web application does not support any sort of profiles or configuration for MFA.

Remediation

Implement MFA to enhance user account security. Integrate MFA methods such as SMS-based codes, authenticator apps, or hardware tokens to provide an additional layer of verification during login. Enforce MFA for sensitive actions, such as making purchases.

- <https://cwe.mitre.org/data/definitions/308.html>
- <https://cwe.mitre.org/data/definitions/1390.html>
- <https://cwe.mitre.org/data/definitions/309.html>

Finding WPT-008: Outdated Software Libraries (Moderate)

Description:	Web frameworks and libraries in use were shown to be vulnerable to a myriad of attacks, such as XSS, Denial of Service (DoS), clickjacking, code execution, and more.
Risk:	Likelihood: High Impacts: Confidentiality, Integrity, Availability

Description

Outdated software libraries may be responsible for the poor programming of the web application, making it susceptible to XSS and other high-severity attacks mentioned in this report.

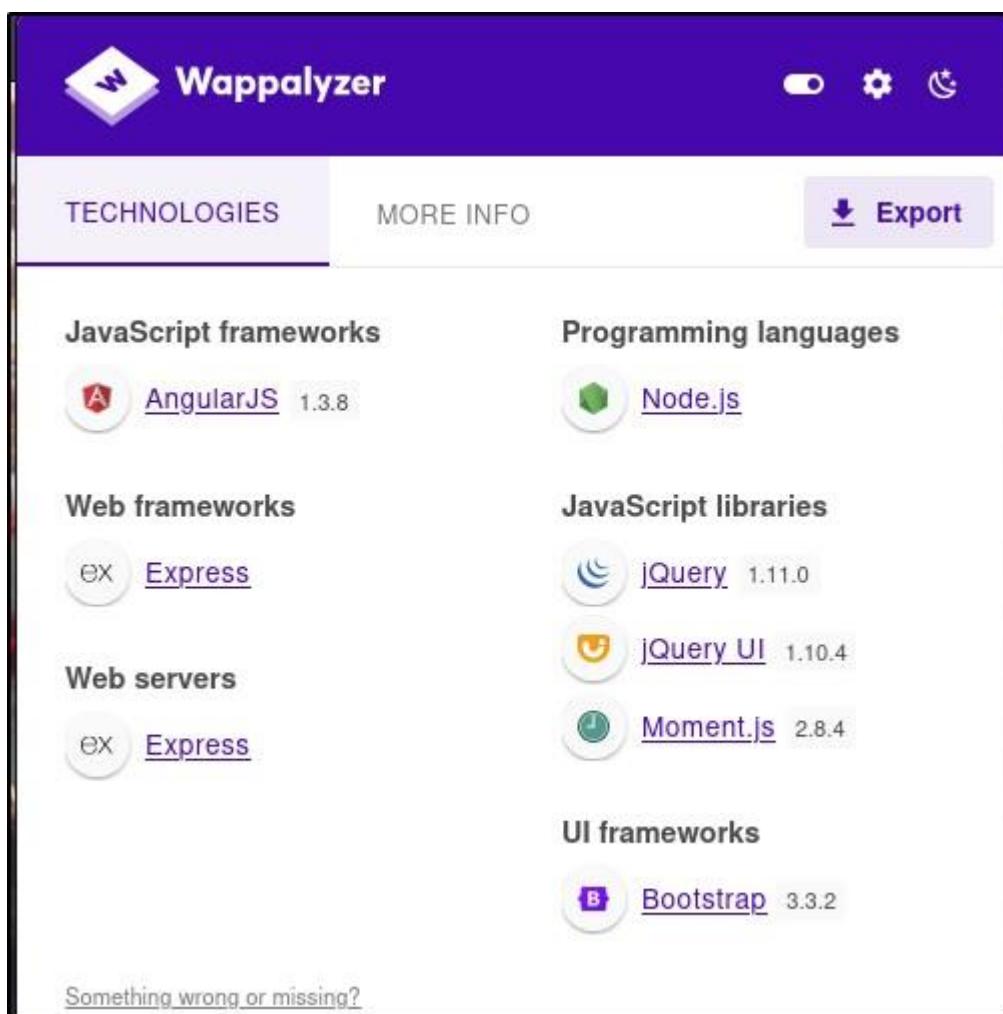


Figure 14 - Wappalyzer showing the technologies in use

Remediation

Regularly update and patch all web frameworks and libraries to the latest secure versions to address known vulnerabilities. Establish a software dependency management process to monitor for security updates and apply patches in a timely manner. Conduct regular vulnerability assessments and penetration testing to identify and remediate potential security risks associated with outdated software components.

- <https://cwe.mitre.org/data/definitions/1395.html>
- [https://owasp.org/Top10/A06_2021-Vulnerable and Outdated Components/](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/)

Vulnerabilities:

- <https://security.snyk.io/package/npm/angular/1.3.8>
- <https://security.snyk.io/package/npm/jquery/1.11.0>
- <https://security.snyk.io/package/npm/jquery-ui/1.10.4>
- <https://security.snyk.io/package/npm/moment/2.8.4>
- <https://security.snyk.io/package/npm/bootstrap/3.3.2>

Finding WPT-009: No Cross-Site Request Forgery (CSRF) Headers or Tokens (Moderate)

Description:	The application does not use anti-CSRF tokens on the forms within the <i>/checkout</i> page or appropriate security headers.
Risk:	Likelihood: Medium Impacts: N/A

Description

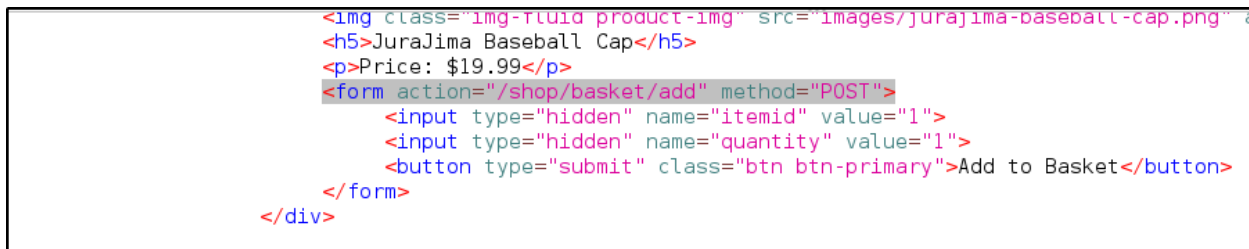
The web application does not make use of anti-CSRF tokens on forms or appropriate security headers to prevent CSRF attacks. Although the impact for this is currently minimal, an attacker may be able to cause a user to carry out an action unintentionally if there are additional features added to this application. This issue was found on multiple forms throughout the web application.

```
<form action="/checkout" method="post">
  <div class="mb-3">
    <label for="address" class="form-label">Address:
    <input type="text" class="form-control" id="ad
  </div>

  <h2>Payment Details</h2>
  <div class="mb-3">
    <label for="cardNumber" class="form-label">Car
    <input type="text" class="form-control" id="ca
```

Absence of Anti-CSRF Tokens
URL: http://localhost/checkout
Risk: 🟡 Medium
Confidence: Low
Parameter:
Attack:
Evidence: <form action="/checkout" method="post">
CWE ID: 352
WASC ID: 9
Source: Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:

Figure 15 - lack of anti-CSRF tokens on the /checkout form



```


<h5>JuraJima Baseball Cap</h5>
<p>Price: $19.99</p>
<form action="/shop/basket/add" method="POST">
  <input type="hidden" name="itemid" value="1">
  <input type="hidden" name="quantity" value="1">
  <button type="submit" class="btn btn-primary">Add to Basket</button>
</form>
</div>

```

Absence of Anti-CSRF Tokens

URL: http://localhost/shop

Risk: 🟡 Medium

Confidence: Low

Parameter:

Attack:

Evidence: <form action="/shop/basket/add" method="POST">

CWE ID: 352

WASC ID: 9

Source: Passive (10202 - Absence of Anti-CSRF Tokens)

Input Vector:

Description:

Figure 16 - lack of anti-CSRF tokens on the /shop/basket/add form

Remediation

Implement CSRF protections by integrating anti-CSRF tokens into forms on the /checkout page and other sensitive areas of the application. Additionally, enforce strict CSRF security headers, such as SameSite cookie attributes and CSRF tokens to prevent unauthorized actions. Regularly review and test the CSRF protection mechanisms to ensure they are effectively mitigating CSRF attacks across the web application.

- <https://cwe.mitre.org/data/definitions/352.html>

Finding WPT-010: Lack of Rate Limiting (Low)

Description:	Brute force attempts against the <code>/login</code> page were possible and were not limited or mitigated by rate limiting tools or CAPTCHAs.
Risk:	Likelihood: High Impacts: Confidentiality

Description

We were able to send over 100 login requests in a short period of time to the application without being rate limited.

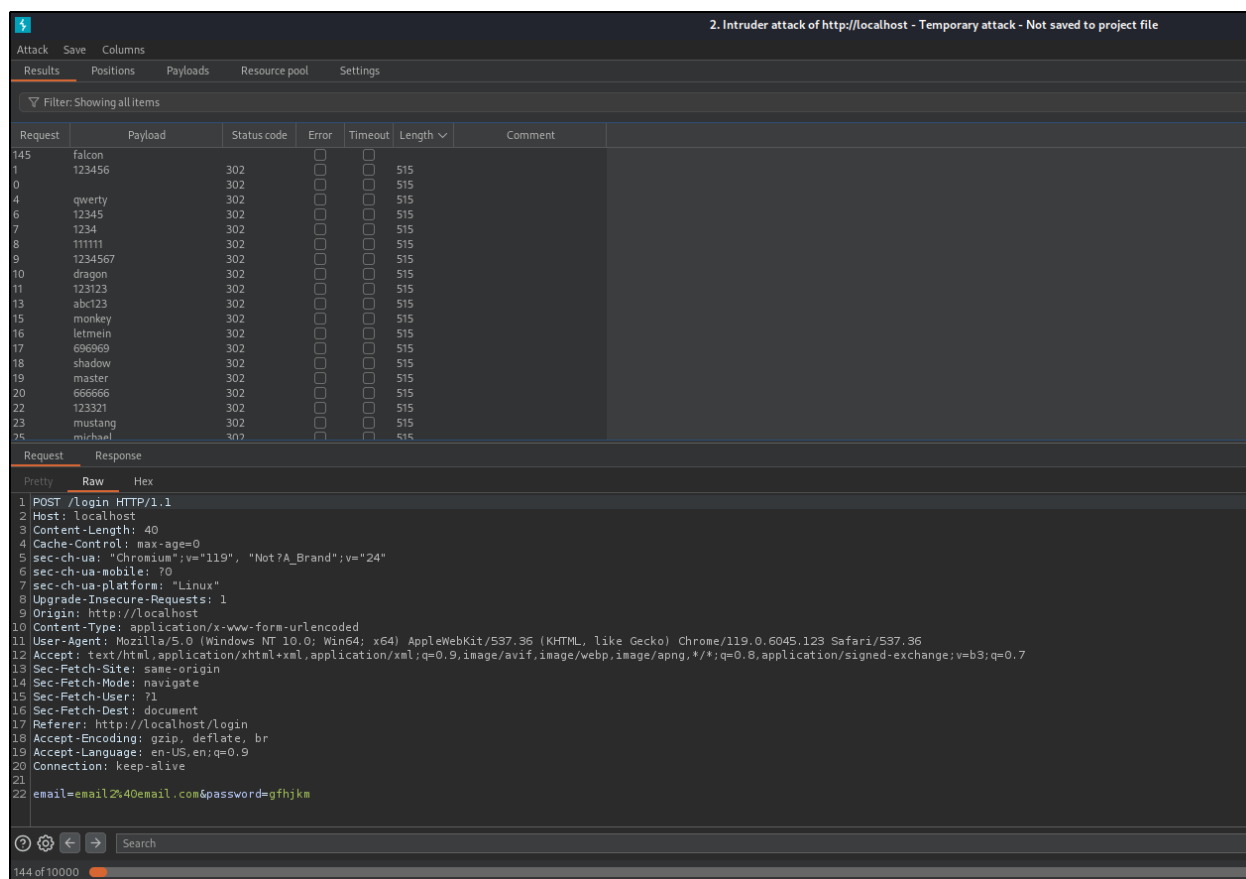


Figure 17 - Burp Suite sending many requests per second

Remediation

Implement rate limiting on the */login* page to mitigate brute force attacks and unauthorized access attempts. Configure rate limiting rules to restrict the number of login attempts per user or IP address within a specified time frame. Consider implementing CAPTCHA challenges after multiple failed login attempts to further deter automated attacks.

- <https://cwe.mitre.org/data/definitions/770.html>
- <https://cwe.mitre.org/data/definitions/799.html>

Finding WPT-011: Lack of TLS/SSL Encryption (Informational)

Description:	The application is running locally, but does not use any secure means of communication.
Risk:	Likelihood: N/A Impacts: N/A

Description

The web application in its current state does not have any encryption mechanisms to secure data in transit. This application is running locally, so this is not inherently an issue during this test.

Remediation

Prior to taking this application live, implement TLS or SSL encryption to secure communications between the web application and users. Obtain and install a valid TLS/SSL certificate from a reputable Certificate Authority (CA) to enable HTTPS protocol. Ensure that all sensitive data, including login credentials and personal information, is transmitted over secure channels to protect against eavesdropping and data interception.

Additional Scans and Reports

SixSecure provides clients with all report information gathered during testing. This includes an OWASP ZAP vulnerability scanning report in a detailed format. This report contains raw vulnerability scans and additional vulnerabilities not exploited by SixSecure.

This report identifies hygiene issues needing attention but are less likely to lead to a breach, however they provide opportunities to further enhance defense-in-depth strategies. For more information, please see the additional document sent with the inclusion of this report.