SECURITY ASSESSMENT

<< Juice Shop Vulnerabilities Report >>

Submitted to: << Omar Tarek Zayed>>
Security Analysts: << Habiba Mohamed >>
<< Ahmed Hassan >>
<< Rahma Ahmed >>
<< Ahmed Hatem >>
<< Salma Nasr >>

Date of Testing: <<12/9/2024>>
Date of Report Delivery: <<11/10/2024>>

Table of Contents

Security Engagement Summary

Engagement Overview

The vulnerability assessment was requested by the Development Team at **Dojuicer**, who are responsible for the organization's web applications. The assessment's goal was to evaluate the security risks posed by a legacy web application and to identify any significant vulnerabilities that could jeopardize the application's integrity and expose the organization to potential data breaches or attacks. The security assessment was performed by a team of experienced security analysts: **Habiba Mohamed, Ahmed Hassan, Rahma Ahmed, Ahmed Hatem, and Salma Nasr**, under the supervision of **Omar Tarek Zayed**.
This assessment was a part of the organization's recurring vulnerability and risk management strategy, scheduled regularly to maintain a robust security posture and mitigate any emerging threats. It aimed to identify critical vulnerabilities, assess their potential business impact, and propose remediation strategies to enhance the application's security.

Scope

The scope of the engagement focused on assessing the legacy web application for vulnerabilities across multiple areas, including authentication mechanisms, data handling, input validation, and access control. The decision to assess this particular application was based on its role in handling sensitive business and customer data, making it crucial to ensure its security integrity.

Executive Risk Analysis

Summarize the overall risk that the report indicates for the scope. (High | Medium | Low)

Explain why this risk level was reported. Include a summary of vulnerabilities in discussion (Executive Summary) form.

Executive Recommendation

The overall risk level for the application was determined to be **High**. This conclusion was based on the identification of multiple critical vulnerabilities, including Broken Authentication, Sensitive Data Exposure, and SQL Injection. These vulnerabilities could lead to unauthorized access, data leaks, or full system compromise if exploited. Given the ease of exploitation for some vulnerabilities, such as SQL injection and weak password management, the risk was elevated to a high level.

The vulnerability assessment revealed significant weaknesses in the application's authentication and data handling processes, leaving it vulnerable to brute-force attacks, data exposure, and injection attacks.

Significant Vulnerability Summary

❖ High Risk Vulnerabilities

- ❖ Password Strength
- ❖ Confidentiality Document
- ❖ Meta Geo Stalking
- ❖ Forgotten Sales Backup
- ❖ Login admin
- ❖ Product Search
- ❖ HTTP-Header XSS
- ❖ Cross-Site Request Forgery (CSRF)
- ❖ Admin Section
- ❖ Five-Star Feedback
- ❖ View Basket
- ❖ Manipulate Basket
- ❖ Forged Feedback
- ❖ Admin Registration
- ❖ Bully Chatbot
- ❖ Allowlist Bypass
- ❖ Forged Review

❖ Medium Risk Vulnerabilities

- ❖ Visual Geo Stalking
- ❖ Payback Time
- ❖ Empty User Registration
- ❖ Score Board
- ❖ CAPTCHA Bypass
- ❖ Login MC SafeSearch
- ❖ security policy

❖ Low Risk Vulnerabilities

- ❖ Zero Stars
- ❖ Missing Encoding

- Repetitive Registration
- Privacy Policy
- Email Leak

Significant Vulnerability Detail

<<Broken Authentication>>

- Password Strength

<<HIGH>>

Vulnerability Overview:

Broken authentication occurs when user credentials (such as passwords) are managed improperly, leading to unauthorized access or account compromise. A system that does not enforce strong password creation guidelines—such as minimum length, complexity, and the prevention of common passwords—is vulnerable to brute-force or credential-stuffing attacks.

Password Strength Issues:

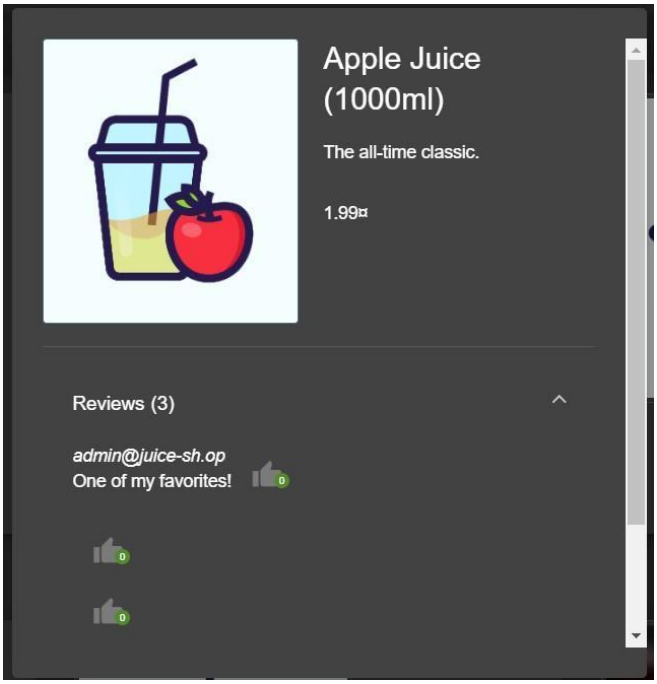
- **Lack of Complexity**: The system does not enforce strong password complexity, allowing users to create weak, easily guessable passwords (e.g., "password123").
- **Short Passwords**: The minimum required password length is insufficient, making it more vulnerable to brute-force attacks.
- **No Multi-Factor Authentication (MFA)**: Absence of an additional layer of authentication increases the risk, as password-only systems are more vulnerable.

Exploit Probability:

The likelihood of this vulnerability being exploited is **high**, given the numerous available tools and databases of commonly used passwords. Attackers can leverage these to launch automated attacks, targeting weak or reused passwords.

Proof of Validation

- Scanned the website for visible or hidden mentions of the administrator's email. Comments in the website's source code revealed the email address: admin@juice-sh.op

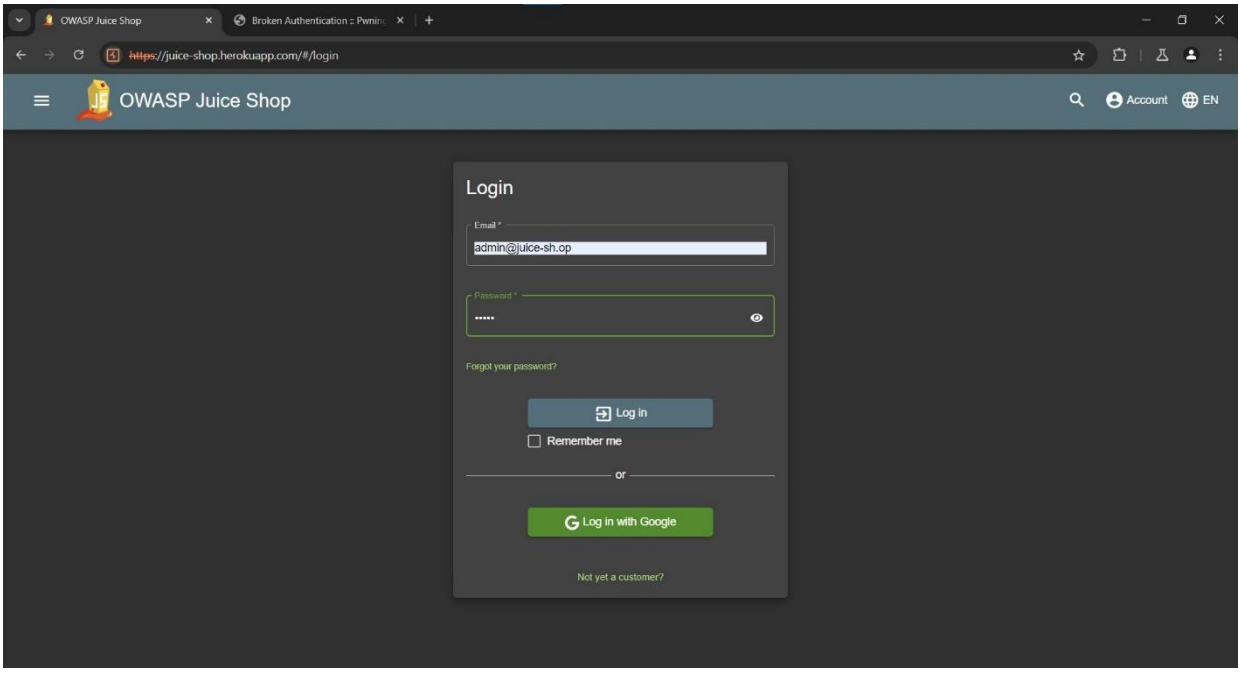


Brute Force Attack:

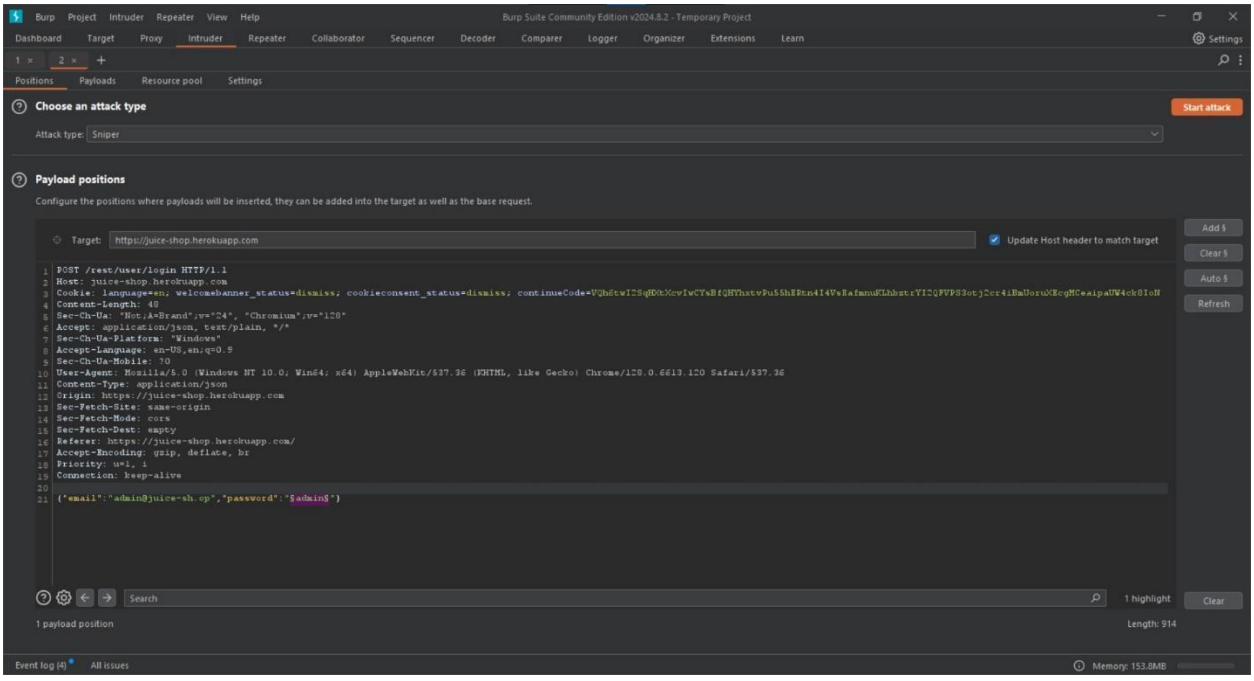
Intercept Login Request

- Attempted to log in with the administrator email and a random password.

- Intercepted the login request using Burp Suite to capture the request details necessary for the brute-force attack.



- Configured Burp Suite's Intruder module to use a list of common passwords, replacing the placeholder password with each entry from the list during the attack.



- Ran the brute force attack and monitored the responses for successful authentication indicators.
- Successfully authenticated using the password: admin123

Business Risk:

Broken authentication can result in significant business risks, including unauthorized access to sensitive user data, system-wide compromise, and financial loss due to fraudulent transactions or data breaches.

Remediation Recommendations:

1. **Enforce Strong Password Policies**: Require passwords to be at least 12 characters long and include a mix of letters, numbers, and special characters.
2. **Implement MFA**: Add a second factor (such as an SMS code or authenticator app) to strengthen the authentication process.
3. **Account Lockout**: Limit the number of failed login attempts to prevent brute-force attacks.
4. **Use Secure Password Hashing**: Ensure all passwords are hashed and salted before being stored in the database.

<< Sensitive Data Exposure >>

- Login MC SafeSearch

<<MEDIUM>>

Vulnerability Description:

The "Login MC SafeSearch" challenge exposes a **Sensitive Data Exposure** issue, where sensitive user credentials are retrievable through OSINT techniques and poor password hygiene. MC SafeSearch's account is protected by a weak password that, while somewhat obscured by character substitution, is ultimately guessable based on the information revealed in a video and prior challenges. The solution demonstrates how publicly available data can lead to account compromise without the need for advanced techniques like SQL Injection.

How the Vulnerability Was Identified:

The vulnerability was identified by leveraging a combination of OSINT (Open Source Intelligence) and prior knowledge obtained from the **Admin Section** and **Login Admin** challenges, where user data is stored in an accessible JSON document. The following steps were taken to solve this challenge:

1. **Gathering Information from the Video**:
    - Upon analyzing the video (which resembles a YouTube-like interface), at **timestamp 0:33**, MC SafeSearch reveals part of his password: “Mr. Noodles,” with a twist of replacing vowels with zeroes.
    - This leads to the assumption that the password is Mr.N00dles.
2. **Finding the Email Address**:
    - In the **Admin Section** challenge, a JSON file containing user data was found, and MC SafeSearch’s email was listed as MC.SafeSearch@juice-sh.op.
    - Alternatively, the email could be guessed based on prior conventions of email addresses in the application.
3. **Logging in with the Credentials**:
    - By combining the email MC.SafeSearch@juice-sh.op and the password Mr.N00dles, the login attempt succeeded, solving the challenge.

Risk Assessment

This vulnerability is assessed as Medium Risk. While it doesn't rely on technical exploits, it highlights the danger of sensitive data exposure and poor password practices. Publicly revealing even small clues about a password, combined with basic OSINT techniques, can lead to account compromise.

Affected Users and Business Impact

This issue directly affects MC SafeSearch, a user in the system, whose account was compromised due to poor password practices. The business impact can range from loss of customer trust to potential legal liabilities if sensitive data were accessed from this account.

Exploit Probability

The probability of exploiting this vulnerability is Medium. The attacker only needs to perform OSINT gathering by watching the video and have prior knowledge of the application's user data (via the JSON file). No advanced tools or techniques are required.

Proof of Validation

The email MC.SafeSearch@juice-sh.op and the password Mr.N00dles were used to log in to MC SafeSearch's account without applying any SQL Injection or bypass techniques. The login was successful, confirming the vulnerability.

- Confidentiality Document

<<HIGH>>

Vulnerability Description:

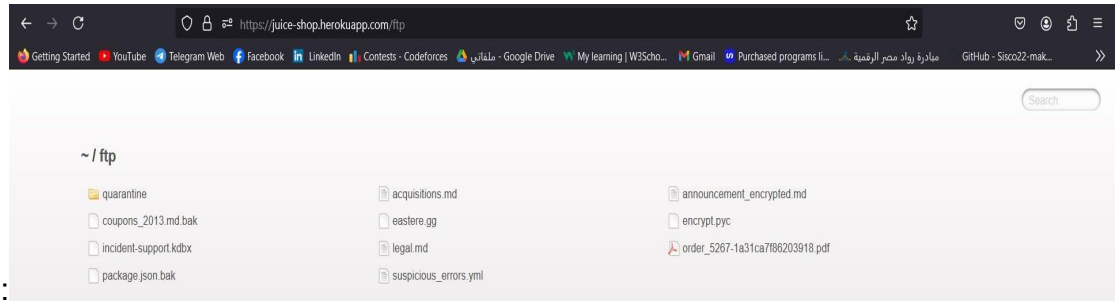
Sensitive Data Exposure occurs when confidential information, such as personal identification numbers, authentication credentials, or financial data, is accessible to unauthorized users. In this case, sensitive data such as passwords, credit card information, or customer records might be transmitted or stored insecurely.

How the Vulnerability Was Identified:

During a security assessment of the legacy web application, it was discovered that sensitive information was exposed via insecure communication channels. A lack of HTTPS implementation on pages handling sensitive data allowed for potential interception of traffic (Man-in-the-Middle attacks). Additionally, sensitive data was found stored in plaintext within the database and logs, increasing the risk of compromise.

Validation of the Vulnerability:

- as you can see, the application discloses the internal data:



- For example, the acquisitions.md file contains sensitive information about the company's acquisitions.

```
# Planned Acquisitions

> This document is confidential! Do not distribute!

Our company plans to acquire several competitors within the next year.
This will have a significant stock market impact as we will elaborate in
detail in the following paragraph:

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet.

Our shareholders will be excited. It's true. No fake news.
```

Business Impact:

The exposure of sensitive customer data puts the organization at significant risk of regulatory fines and reputational damage. The leak of credentials could result in account takeover, while financial data exposure could lead to fraud.

Remediation / Mitigation:

To mitigate this vulnerability, it is recommended to:

1. Implement HTTPS with a valid SSL certificate for all pages, especially those handling sensitive information.
2. Encrypt sensitive data both in transit and at rest using strong encryption standards (AES-256 for data at rest, TLS 1.2+ for data in transit).
3. Regularly audit application logs and databases to ensure sensitive information is not stored in plaintext.
4. Implement a secure logging mechanism that excludes sensitive information.

- Meta Geo Stalking

<<HIGH>>

Vulnerability Description:

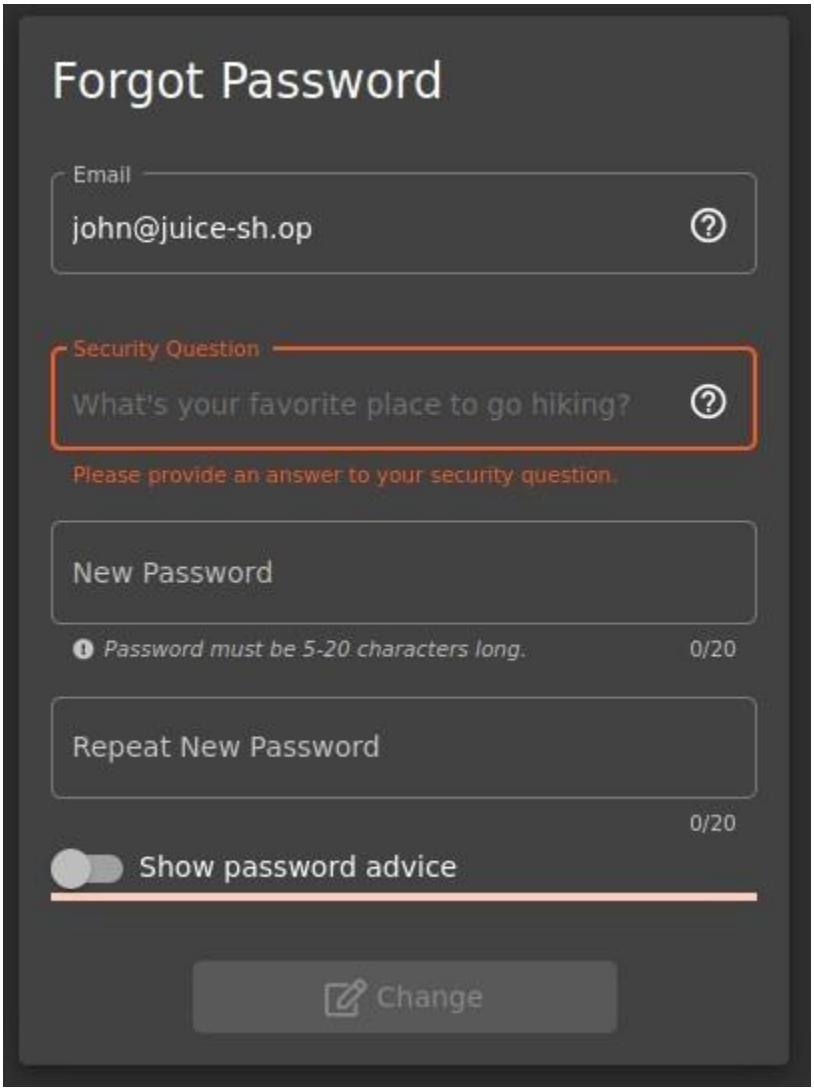
Meta Geo Stalking refers to the exposure of sensitive geographic metadata within images or files uploaded by users. In the Juice Shop web application, images uploaded by users contain metadata that includes location information (latitude and longitude). Attackers can exploit this data to track users' physical locations, leading to privacy violations or even physical stalking.

How the Vulnerability Was Identified:

This vulnerability was discovered by uploading an image containing EXIF (Exchangeable Image File Format) data. The application does not strip this metadata upon upload. After downloading the image, EXIF data was extracted using tools such as exiftool or a web-based EXIF viewer, revealing sensitive location information embedded in the image.

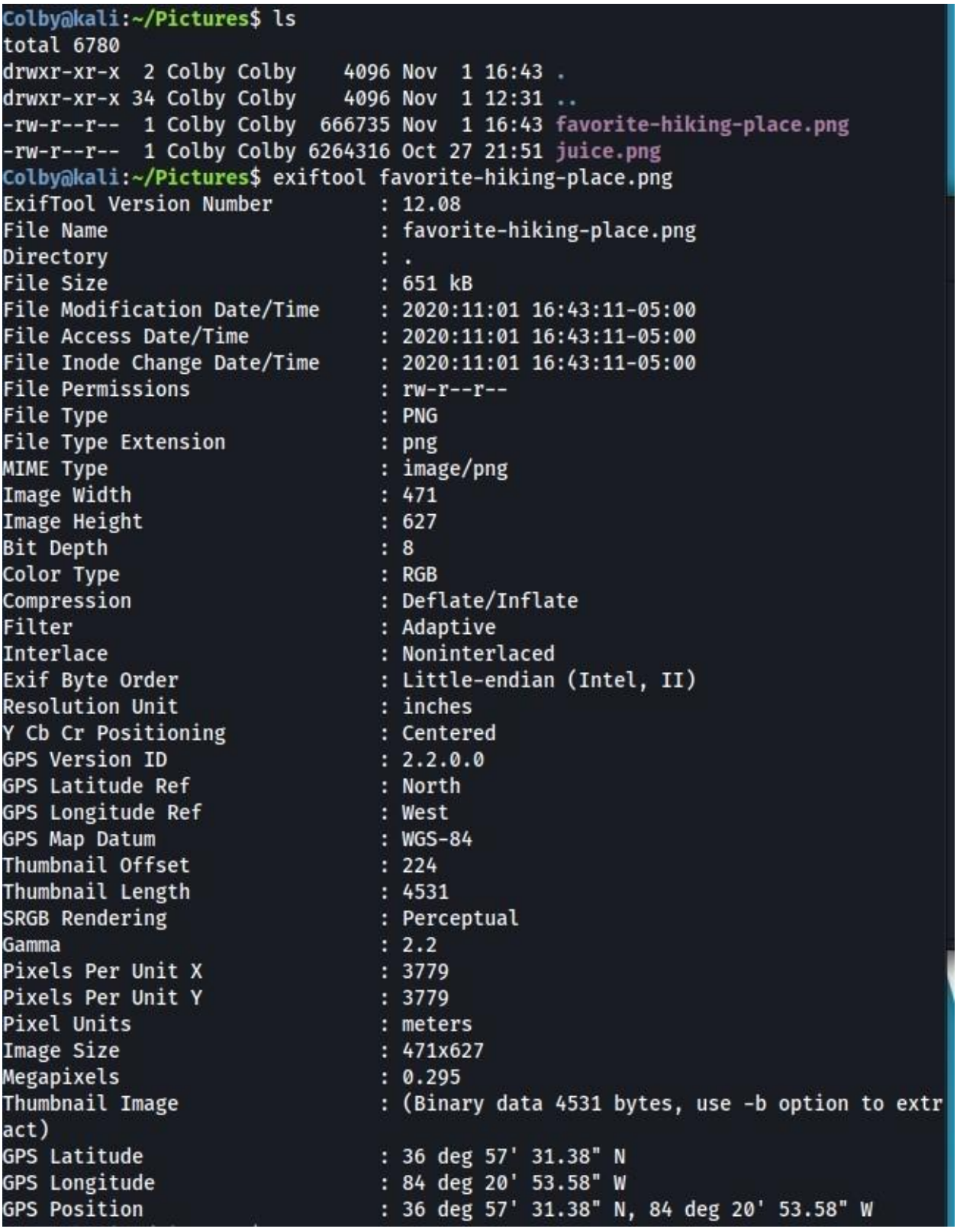
Proof of Validation:

Before going to the photo wall, it's important to know what we're looking for, so open the "Forgot Password" link and enter John's email address (which we collected in the "Admin Section" challenge. Alternatively, guess his email address).



Hiking, eh? OK. Let's check the Photo Wall. There's only one photo of a trail, but to be safe it's a good idea to check the caption: "I love going hiking here… (© j0hNny)". This certainly appears to be the right photo, so save it to your system and let's check out the metadata. ExifTool is a fantastic tool for this type of thing.

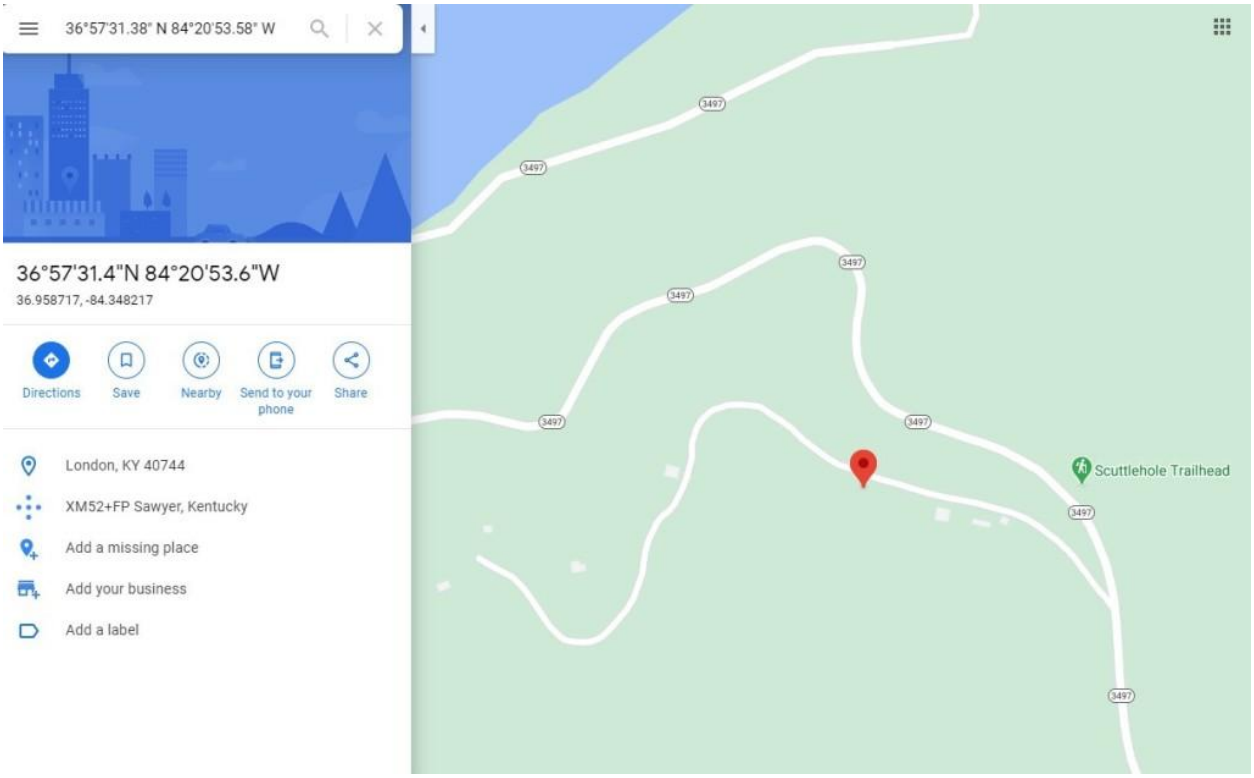
I love going hiking here… (© j0hNny)



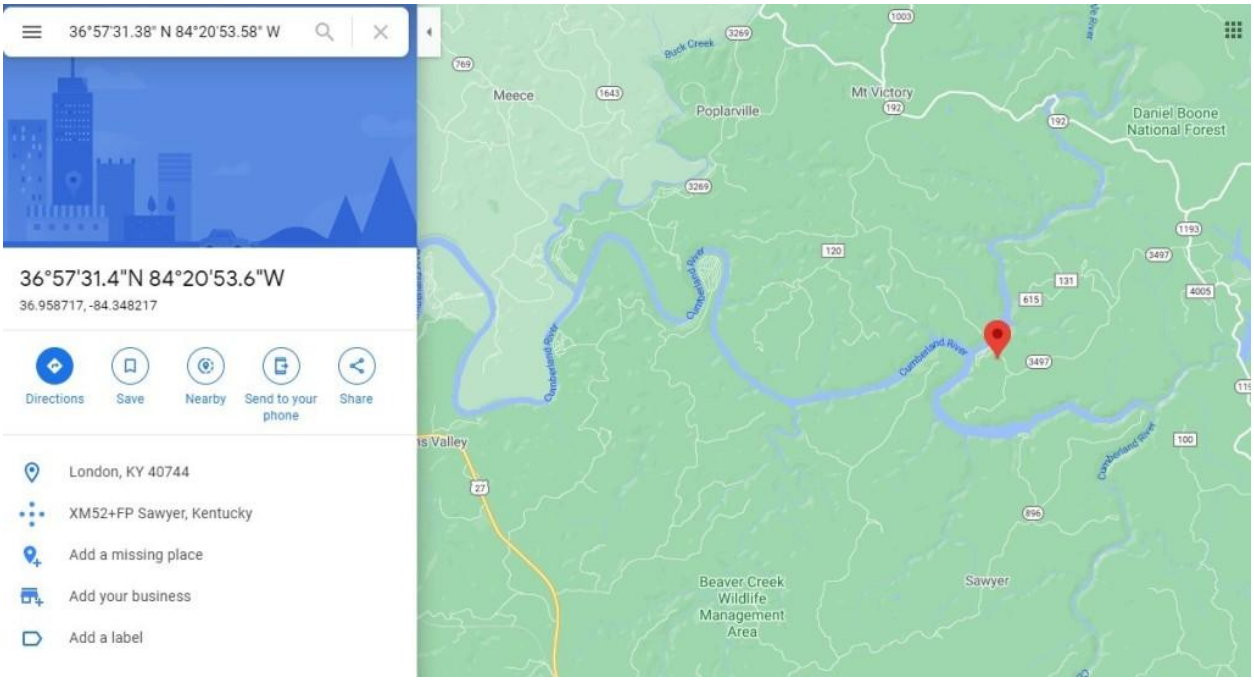
```
Colby@kali:~/Pictures$ ls
total 6780
drwxr-xr-x  2 Colby Colby    4096 Nov  1 16:43 .
drwxr-xr-x 34 Colby Colby    4096 Nov  1 12:31 ..
-rw-r--r--  1 Colby Colby  666735 Nov  1 16:43 favorite-hiking-place.png
-rw-r--r--  1 Colby Colby 6264316 Oct 27 21:51 juice.png
Colby@kali:~/Pictures$ exiftool favorite-hiking-place.png
ExifTool Version Number         : 12.08
File Name                       : favorite-hiking-place.png
Directory                       : .
File Size                       : 651 kB
File Modification Date/Time     : 2020:11:01 16:43:11-05:00
File Access Date/Time           : 2020:11:01 16:43:11-05:00
File Inode Change Date/Time     : 2020:11:01 16:43:11-05:00
File Permissions                : rw-r--r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 471
Image Height                    : 627
Bit Depth                       : 8
Color Type                      : RGB
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Exif Byte Order                 : Little-endian (Intel, II)
Resolution Unit                 : inches
Y Cb Cr Positioning             : Centered
GPS Version ID                  : 2.2.0.0
GPS Latitude Ref                : North
GPS Longitude Ref               : West
GPS Map Datum                   : WGS-84
Thumbnail Offset                : 224
Thumbnail Length                : 4531
SRGB Rendering                  : Perceptual
Gamma                           : 2.2
Pixels Per Unit X               : 3779
Pixels Per Unit Y               : 3779
Pixel Units                     : meters
Image Size                      : 471x627
Megapixels                      : 0.295
Thumbnail Image                 : (Binary data 4531 bytes, use -b option to extr
act)
GPS Latitude                    : 36 deg 57' 31.38" N
GPS Longitude                   : 84 deg 20' 53.58" W
GPS Position                    : 36 deg 57' 31.38" N, 84 deg 20' 53.58" W
```

Fortunately, Johnny isn't savvy enough to strip exif data from his photos, so after looking up how Google Maps wants coordinates to be formatted, format the string and search.

**For a complicated set of reasons, other recommended formats may place the location in an entirely different area, so use "36°57'31.38" N 84°20'53.58" W"*



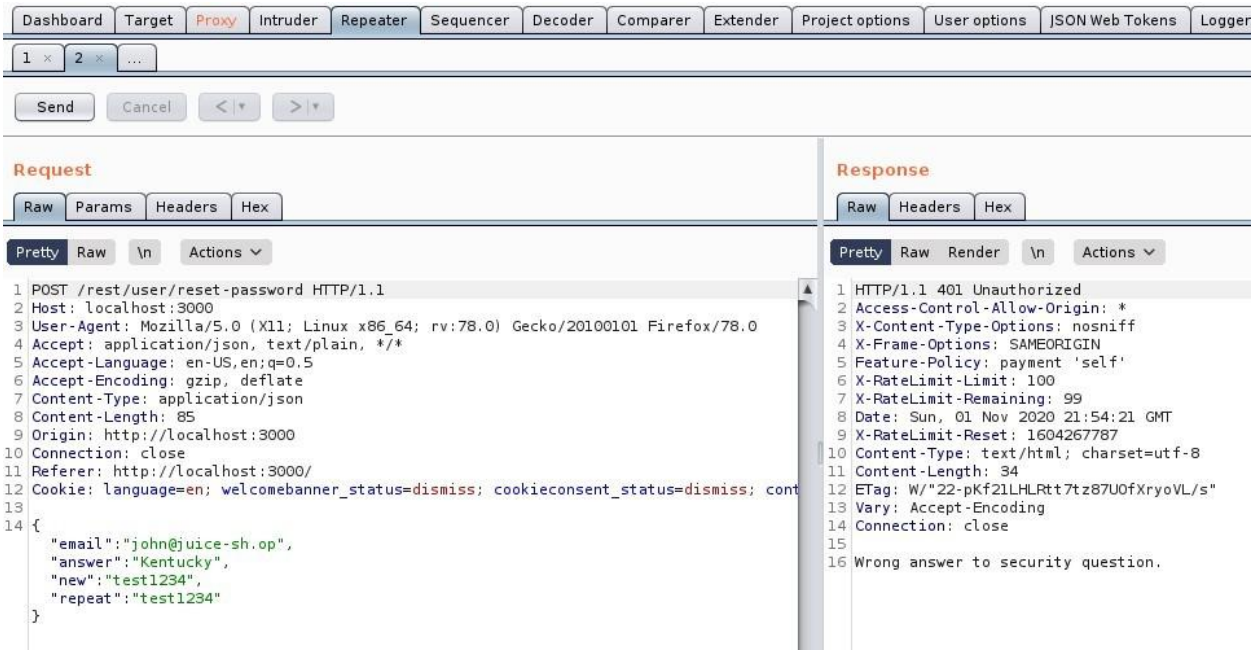
Be aware that you're not just looking for candidate locations simply in the immediate area, so be sure to zoom out and gather more information about the general area.



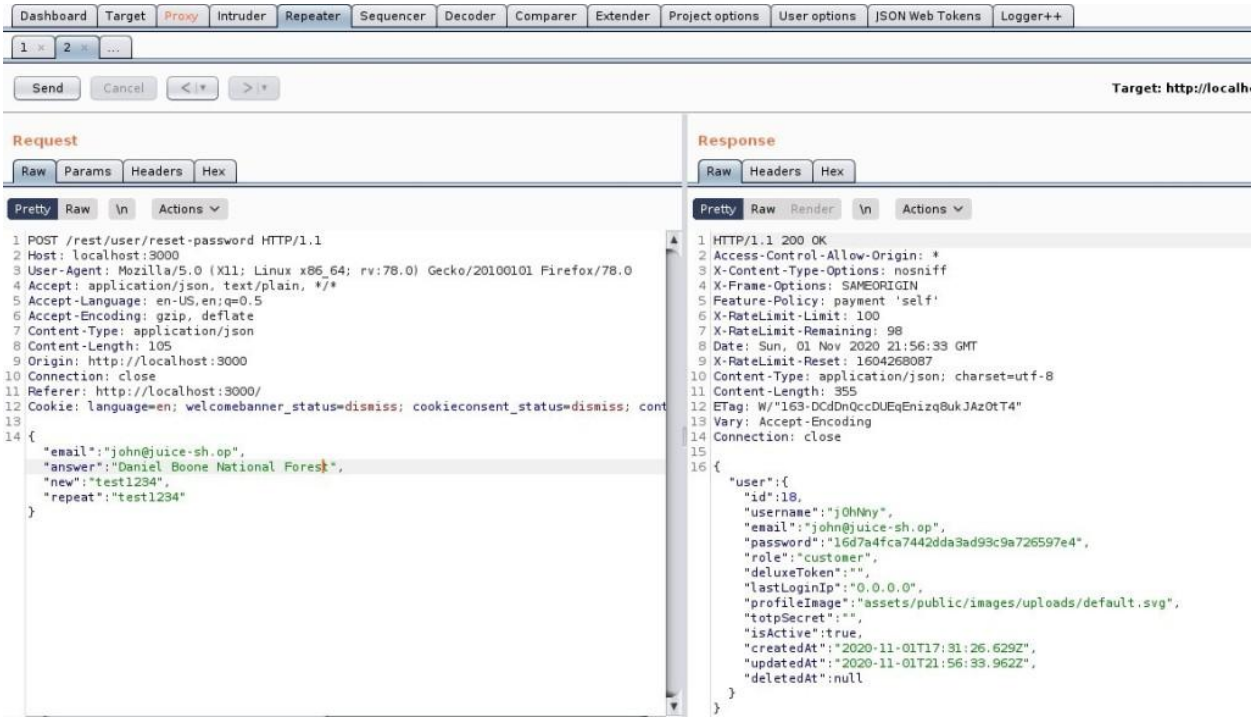
Now that we have enough information to build a list of potential locations, go ahead and do that.

```
Kentucky
Sawyer
Daniel Boone
Daniel Boone National Forest
Scuttlebutt
Scuttlebutt Trail
```

At this point, the obvious tool to use would be Burp's Intruder, but for the sake of variety I'm going to use Repeater.



After only a few failed attempts, the correct answer returns an abundance of user information.



Business Risk:

The exposure of users' geographic data can have significant privacy and security implications. Malicious actors could track users' locations, leading to privacy breaches, stalking, or physical harm. This vulnerability could also result in legal liabilities under privacy regulations such as GDPR, which mandates strict handling of personal data.

Exploit Probability:

The probability of this vulnerability being exploited is **high**, as attackers with basic knowledge of metadata extraction tools can easily retrieve the sensitive geographic information. Additionally, users often remain unaware of the risks associated with metadata in their uploaded files.

Remediation Recommendations:

To mitigate risks associated with metadata in shared images and sensitive data exposure:

- Metadata Scrubbing: Encourage users to remove metadata from photos before uploading them to public platforms or implement server-side processes to automatically strip this data.
- Use Non-Descriptive Security Questions: Employ security questions that do not directly relate to information that could be publicly deduced or found through OSINT techniques. A better option is to not use security questions and rather send an email for resetting password.

❖ Forgotten Sales Backup

<<HIGH>>

Vulnerability Description:

This vulnerability arises due to the absence of access control measures and lack of encryption for sensitive files. The backup contains detailed sales records, which include customer names, addresses, and other confidential information. The absence of file-level security means that anyone who can access the URL can download the file, resulting in a significant breach of privacy and potential legal implications under data protection regulations like GDPR.

Vulnerability Impact:

Sensitive data exposure such as this can lead to a range of issues, including:

- **Identity theft**: Attackers can use customer data (such as names, email addresses, and transaction histories) for phishing or fraud.
- **Regulatory fines**: Exposure of personal data violates privacy regulations (e.g., GDPR, CCPA), which could result in heavy fines.
- **Reputation damage**: A breach of sensitive data could severely damage the organization's reputation, leading to a loss of customer trust.

Exploitability:

This vulnerability is **easy to exploit**. It only requires knowledge of the file's location or an automated scan that can reveal exposed directories and files. No special privileges or advanced skills are needed to exploit this issue.

Proof of Concept:

1. **Access the FTP Server**: Navigated to the FTP server directory listing at juice-shop.herokuapp.com/#/ftp, which was discovered in a previous challenge (Privacy Policy). This server hosted various files, including backups and configuration files.
2. **Identify the Target File**: Scanned through the list of files and identified coupons_2013.md.bak as a likely candidate for containing sensitive sales data due to its naming convention indicating it's a backup file.

3. **Exploit NULL Byte Injection**: Utilized a NULL byte injection technique to access the backup file. Many web applications do not properly sanitize input, allowing for directory traversal or file inclusion attacks if the application is improperly handling file names.
    - **Construct the URL**: Appended a NULL byte character, represented in URL encoding as %00, to the file request to potentially bypass any restrictions or parsing issues by the server. This was tried because older or misconfigured servers might treat the NULL byte as a string terminator, thus ignoring any file extension checks or other controls.
    - **Request the File**: Accessed the URL http:// juice-shop.herokuapp.com/#//ftp/coupons_2013.md.bak%00.md in a browser.
4. **Review the Data**: Upon successful retrieval, reviewed the content of coupons_2013.md.bak for any sensitive information, confirming the exposure of sales data or discount coupons that should not have been publicly accessible.

Remediation:

- **Proper Access Controls**: Ensure that backup files are not stored in publicly accessible directories. Use proper access control mechanisms to restrict access.
- **Regular Audits and Cleanups**: Perform regular audits of storage locations and cleanup legacy or unused files to prevent accidental exposure.
- **Input Sanitization**: Enhance security measures to sanitize user inputs, especially in file retrieval functionalities, to prevent directory traversal or file inclusion attacks.

- Visual Geo Stalking

<<Medium>>

Vulnerability Description:

Visual Geo Stalking is a vulnerability found in web applications that involve the misuse of location-based metadata in user-uploaded images. In OWASP Juice Shop, users can upload pictures, often unaware that these images contain hidden metadata, specifically geolocation information embedded within EXIF data. If this metadata is not removed or sanitized before storing or displaying the image, it can lead to a serious privacy breach. Attackers can exploit this vulnerability by extracting the geolocation information from the images, which might reveal sensitive details about a user's real-world location.

Probability of Exploit:

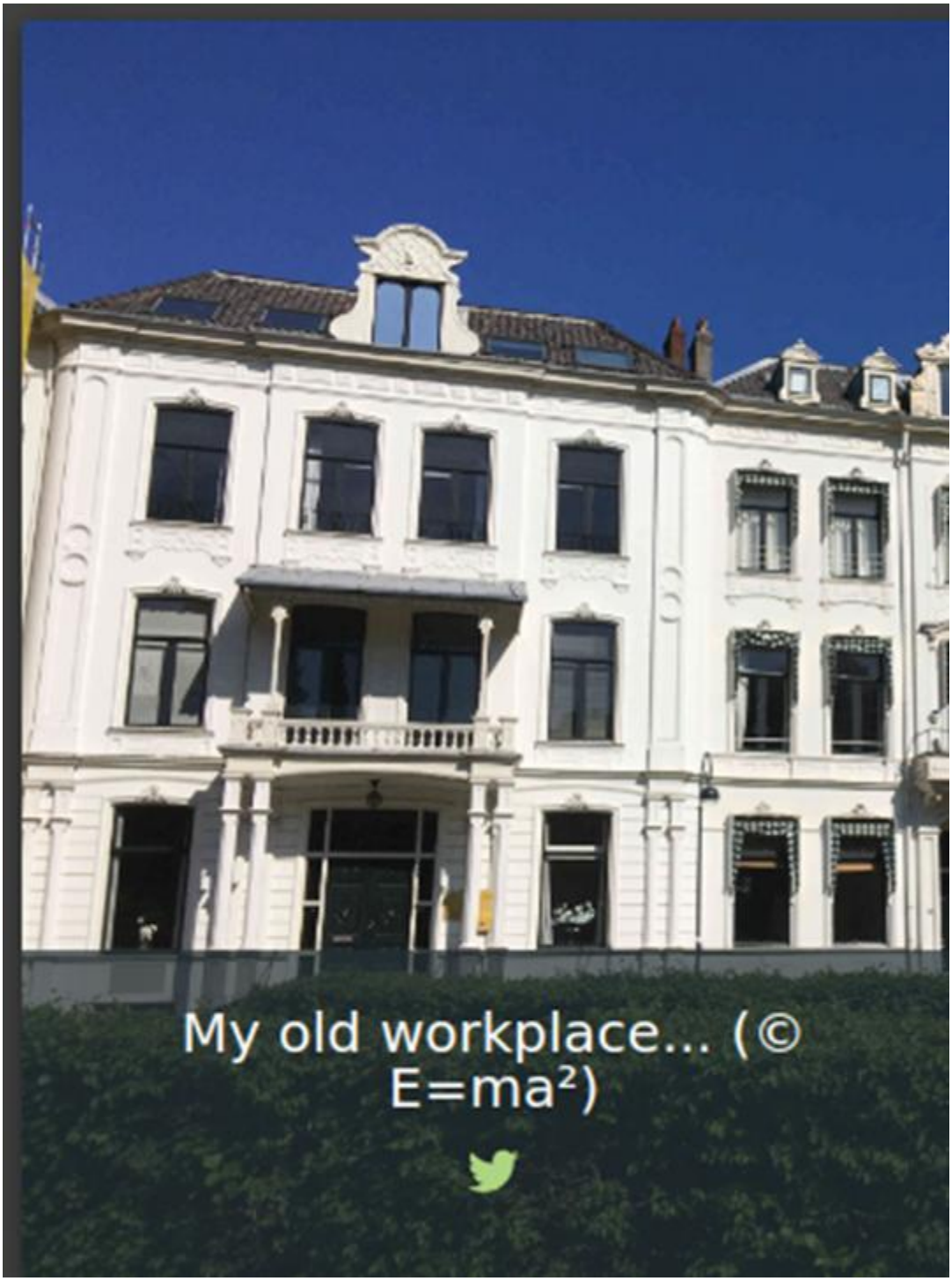
Given the wide availability of EXIF extraction tools, the likelihood of exploiting this vulnerability is high if images with sensitive data are uploaded and accessible. Attackers with minimal technical skills can extract and interpret EXIF data. However, the actual impact depends on whether users are aware of the data embedded in their images and how accessible these images are within the Juice Shop environment.

Impact:

This vulnerability poses a high risk to user privacy and security. Depending on the level of exposure, attackers could gain precise details about a user's physical location, personal habits, and routine, which can lead to more severe consequences, including stalking, theft, and physical harm.

Proof-of-Concept:

1. Photo Examination:
    - Emma's post on a social media-like platform includes a photo of a building with a caption referencing a past workplace.
    - The challenge involves identifying Emma's former workplace through the image provided.



Detailed Image Analysis

2. Utilizing Google Lens:
    - The image was analyzed using Google Lens, which identified the location as the "Kenau Park area in Haarlem."

Stock Photo ID: 1627793506

HAARLEM-JANUARY 19, 2020. The Kenau park is a small park near the central station of Haarlem. It's surrounded by stately buildings from the 18th century. The name comes from a Dutch female hero.

Show more

Important information

Editorial Use Only.

Interested in using Editorial content for commer
commercial license with Asset Assurance™ ma
the legal coverage and peace of mind you need.

3. Google Maps Exploration:
    - Using Google Maps to virtually visit the location, various business signs and plaques were examined to match potential answers to security questions.

4. Identifying Key Details:
    - A close inspection of another part of the image revealed a sign for "ITsec" on a window. This detail seemed crucial but was initially overlooked.

Answering the Security Question and Resetting the Password

5. Security Question Answering:
    - The term "ITsec" was attempted as an answer to a security question regarding Emma's former workplace, based on the context provided by the visual clues in the photo and the information deduced from online tools.

6. Password Reset:
    - Successfully using "ITsec" as the security question answer allowed the password for Emma's account to be reset.

Remediation:

To prevent sensitive data exposure through visual or other publicly accessible information:

- **Educate Users**: Inform users about the risks associated with sharing detailed photos or information that could inadvertently reveal answers to security questions or other sensitive data.
- **Privacy Settings**: Encourage the use of strict privacy settings on social media and other platforms to control who can view personal information and posts.
- **Security Question Choices**: Advise against using easily discoverable information as answers to security questions. Encourage more personal and less predictable answers.

- Zero Stars

<<LOW>>

Description:

Give devastating zero-star feedback to the store

Methodology and Solution:

If we send normal feedback (with any star value), copy the request, change the rating to 0 and resend it, the server accepts it.

1. **Setup Interception**: Configured Burp Suite to intercept the HTTP requests sent from the browser using Proxy functionality.
2. **Submit Review**: Attempted to submit a review normally through the website's interface, capturing the outgoing request in Burp Suite.
3. **Modify the Request**: Altered the ratings parameter in the intercepted request to 0, despite the website originally not allowing a zero-star rating to be submitted through its user interface.

4. **Forward the Request**: Sent the modified request to the server, successfully submitting a zero-star review.

Remediation

To prevent such issues and secure the application against improper input validation:

- **Implement Server-Side Validation**: Ensure that all inputs, including ratings, are validated on the server side to prevent unauthorized values from being processed.

- **Secure Client-Side Logic**: Enhance client-side scripts to prevent easy tampering from the developer console. However, always assume that client-side controls can be bypassed.
- **Use Proper Data Types**: Ensure that the data type for ratings strictly accepts only valid ratings and does not default to accepting null or other unintended values as valid input.

<< SQL Injection >>

- Login admin

<<HIGH>>

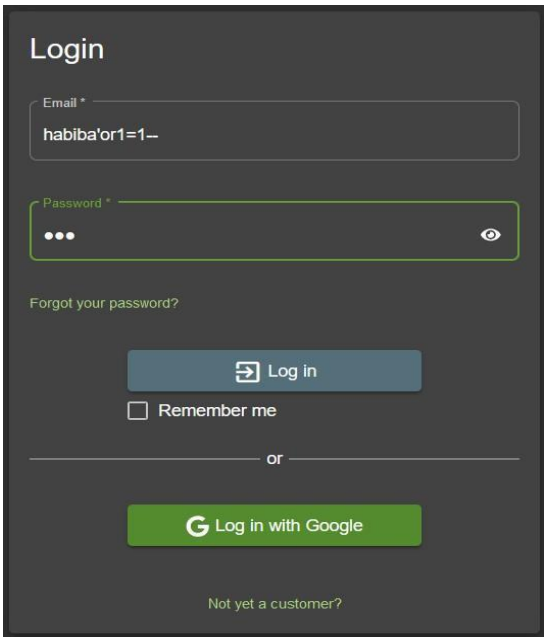
Vulnerability Summary
SQL Injection is a critical vulnerability that allows an attacker to interfere with the queries an application makes to its database. In this case, the application's admin login page is vulnerable to SQL injection, enabling an attacker to bypass authentication and gain administrative access to the system.

Risk Level: High
SQL Injection on the admin login page poses a severe risk as it can allow unauthorized users to bypass authentication and gain full control over the web application. Attackers can steal sensitive data, modify application behavior, or launch further attacks.

Probability of Exploit: High
SQL Injection is a well-known and highly exploited vulnerability. Given the simplicity of exploiting it with basic SQL knowledge, this vulnerability poses a high probability of being exploited in the wild.

Details and Proof-of-Validation
The vulnerability was identified during the manual testing phase of the web application's admin login form. By entering specially crafted SQL code into the username or password fields, an attacker can manipulate the database query logic. For example, entering the following payload:

habiba' OR '1'='1'; --



This input exploits a SQL injection vulnerability in the authentication mechanism. It works by modifying the SQL query to always return true, effectively bypassing the need for correct login credentials.

- **Original Query** (pseudo-code):

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password';
```

- **Injected Query**:

```
SELECT * FROM users WHERE username = 'admin' OR '1'='1'; -- ' AND password = 'password';
```

Here, the condition OR '1'='1' always evaluates to true, allowing the attacker to bypass authentication and gain admin access.

Remediation Recommendations

1. **Use Prepared Statements**: Convert dynamic queries into prepared statements or parameterized queries, which separate SQL logic from data inputs.
2. **Input Validation**: Implement strong input validation mechanisms, ensuring that only valid and expected data types are allowed.
3. **Error Handling**: Disable detailed database error messages on production sites to avoid leaking information about the database structure.
4. **Database Access Controls**: Ensure the least privilege principle is applied to the database account used by the application, reducing the impact of potential exploits.

<< XSS >>

- ❖ Product Search

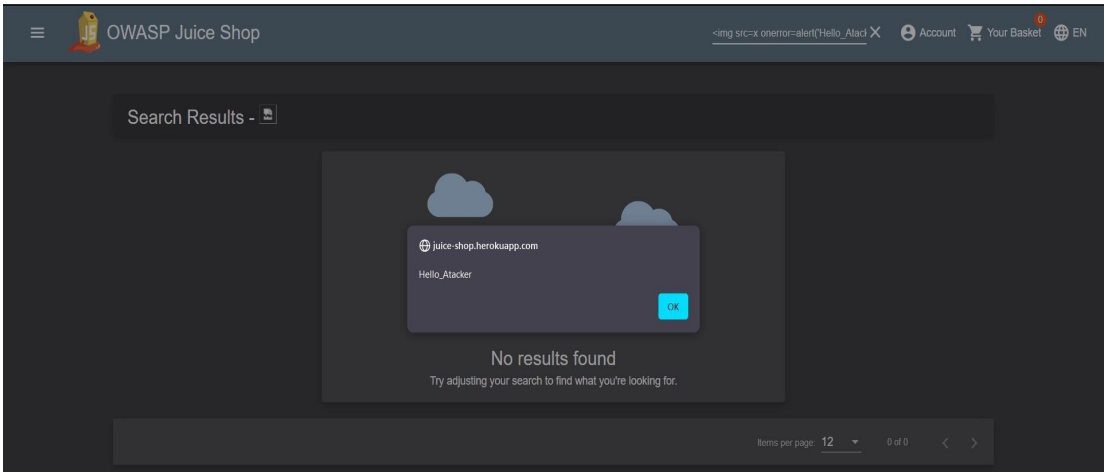
<<HIGH>>

Vulnerability Detail

- **How the Vulnerability was Identified:** During the vulnerability assessment, it was discovered that the product search feature of the Dojuicer web application is vulnerable to a **DOM-based Cross-Site Scripting (XSS)** attack. This type of XSS occurs when the application handles untrusted input directly in the browser via JavaScript. Specifically, when search input parameters are not properly sanitized and are passed to client-side code that manipulates the DOM.
- **How It Was Validated:** By entering the following payload in the search bar:

  img src=x onerror=alert (“Hello_Attacker”)

- JavaScript was executed, resulting in a pop-up. This indicates that user input is being inserted into the DOM without proper escaping or sanitization. Screenshots and browser logs confirm the execution of the malicious script.



- **Proof of Exploit:** A screenshot of the alert box pop-up triggered by the payload confirms the successful exploitation of DOM XSS in the product search functionality.
- **Probability of Exploit:** The probability of this vulnerability being exploited is **High**. DOM-based XSS vulnerabilities are often targeted because they do not require complex setups or the exploitation of a server-side flaw. Any attacker can inject malicious JavaScript into the web application, leading to session hijacking, redirection to malicious sites, or data theft.

Business Risk Impact

This vulnerability poses a **High** risk to the organization due to the potential for:

- **User Data Compromise:** An attacker could exploit this flaw to steal user session cookies, which could allow unauthorized access to user accounts.
- **Reputation Damage:** Should a successful exploit become public, the brand's reputation could be severely damaged, leading to loss of customer trust.
- **Regulatory Impact:** If the attack compromises sensitive user data (e.g., personal identifiable information), this could result in regulatory penalties for non-compliance with data protection laws such as GDPR.

Recommended Remediation

To mitigate this vulnerability, the following steps should be taken:

1. **Input Sanitization and Encoding:** All input passed to the client-side scripts should be properly sanitized and encoded. Use libraries or frameworks that automatically escape user input when inserting into the DOM.
2. **Content Security Policy (CSP):** Implement a strict CSP to prevent the execution of untrusted scripts.
3. **Security Testing:** Regularly perform static and dynamic code analysis to detect and remediate XSS vulnerabilities before deployment.

Security Analysis Methodology

1. **Tools Used:** Burp Suite, OWASP ZAP, and manual testing in developer tools to inspect and validate the DOM interactions.
2. **Actions Taken:**
    - Accessed the product search functionality of the web application.
    - Used a variety of input payloads to determine how the application handles user input in the DOM.
    - Identified the lack of proper input validation, leading to the DOM XSS.

- ❖ HTTP-Header XSS

<<HIGH>>

Description:

This vulnerability occurs due to improper validation of user-controlled HTTP headers, specifically the True-Client-IP header, which allows an attacker to inject a malicious JavaScript payload. When the user's IP address is displayed on the "Last Login IP" page, the application directly reflects the content of this header without proper sanitization. This leads to the execution of the injected payload, causing a Cross-Site Scripting (XSS) attack.

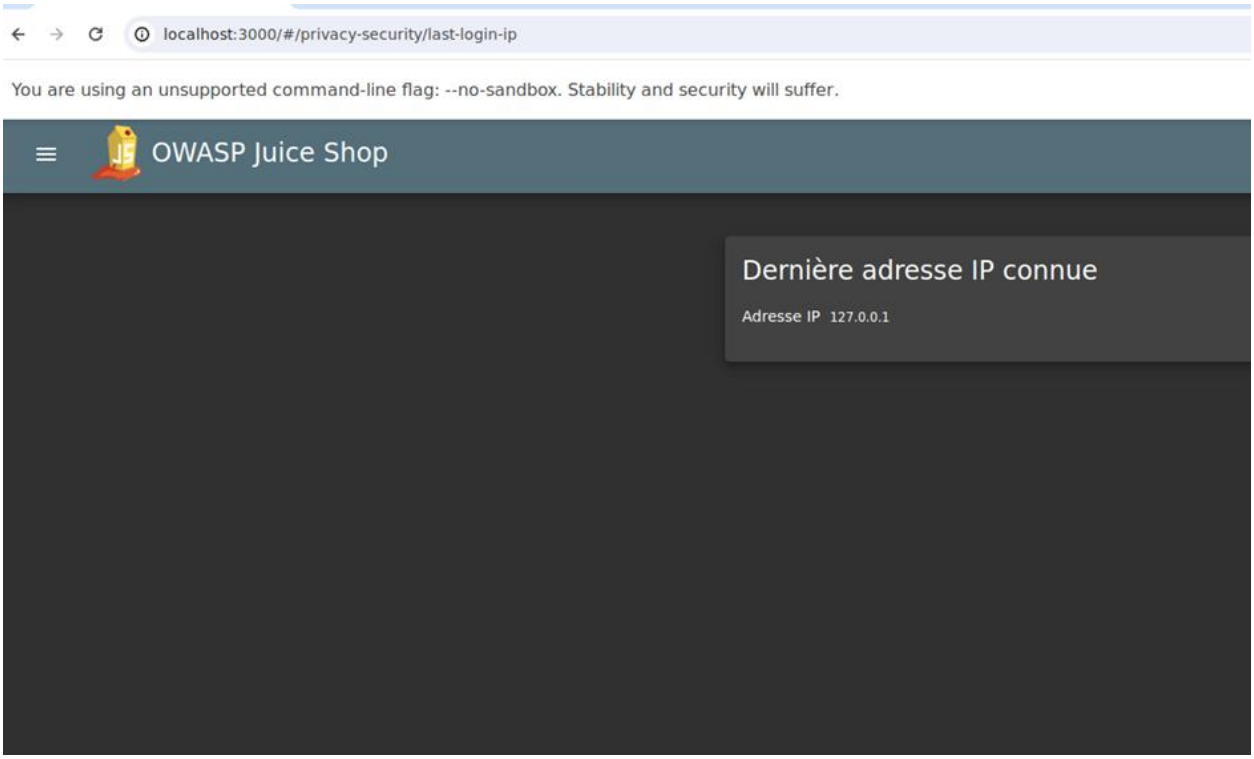
Risk Impact:

- **Business Impact:** High
    - Successful exploitation of this vulnerability can lead to session hijacking, the application's integrity, potentially damaging the organization's reputation.

Proof of Vulnerability:

Affected Page:

- Page: Last Login IP Page
- Vulnerable Parameter: True-Client-IP (HTTP header)

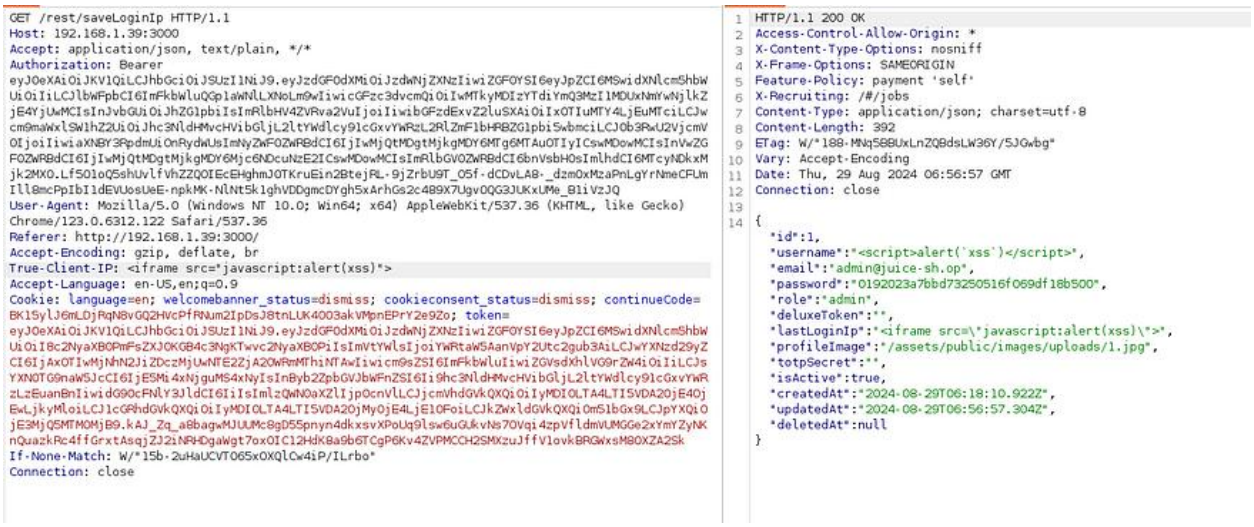


Attack Scenario:

In this scenario, the attacker manipulates the HTTP True-Client-IP header by injecting malicious JavaScript code. When the server reflects this value onto the "Last Login IP" page, the script is executed in the victim's browser. This can lead to unauthorized actions, such as stealing session tokens, cookies, or other sensitive information from users who visit the affected page.

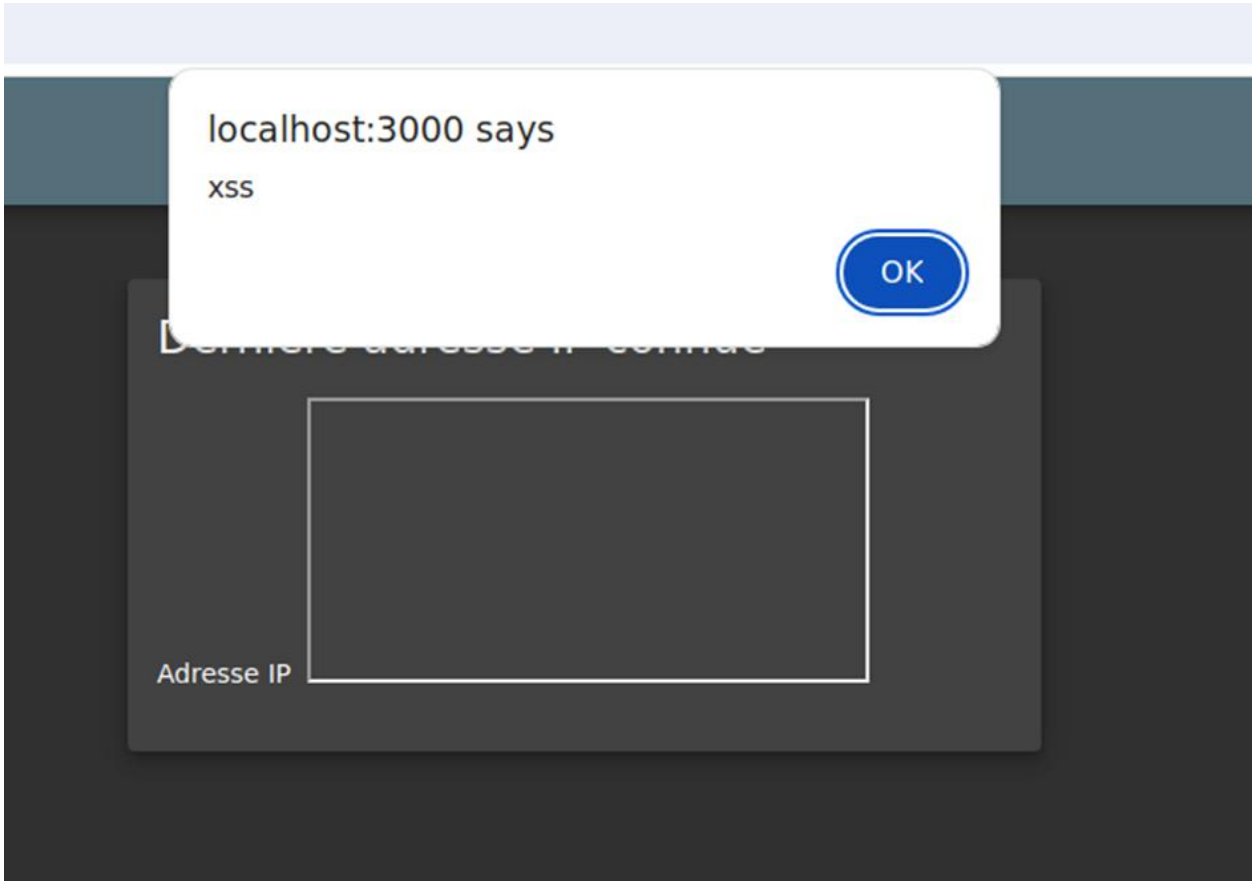
Exploit Process:

1. Intercept the Request:
    - Use a proxy tool (such as Burp Suite) to intercept a request to the "Last Login IP" page.
2. Modify the HTTP Header:
    - Modify the True-Client-IP header to include a malicious JavaScript payload. For example: True-Client-IP: <iframe src="javascript:alert(`xss`)">



Execute the Payload:

- When the page is rendered, the injected payload <iframe src="javascript:alert(`xss`)"> is executed in the browser, demonstrating the XSS vulnerability.



Remediation

- **Input Validation and Sanitization:** Ensure all user inputs, including HTTP headers, are properly validated and sanitized to prevent injection attacks.
- **Escape Output:** Always escape user inputs when displaying them in the user interface to prevent XSS.

<< Broken Access Control >>

- ❖ Cross-Site Request Forgery (CSRF)

<<HIGH>>

Vulnerability Summary

Cross-Site Request Forgery (CSRF) is a critical security flaw that allows an attacker to trick a user into performing unintended actions on a web application where they are authenticated. The attacker essentially forces the user's browser to send an unauthorized request to the web server on their behalf, typically exploiting the user's session. If the user is logged in, these actions can be executed with their privileges, leading to potential compromises like changing user data, performing transactions, or escalating privileges.

Risk Level: HIGH

This vulnerability has been classified as high risk due to the potential for exploiting privileged actions without user consent, causing serious damage to both user trust and system integrity. The probability of exploitation is significant if CSRF protection mechanisms (such as tokens) are not adequately implemented.

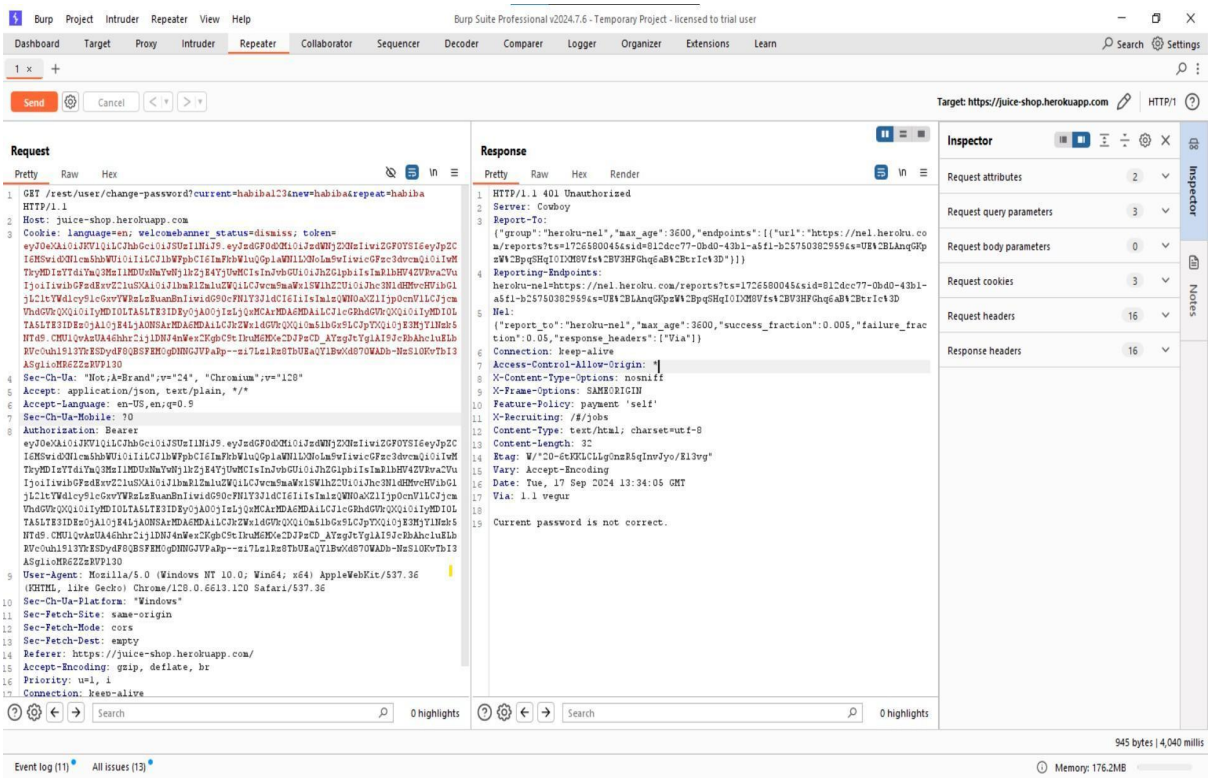
Impact

If exploited, an attacker can:

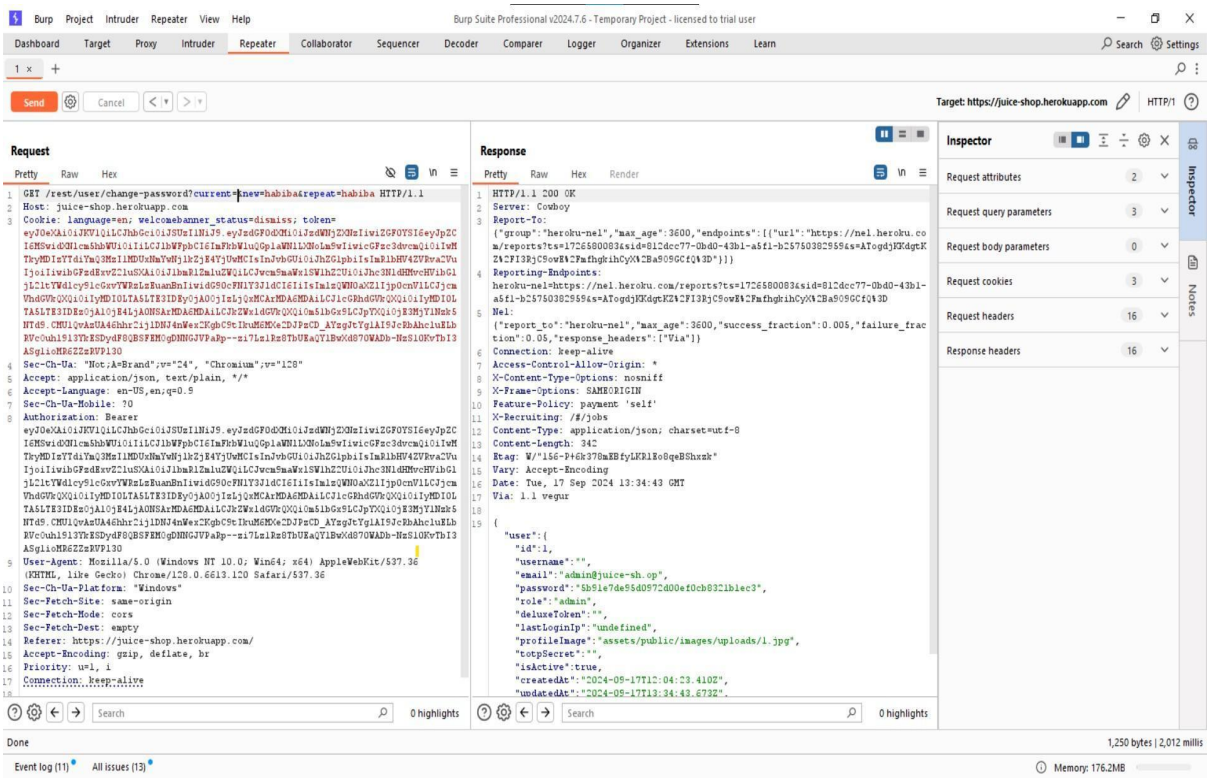
- Alter critical user data
- Execute unintended user actions, such as changing passwords
- Gain unauthorized access to sensitive parts of the application
- Impact business operations by hijacking administrative accounts

Proof of Vulnerability

- With old password:



- Without old password:



Mitigation Recommendations

To mitigate CSRF vulnerabilities:

- **Implement Anti-CSRF Tokens**: Include unique tokens in forms and validate them on the server.
- **Check the Origin and Referrer**: Validate headers to ensure that requests originate from the trusted domain.
- **Use SameSite Cookies**: Mark session cookies with SameSite=strict to prevent them from being sent on cross-site requests.

- **Require User Re-Authentication**: For sensitive actions, such as changing passwords or transferring funds, require the user to re-enter their credentials.

- ❖ Admin Section

<<HIGH>>

Vulnerability Overview

Broken access control is a significant security vulnerability that occurs when an application does not properly restrict user access to sensitive resources. In the case of the admin section of the Juice Shop application, inadequate access control measures can allow unauthorized users to access admin functionalities, leading to potential data leaks, unauthorized modifications, and administrative privilege escalation.

Identification and Validation

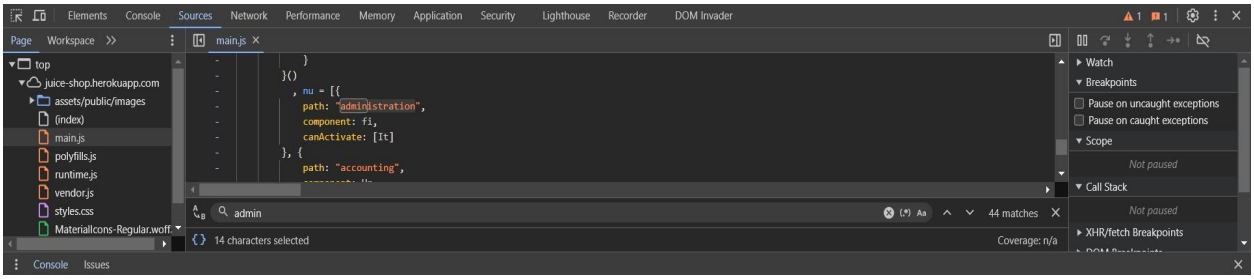
During the vulnerability assessment, it was discovered that the admin section could be accessed without proper authentication checks. This was validated by attempting to navigate directly to the admin URL (/admin) without being logged in. The application responded by providing access to administrative functions, indicating that access controls were either misconfigured or absent.

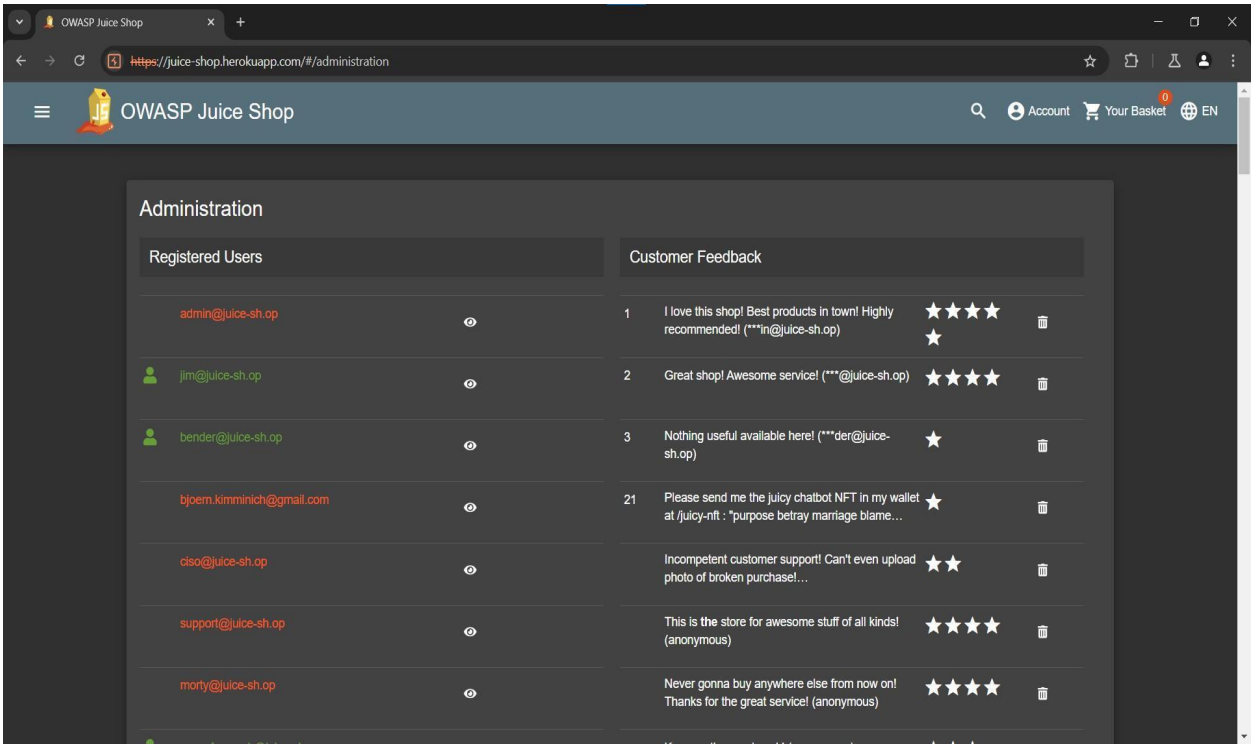
Probability of Exploit

The probability of exploitation for this vulnerability is **High**. Given the nature of the flaw, an attacker could easily exploit it by simply entering the admin URL into their browser. If the application does not implement any form of session or authentication checks, the attacker gains access to sensitive functions and data.

Proof of Validation

This led me to search through the code once more using the search parameter “admin” and I saw administration which I input into the URL, and I was able to access the admin page with admin privilege:





Risk Level

High

- **Explanation**: The potential for unauthorized access to sensitive administrative functions poses a critical risk to the integrity and confidentiality of the application. If exploited, attackers could manipulate data, alter application behavior, or gain additional unauthorized access to user accounts.

Recommendations for Remediation

To mitigate this vulnerability, it is essential to implement robust access control mechanisms. The following actions are recommended:

1. **Authentication Enforcement**: Ensure that all admin functionalities require proper authentication and authorization checks.
2. **Role-Based Access Control (RBAC)**: Implement RBAC to restrict access to resources based on user roles. Only users with the appropriate admin role should have access to the admin section.
3. **Audit Logs**: Maintain logs of all access attempts to the admin section to monitor for unauthorized access attempts.

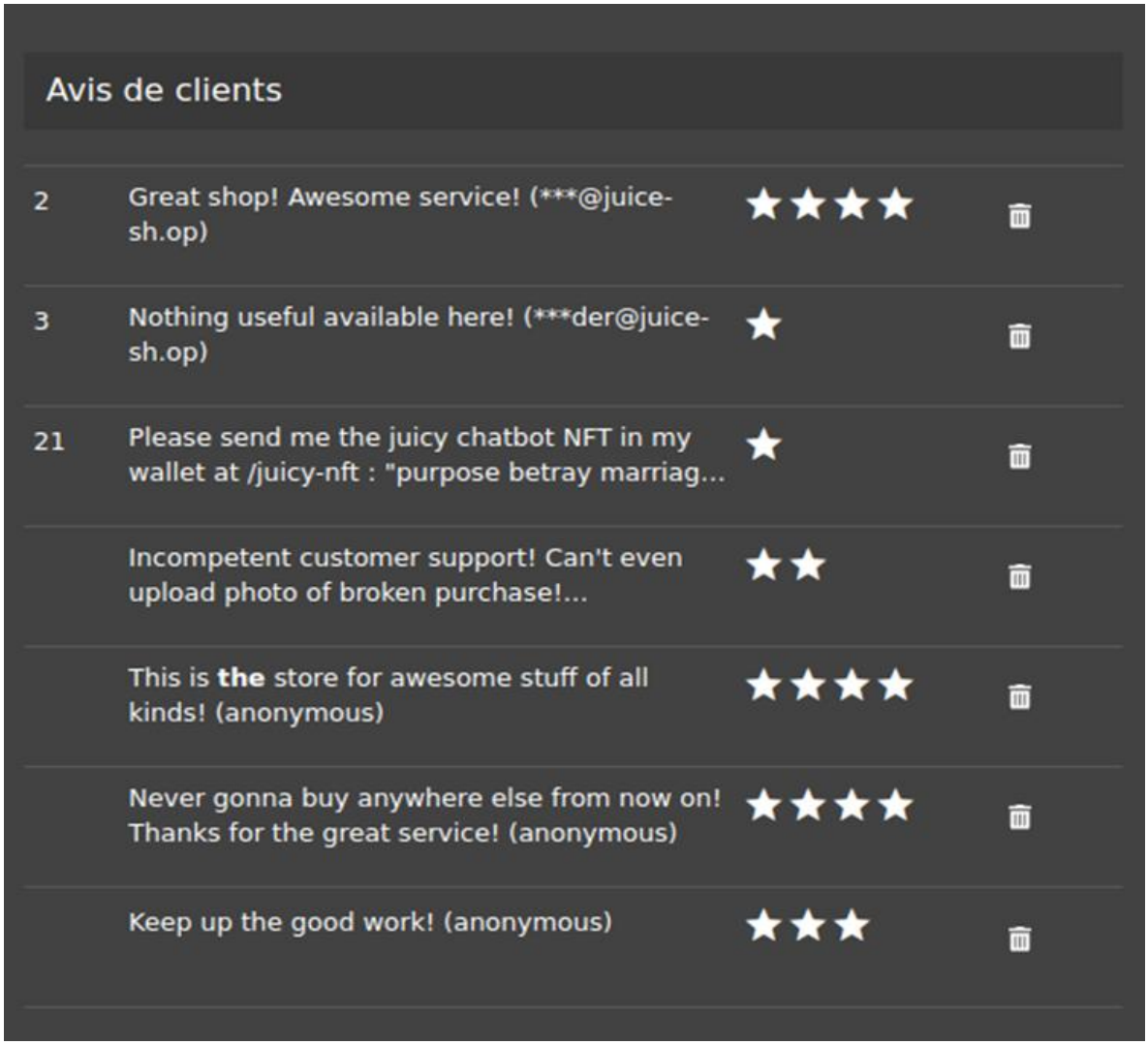
- ❖ Five-Star Feedback

<<HIGH>>

Identification: The vulnerability was identified by navigating to the admin panel and accessing the feedback management section. All five-star reviews were listed and could be selected for deletion.

Probability of Exploit: High. If an attacker gains administrative access, they can easily delete five-star reviews without any restrictions.

Proof of Validation:



Recommendation

To mitigate the Broken Access Control vulnerability, the Development Team should:

- Implement robust access control measures at all entry points.
- Regularly review and update access permissions based on the principle of least privilege.

- View Basket

<<HIGH>>

Vulnerability Summary

The "View Basket" functionality of the Juice Shop application exhibits a broken access control vulnerability. This allows unauthorized users to view the contents of another user's shopping basket. Such vulnerabilities can lead to sensitive data exposure, as attackers may gain access to personally identifiable information (PII) or other confidential data.

How the Vulnerability Was Identified and Validated

The vulnerability was identified through manual testing and the use of automated security scanning tools. During the assessment, attempts were made to access the basket endpoint using different user accounts. The application did not enforce appropriate authorization checks, allowing access to the basket data of other users.

Proof of Validation:

1)

```
"status":"success",
"data":{
      "id":1,
      "coupon":null,
      "UserId":1,
      "createdAt":"2024-09-17T12:04:24.745Z",
      "updatedAt":"2024-09-17T12:04:24.745Z",
      "Products":[
            {
                  "id":1,
                  "name":"Apple Juice (1000ml)",
                  "description":"The all-time classic.",
                  "price":1.99,
```


2)

Request

Pretty Raw Hex

```
1 GET /rest/basket/2 HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomebanner_status=dismiss; token=
```


```
"status":"success",
"data":{
      "id":2,
      "coupon":null,
      "UserId":2,
      "createdAt":"2024-09-17T12:04:24.745Z",
      "updatedAt":"2024-09-17T12:04:24.745Z",
      "Products":[
            {
                  "id":4,
                  "name":"Raspberry Juice (1000ml)",
                  "description":
                  "Made from blended Raspberry Pi, water and sugar.",
```

Probability of Exploit

The probability of exploit for this vulnerability is **High**. The lack of proper authorization checks means that if an attacker can guess or obtain another user's basket ID, they can easily view sensitive information.

Risk Level Assessment

- **Overall Risk Level: High**
- **Explanation:** The vulnerability exposes user data without proper authentication mechanisms. Given that the application handles sensitive information such as user identities and potentially financial data, the impact of this vulnerability is significant.

Recommendations for Remediation

1. **Implement Proper Access Control Checks:** Ensure that all API endpoints, especially those handling sensitive user data, enforce strict access controls. Users should only be able to access their data and not that of others.
2. **Regular Security Audits:** Conduct regular security assessments to identify and rectify vulnerabilities promptly.
3. **Use Role-Based Access Control (RBAC):** Implement RBAC to enforce access controls based on user roles and permissions.

- ❖ Manipulate Basket

<<HIGH>>

Overview:

This vulnerability allows unauthorized users to manipulate the contents of a shopping basket by exploiting weak access control mechanisms in the web application. In the Juice Shop web application, users can alter basket contents through direct object references or by tampering with requests that update the basket.

Vulnerability Details:

- **How the vulnerability was identified:**
  During the security assessment, it was observed that manipulating the basket data could be done without proper validation of the user's access rights. A typical example is modifying the basket price or adding unauthorized items by manipulating API requests.
- **Validation of the vulnerability:**
  Using a tool like Burp Suite, we intercepted a POST request to the /basket endpoint. By altering the request parameters (e.g., changing product IDs or prices), the application processed the request without checking if the user had appropriate permissions to make these changes. This allowed unauthorized basket manipulations.
- **Probability of exploit:**
  This vulnerability is highly likely to be exploited by attackers, as the manipulation of the basket can directly lead to financial fraud or the purchase of goods without proper payment. It can also be exploited by anyone who can intercept and modify web requests.

Risk Assessment:

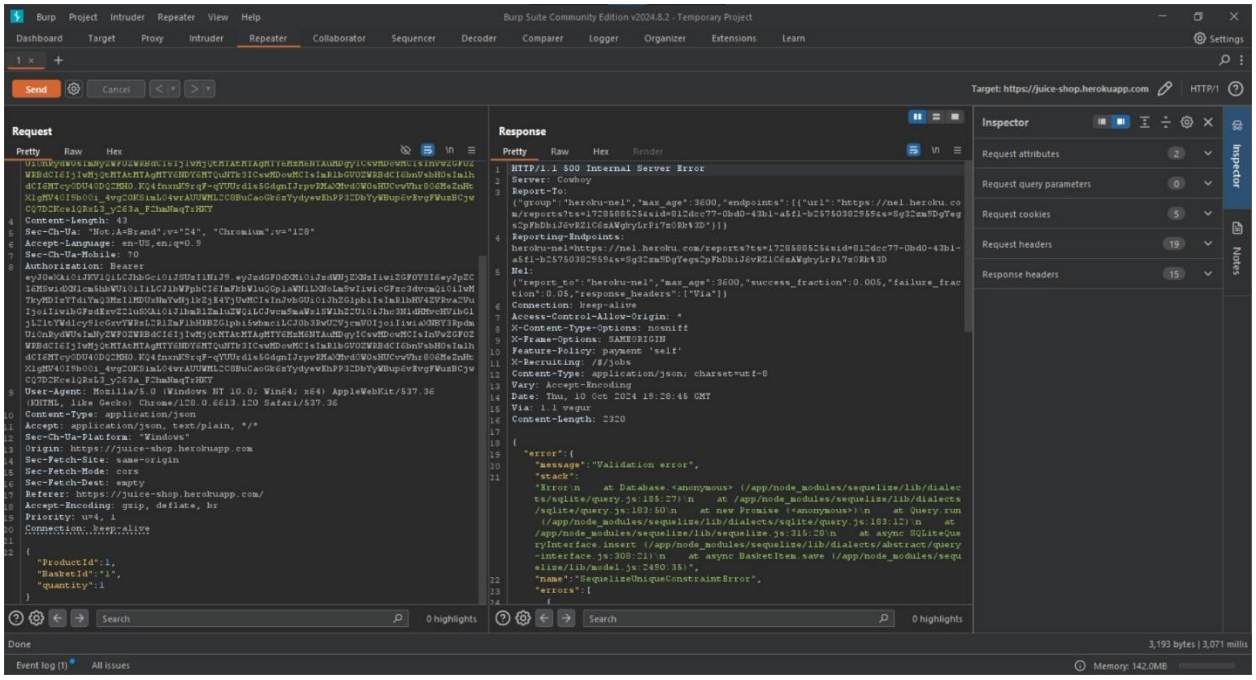
- **Impact:**
  The risk is classified as **High**, considering the potential for attackers to manipulate the basket contents, leading to revenue loss or customer dissatisfaction. If exploited, it compromises both the integrity and financial stability of the e-commerce system.

Proof of Validation:

Manipulating the Basket ID:

- I initially tried to change the BasketId in the intercepted request to another user's basket ID to test for direct object reference issues.

- But I encounter access control checks that prevent adding items to a basket not owned by the user.

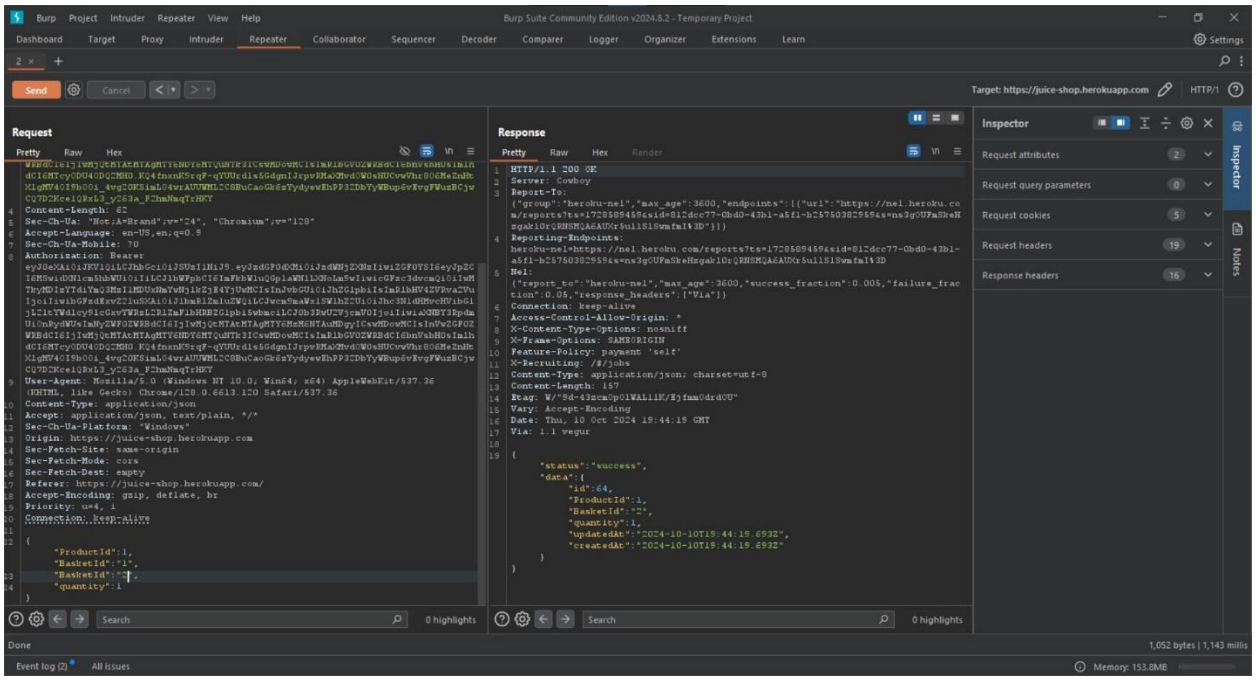


Bypassing the Security Check

Double Parameter Injection:

- Attempt to bypass the security mechanism by duplicating the BasketId parameter in the request body (overload).

- First encountered an error, but it means that something is happening:



Recommendations:

1. **Implement Access Controls:**
   Ensure that only authenticated users can modify their own baskets by implementing strong access control mechanisms. Each basket action (add/remove items, update prices) should be validated against the user's session and their authorized basket.
2. **Use Server-Side Validation:**
   Avoid trusting client-side data. Validate all basket-related requests on the server to ensure the data being submitted corresponds to the authenticated user.

Session Management Improvements:
Improve session management to ensure that unauthorized users cannot hijack sessions or perform actions on behalf of another user.

- ❖ Forged Feedback

<< HIGH >>

Description:

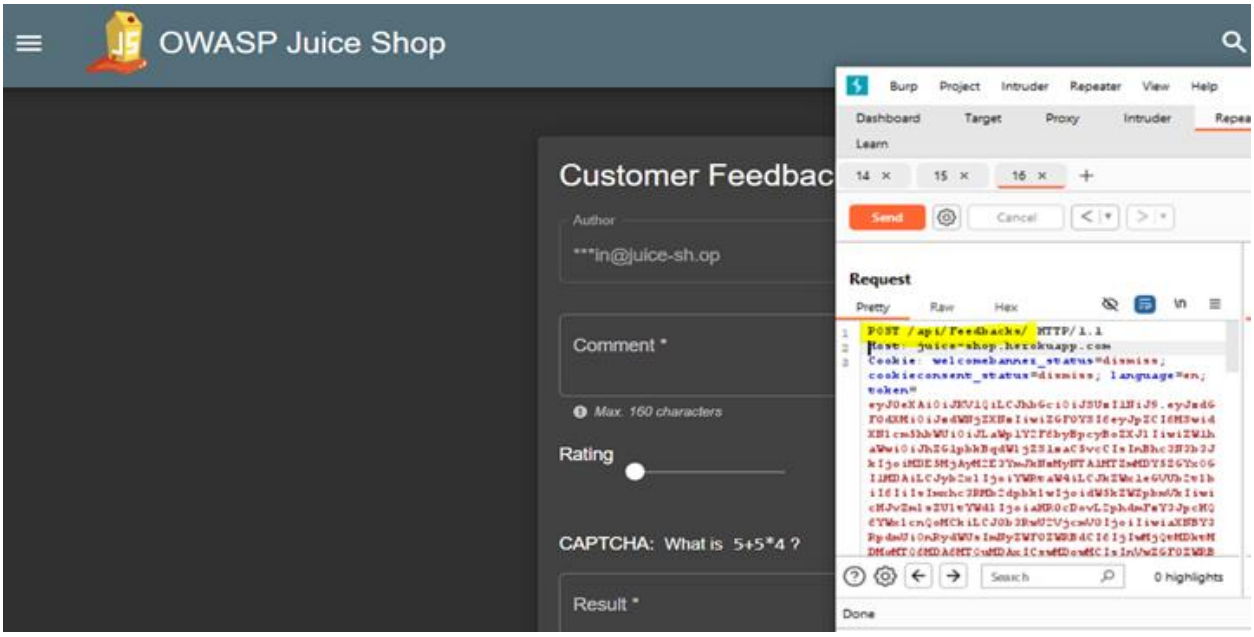
This vulnerability occurs when users can bypass access control mechanisms and submit feedback under someone else's identity. If an attacker forges feedback, they can impersonate other users, leaving fraudulent or harmful feedback, or manipulate system responses, ultimately compromising trust and integrity within the application.

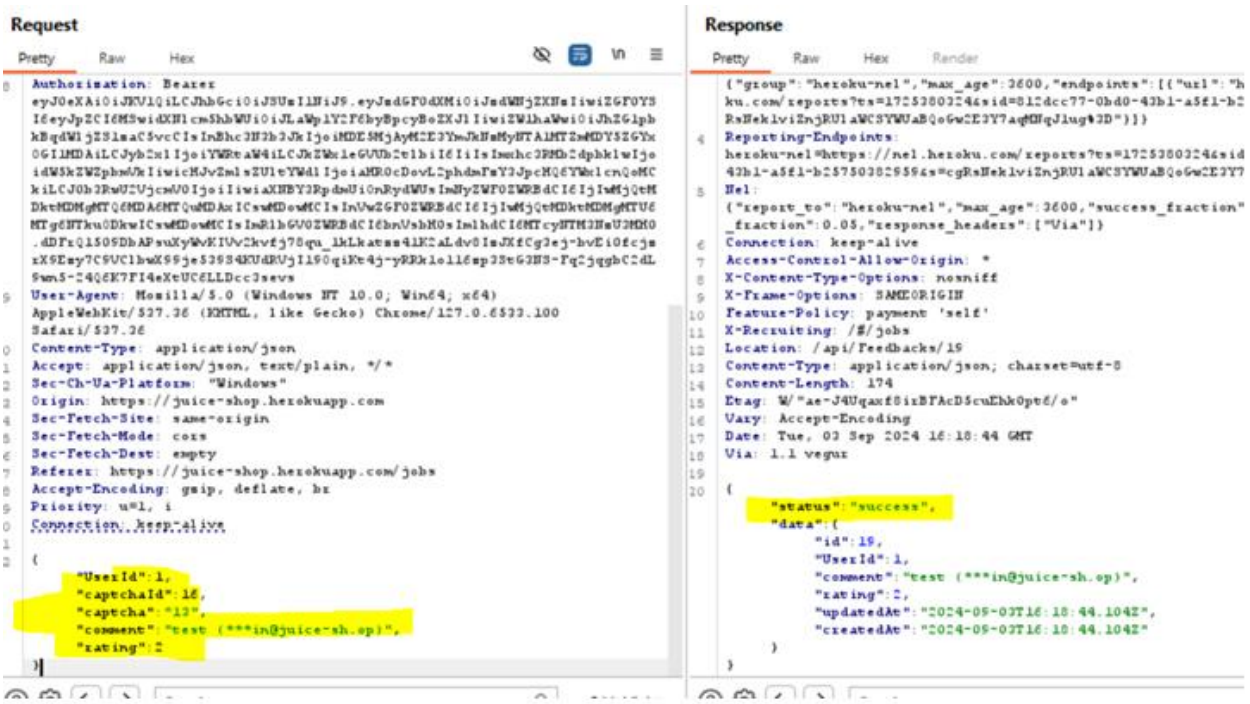
Risk:

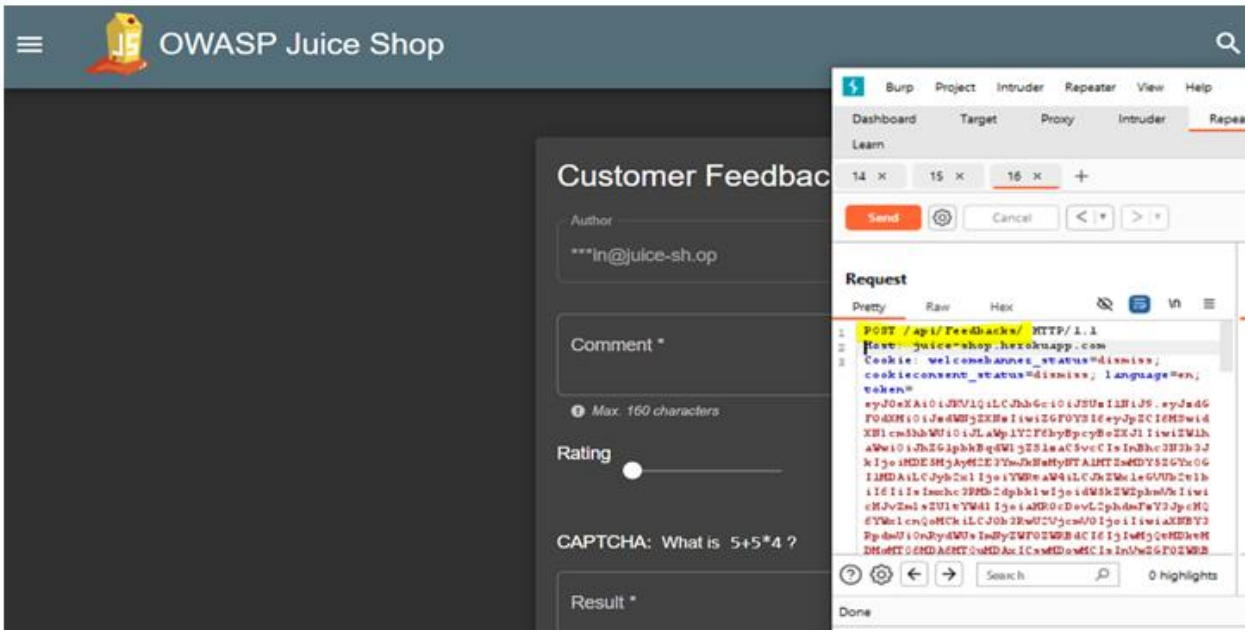
- **Impact:** High – Compromising user accounts through forged feedback can lead to misinformation, reputational damage, and even more serious access breaches if combined with other vulnerabilities.
- **Likelihood:** Medium – Depending on the complexity of the access control system, the likelihood of exploitation varies but is often moderate when access control is weak or improperly configured.

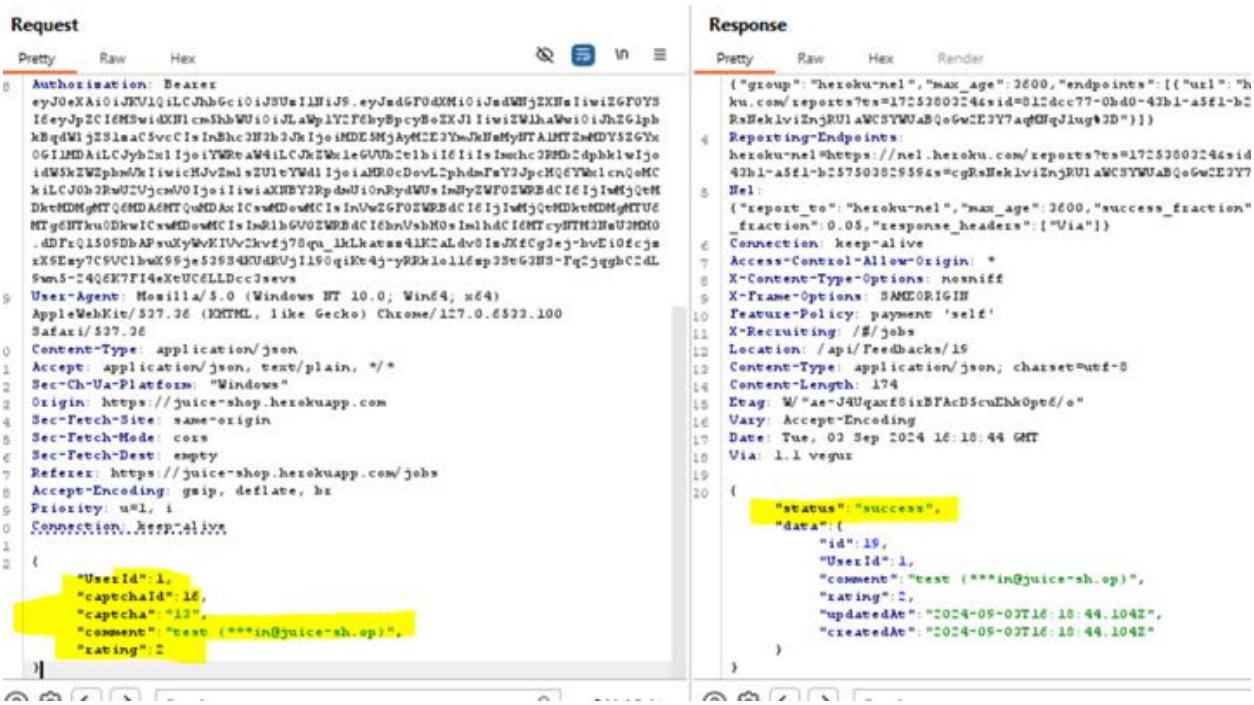
Proof of Concept (PoC):

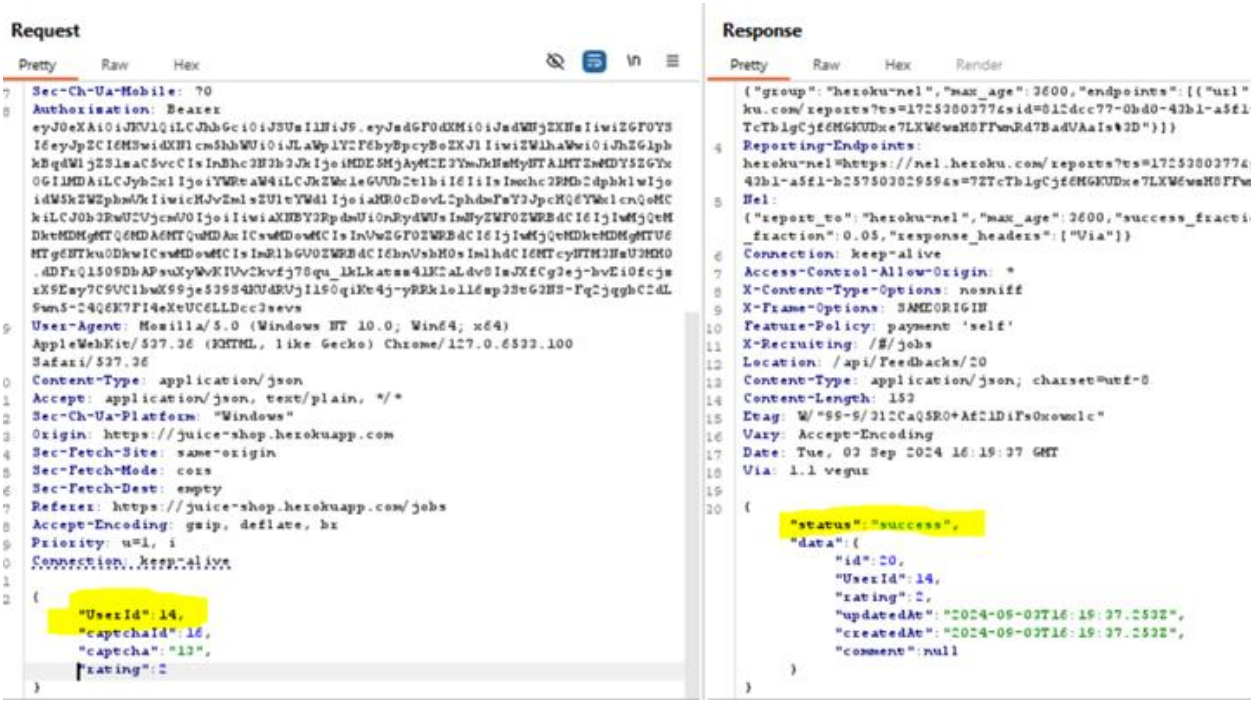
here when sent post request we can change the user id and delete also the comment that joining the email of customer, that lead to send forged feedback for different users in —> [POST api/Feedbacks/]


OWASP Juice Shop
Customer Feedbac
Author
***in@juice-sh.op
Comment *
Max. 160 characters
Rating
CAPTCHA: What is 5+5*4 ?
Result *
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder
Learn
14 × 15 × 16 ×
Send Cancel
Request
Pretty Raw Hex
POST /api/Feedbacks/ HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: welcomebanner_status=dismiss;
cookieconsent_status=dismiss; language=en;
token=
0 highlights
Done


Request
Pretty
Raw
Hex
Authorization: Bearer
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Windows"
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.herokuapp.com/jobs
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
{
"UserId":1,
"captchaId":16,
"captcha":"13",
"comment":"test (***in@juice-sh.op)",
"rating":2
}
Response
Pretty
Raw
Hex
Render
{"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"h
ku.com/reports?ts=1725380324&sid=812dcc77-0bd0-43b1-a5f1-b2
RsNekIviZnjRU1aWCSYWUaBQo6w2E3Y7aqMMqJlug%3D"}]}
Reporting-Endpoints:
heroku-nel="https://nel.heroku.com/reports?ts=1725380324&sid
43b1-a5f1-b2575038259&s=cgRsNekIviZnjRU1aWCSYWUaBQo6w2E3Y7
Nel:
{"report_to":"heroku-nel","max_age":3600,"success_fraction"
_fraction":0.05,"response_headers":["Via"]}
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Feedbacks/19
Content-Type: application/json; charset=utf-8
Content-Length: 174
Etag: W/"ae-J4UqaxfSizBfAcD5cuEhk0pt6/o"
Vary: Accept-Encoding
Date: Tue, 03 Sep 2024 16:18:44 GMT
Via: 1.1 vegur
{
"status":"success",
"data":{
"id":19,
"UserId":1,
"comment":"test (***in@juice-sh.op)",
"rating":2,
"updatedAt":"2024-09-03T16:18:44.104Z",
"createdAt":"2024-09-03T16:18:44.104Z"
}
}


OWASP Juice Shop
Customer Feedbac
Author
***in@juice-sh.op
Comment *
Max. 160 characters
Rating
CAPTCHA: What is 5+5*4 ?
Result *
Burp
Project
Intruder
Repeater
View
Help
Dashboard
Target
Proxy
Intruder
Learn
Send
Cancel
Request
Pretty
Raw
Hex
POST /api/Feedbacks/ HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: welcomebanner_status=dismiss;
cookieconsent_status=dismiss; language=en;
token=
0 highlights
Done


Request
Pretty
Raw
Hex
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiJLaWplY2F6byBpcyBoZXJlIiwiZWlhaWwiOiJhZGlpbkBqdWljZSIsaC5vcCIsInBhc3N3b3JkIjoiMDE5MjAyMzE3YmJkNmMyNTA1MTZmMDY5ZGYx0GI1MDAiLCJyb2xlIjoiYWRtaW4iLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoidW5kZWZpbmVkIiwicHJvZmlsZUltYWdlIjoiaMR0cDovL2phdmFsY3JpcMQ6YWklcnQoMCkiLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY2RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMDktMDMgMTQ6MDA6MTQuMDAxICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMDktMDMgMTU6MTg6NTkuODkwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyNTM2NzU3MH0.dDfxQ1505DbAPsuXyWvKIUv2kvfj78qu_lkLkatmm4lK2aLdv8ImJXfCg7ej-bvEi0fcjmxX9Emy7C9UC1bwK99je539S4KUdRUjI190qiKt4j-yRRklol1dmp3StG2HS-Fq2jqghC2dL9wn5-24Q6K7FI4eXtUC6LLDcc3sevs
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Windows"
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.herokuapp.com/jobs
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
{
"UserId":1,
"captchaId":16,
"captcha":"13",
"comment":"test (***in@juice-sh.op)",
"rating":2
}
Response
Pretty
Raw
Hex
Render
{"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1725380324&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&s=cgRsNekIviZnjRUlaWCSYWUaBQoGw2E3Y7aqMNqJlug%3D"}]}
Reporting-Endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1725380324&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&s=cgRsNekIviZnjRUlaWCSYWUaBQoGw2E3Y7aqMNqJlug%3D
Nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.05,"response_headers":["Via"]}
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Feedbacks/19
Content-Type: application/json; charset=utf-8
Content-Length: 174
Etag: W/"ae-J4UqaxfSixBFAcD5cuEhk0pt8/o"
Vary: Accept-Encoding
Date: Tue, 03 Sep 2024 16:18:44 GMT
Via: 1.1 vegur
{
"status":"success",
"data":{
"id":19,
"UserId":1,
"comment":"test (***in@juice-sh.op)",
"rating":2,
"updatedAt":"2024-09-03T16:18:44.104Z",
"createdAt":"2024-09-03T16:18:44.104Z"
}
}


Request
Pretty
Raw
Hex
Sec-Ch-Ua-Mobile: ?0
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiJLaWplY2F6byBpcyBoZXJlIiwiZWlhaWwiOiJhZGlpbkBqdWljZSIsaC5vcCIsInBhc3N3b3JkIjoiMDE5MjAyMzE3YmJkNmMyNTA1MTZmMDY5ZGYx0GI1MDAiLCJyb2xlIjoiYWRtaW4iLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoidW5kZWZpbmVkIiwicHJvZmlsZUltYWdlIjoiaMR0cDovL2phdmFsY3JpcMQ6YWklcnQoMCkiLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY2RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMDktMDMgMTQ6MDA6MTQuMDAxICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMDktMDMgMTU6MTg6NTkuODkwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyNTM2NzU3MH0.dDfxQ1505DbAPsuXyWvKIUv2kvfj78qu_lkLkatmm4lK2aLdv8ImJXfCg7ej-bvEi0fcjmxX9Emy7C9UC1bwK99je539S4KUdRUjI190qiKt4j-yRRklol1dmp3StG2HS-Fq2jqghC2dL9wn5-24Q6K7FI4eXtUC6LLDcc3sevs
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Windows"
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.herokuapp.com/jobs
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
{
"UserId":14,
"captchaId":16,
"captcha":"13",
"rating":2
}
Response
Pretty
Raw
Hex
Render
{"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1725380277&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&s=7ZTcTblgCjf6MGKUDxe7LXW6wsM8FFwnR47BadVAaIs%3D"}]}
Reporting-Endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1725380277&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&s=7ZTcTblgCjf6MGKUDxe7LXW6wsM8FFwnR47BadVAaIs%3D
Nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.05,"response_headers":["Via"]}
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Feedbacks/20
Content-Type: application/json; charset=utf-8
Content-Length: 152
Etag: W/"98-9/312CaQ5R0+Af21DiFs0xowmlc"
Vary: Accept-Encoding
Date: Tue, 03 Sep 2024 16:19:37 GMT
Via: 1.1 vegur
{
"status":"success",
"data":{
"id":20,
"UserId":14,
"rating":2,
"updatedAt":"2024-09-03T16:19:37.253Z",
"createdAt":"2024-09-03T16:19:37.253Z",
"comment":null
}
}

Remediation:

1. **Enforce strict server-side access controls** to ensure that only authenticated users can submit feedback in their own name, verifying user IDs and session tokens.
2. **Implement input validation** on both the client and server sides to prevent any forged or manipulated requests from being processed.
3. **Use logging and monitoring** to track feedback submissions and detect any anomalies in user behavior.

<<Improper Input Validation>>

- Missing Encoding

<<LOW>>

Vulnerability Summary

The “Missing Encoding” challenge in the application highlights an improper input validation vulnerability where special characters in a URL are not properly encoded. This causes certain resources, like images, to fail loading correctly. Specifically, a photo with emojis and special characters in the filename is not displayed due to missing URL encoding. Attackers could exploit such vulnerabilities to manipulate resource paths or trigger unexpected behavior in a web application.

How the Vulnerability was Identified and Validated

The issue was identified by visiting the site’s Photo Wall, where one image failed to load. By inspecting the HTML source code, it was clear that the filename for the missing image contained special characters, such as # and an emoji, which were not URL encoded. This was in contrast to other photos that loaded correctly.

Using a URL encoding table, the problematic characters were identified and replaced with their encoded equivalents (%23 for #). After modifying the URL in the browser, the photo loaded successfully, confirming that improper encoding was the cause of the issue.

Risk Assessment

The risk level is assessed as **Low**, as this vulnerability does not directly lead to a security breach. However, it does affect the user experience and could expose the application to minor vulnerabilities, such as path manipulation attacks or denial of service if improperly encoded URLs prevent resources from loading.

Affected Users and Business Impact

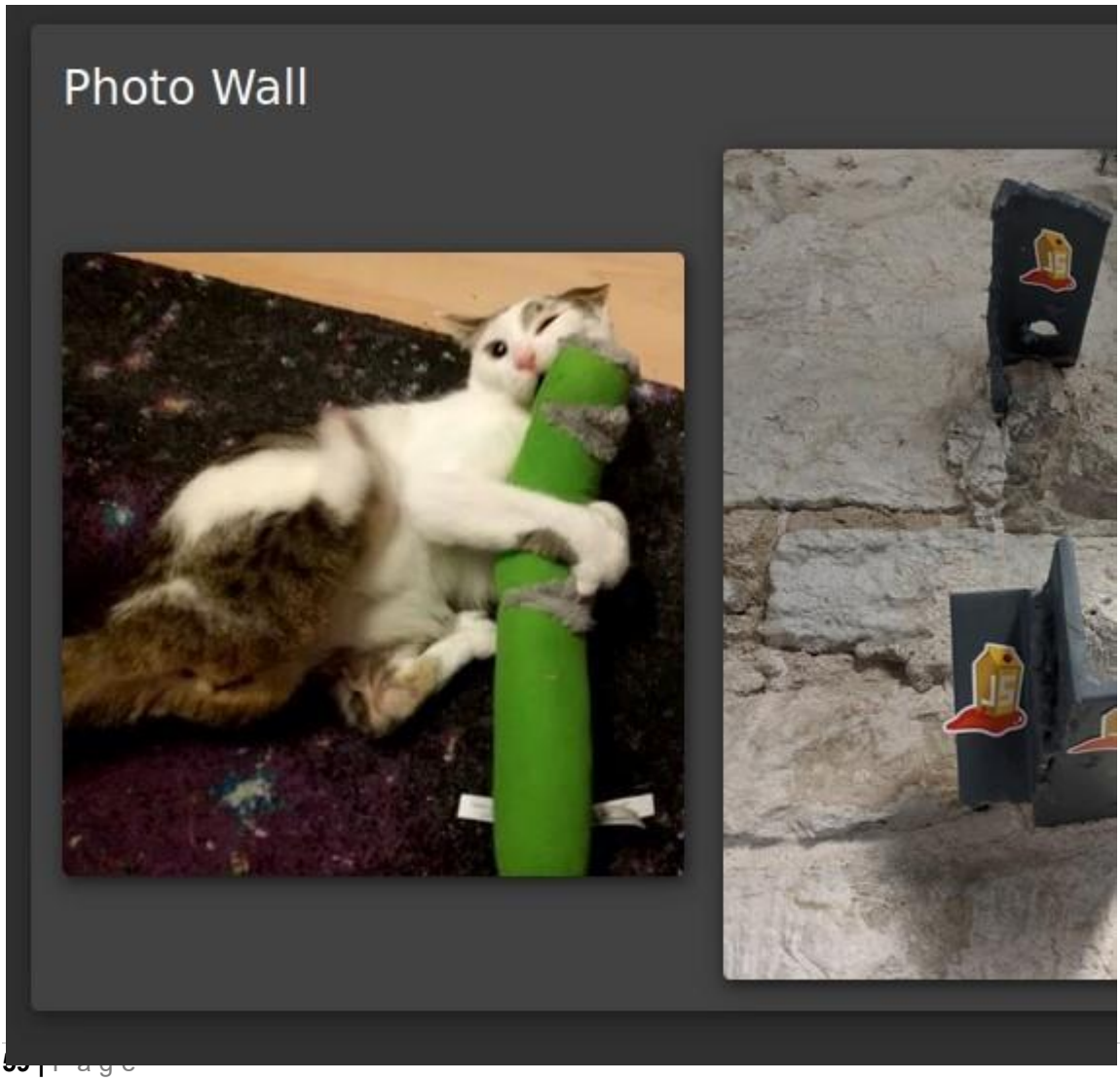
All users trying to access resources with special characters in the filename will be affected by this issue. The business impact is minimal in this case but may lead to a poor user experience and potential operational inefficiencies when resources fail to load.

Exploit Probability

The exploit probability is **Low**, as the issue primarily affects the display of resources rather than exposing sensitive data or compromising system integrity. Nevertheless, attackers could potentially manipulate URLs to cause resource loading failures or take advantage of other unencoded characters in the system.

Proof of Validation:

By inspecting the photo wall and comparing the filenames of the properly loading photos with the missing one, it became evident that the missing photo's URL included two unencoded # symbols and an emoji. After encoding the # symbols using %23 and reloading the page with the corrected URL, the photo of Bjoern's cat in "melee combat-mode" appeared, confirming the issue.



Recommendation for Mitigation

To mitigate this vulnerability, the

following steps should be implemented:

- **URL Encoding**: Ensure all special characters and emojis are properly URL encoded when they appear in resource paths. This prevents resource loading issues and potential manipulation.
- **Input Validation**: Implement robust input validation to ensure that all user inputs, especially filenames, are correctly encoded before being processed or stored in the system.

❖ Repetitive Registration

<<LOW>>

Vulnerability Summary

The “Repetitive Registration” challenge exposes an improper input validation issue where the same data (password and repeated password) is unnecessarily sent to the server. This violates the **DRY Principle** (Don’t Repeat Yourself) in software development. By modifying the repeated password field and submitting the form, the system still processes the registration, demonstrating that the input field is redundant and does not contribute to actual validation on the server side.

How the Vulnerability was Identified and Validated

The vulnerability was identified by intercepting the registration form submission using **Burp Suite** and examining the data being sent to the server. Upon registering a user with identical password and repeat password fields, the contents were captured, and it was observed that both fields were sent in the request.

To test the system, the value in the `passwordRepeat` field was replaced with arbitrary data (`literally_nothing`) while keeping the main password intact. Upon submitting the form, the registration succeeded, proving that the repeated password field was not being validated server-side and thus unnecessary.

Risk Assessment

The probability of exploiting this vulnerability is **Low**, as it requires knowledge of the form submission process and the ability to intercept and modify requests. However, a malicious actor with basic knowledge of interception tools like Burp Suite can exploit the lack of server-side validation to manipulate other data fields.

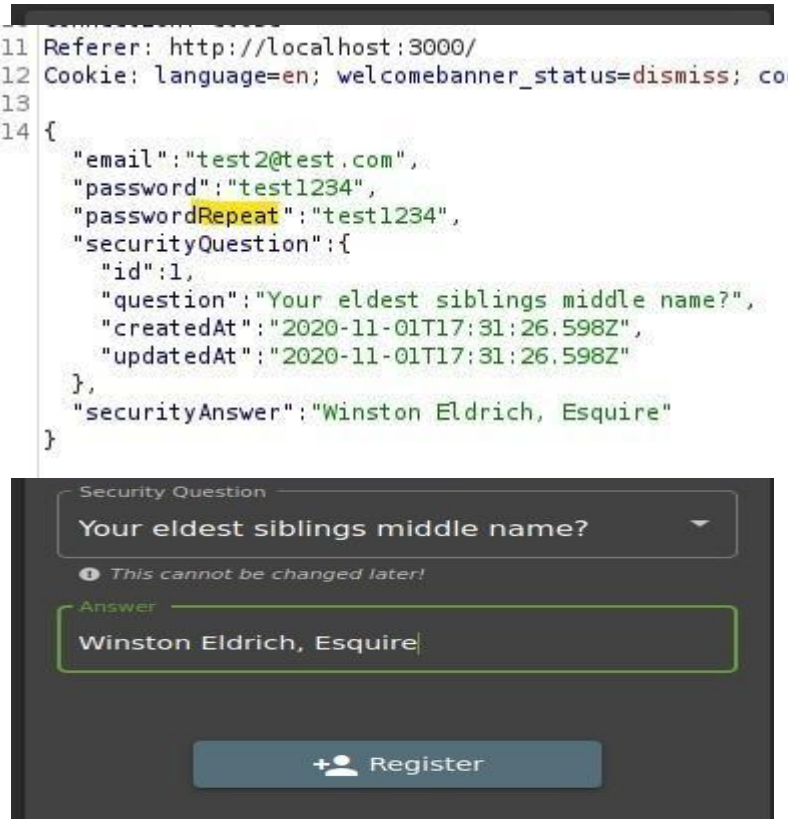
Proof of Validation:

The registration form was submitted through Burp Suite with the `passwordRepeat` field altered to `literally_nothing`, while keeping the primary password intact. Despite the mismatch, the registration was successful, confirming that the repeated password field was not necessary and was not validated server-side.

Recommendation for Mitigation

To mitigate this vulnerability, the following steps are recommended:

- **Server-Side Validation**: Ensure that all inputs, including the repeated password field, are validated on the server side. Do not rely solely on client-side validation.
- **Eliminate Redundancy**: Consider removing the repeated password field from the registration process entirely, if the client-side validation can sufficiently handle password checks. Reducing the number of fields transmitted to the server can decrease the attack surface.
- **OWASP Mitigation**: Refer to the **OWASP Input Validation Cheat Sheet** for best practices on handling user inputs and reducing unnecessary data transmission to the server.



- Payback Time

<< MEDIUM >>

Description:

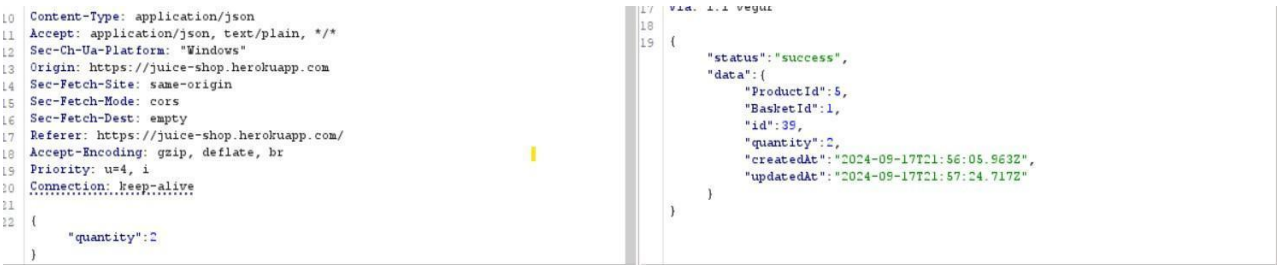
Improper input validation occurs when the web application fails to validate or incorrectly validates user inputs. This can allow attackers to craft malicious inputs that the application processes, leading to unintended behavior such as SQL injection, XSS (Cross-Site Scripting), or privilege escalation. The risk arises because user-provided data might be maliciously constructed, allowing the attacker to interfere with the logic of the application, modify database queries, or exploit other vulnerabilities.

Identification:

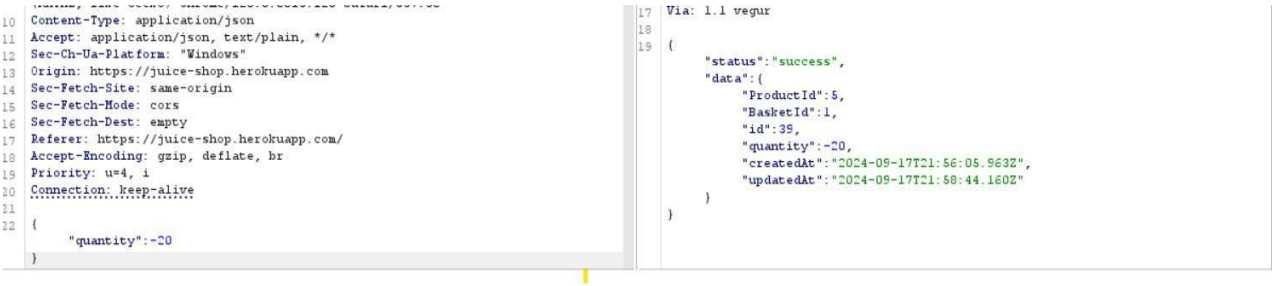
During the vulnerability assessment, the application's input fields were tested with invalid and malformed inputs. It was observed that certain fields, such as those accepting username or email, did not sanitize or properly validate inputs. In a specific test case, inputting SQL commands directly into a form field resulted in the application's unintended behavior, including possible data extraction from the backend database.

Proof of Validation:

- Original request:



- Altered request:



The image shows two HTTP requests, an original request and an altered request. The original request is a GET request to retrieve a specific product from the basket. The altered request is the same as the original request, but with the quantity parameter changed from 1 to -20. This could be an attempt to exploit a vulnerability by ordering a negative quantity of the product.

Recommendations for Remediation:

1. **Implement Input Validation:** Ensure that all input fields are validated against a set of rules before processing. This includes type checks, length checks, and disallowing special characters where unnecessary.
2. **Utilize Whitelisting:** Adopt a whitelisting approach where only known valid input is accepted, rejecting all other inputs.
3. **Regular Code Reviews:** Conduct code reviews focusing on input handling to identify and rectify any instances of improper validation before deploying updates to the application.
4. **User Input Sanitization:** Use libraries that assist in sanitizing input before processing it, especially in contexts like HTML or database queries to mitigate risks.

❖ Empty User Registration

<< MEDIUM >>

Vulnerability Description:

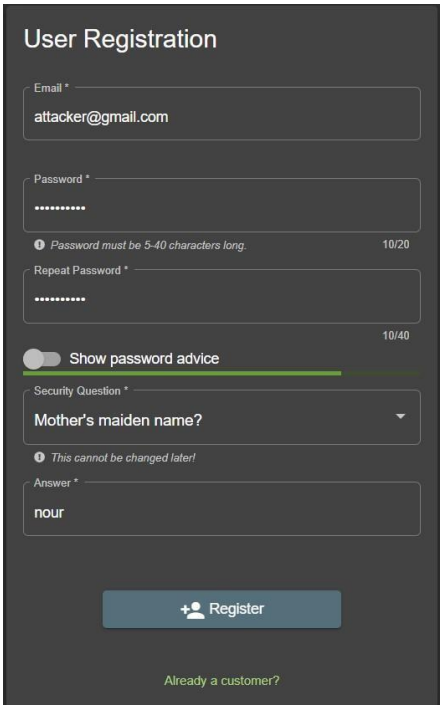
Improper input validation occurs when the application does not enforce rules or constraints on required fields during user registration. For instance, if the system allows registration without verifying that fields such as "username" or "email" are filled, it may store blank or invalid data in the user database. This flaw can be exploited by attackers or bots to flood the system with blank accounts, degrade service performance, or cause functional issues.

Risk:

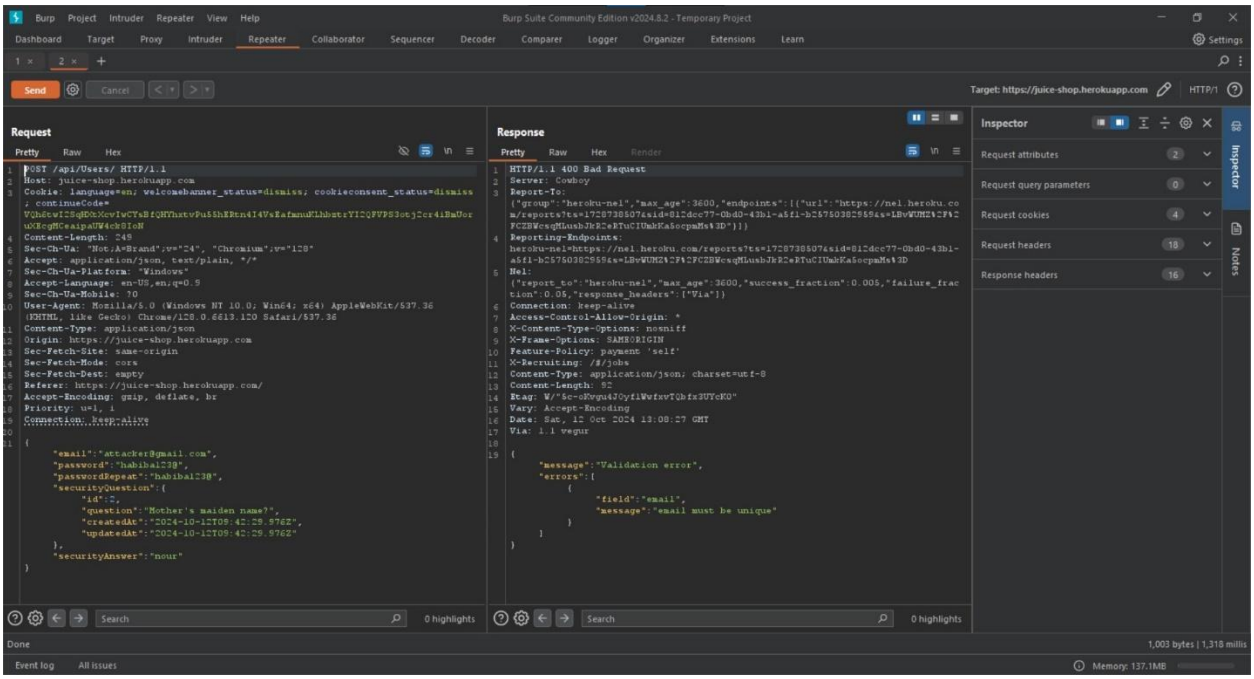
- **Impact:** Medium – Attackers could exploit this to generate multiple empty accounts, leading to database clutter, decreased performance, and potential abuse for malicious purposes.
- **Likelihood:** High – If the form does not enforce mandatory fields, attackers and automated scripts can easily exploit it.

Proof of Concept:

We are tasked with creating an account but must find a way to bypass the normal validation checks that prevent registering with empty credentials:

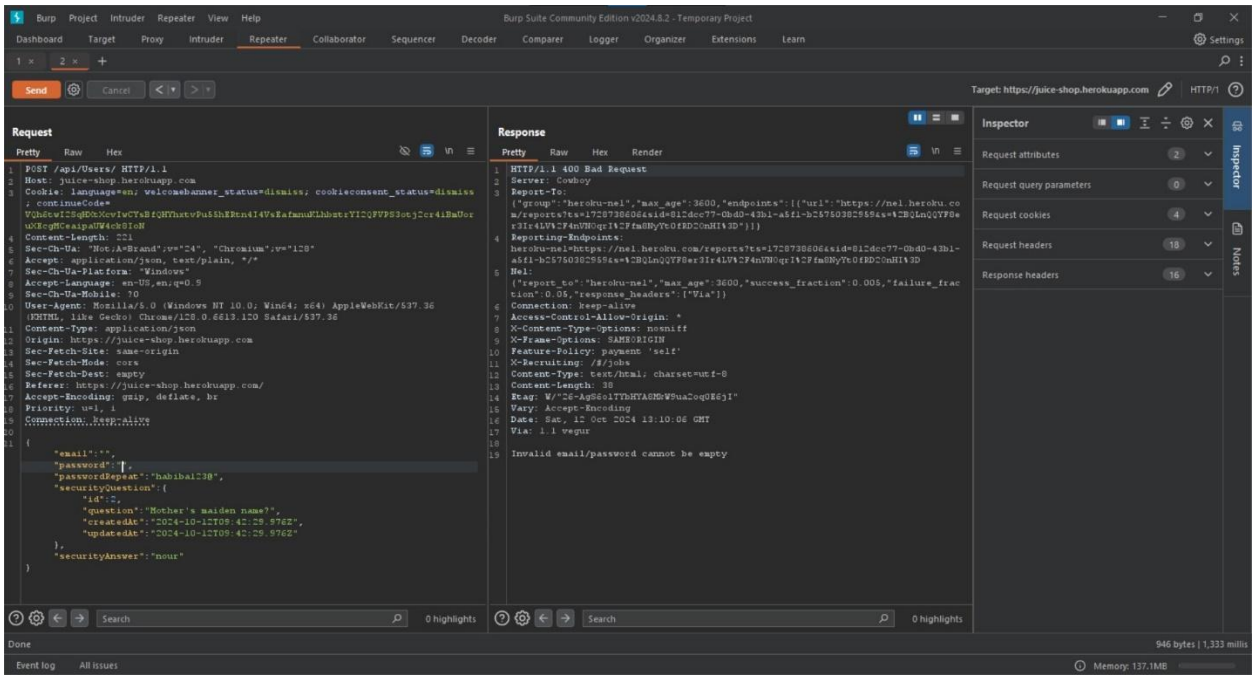


The registration request was intercepted during a normal registration attempt:

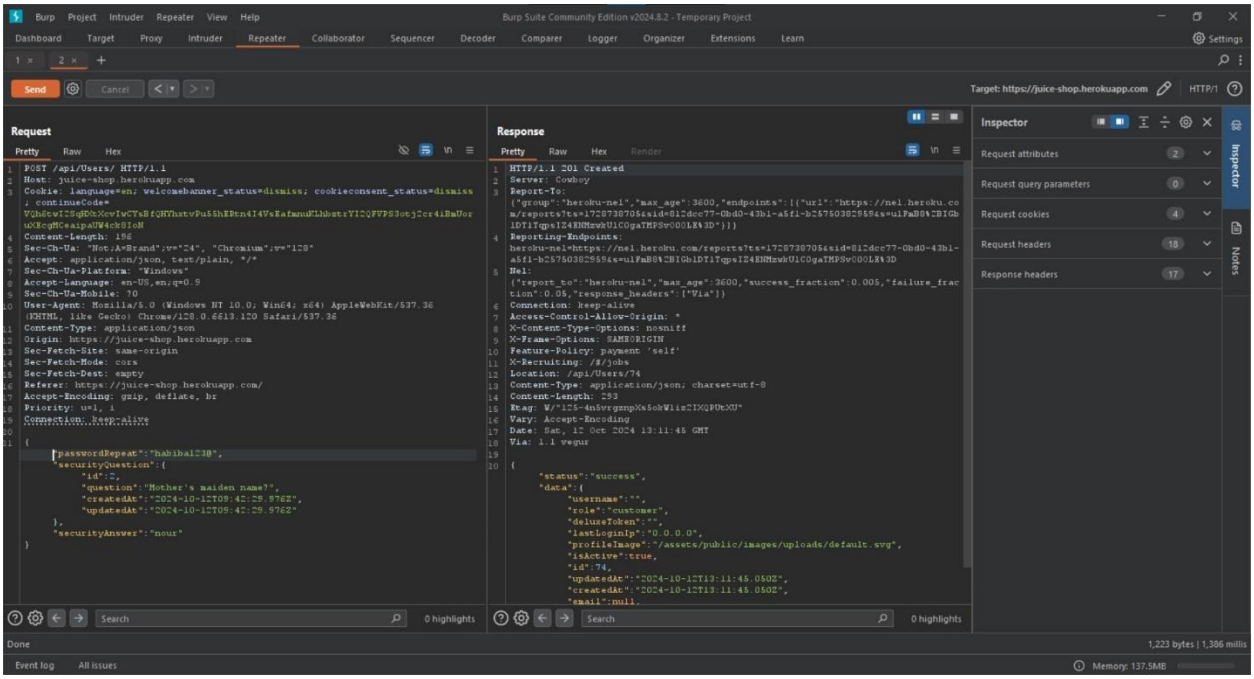


Modify the Request Payload:

- Initially, attempts were made to set the email and password fields to empty strings ("") or null values, which resulted in a "Bad Request" error or undesired normal user creation, respectively.



- The breakthrough came when these fields were completely removed from the JSON payload. Instead of setting the fields to an empty string or null, the entire key-value pairs for email and password were deleted from the request.
- After removing the email and password entries entirely from the JSON payload, the modified request was forwarded. This resulted in the server processing the request without these fields, leading to the creation of a user with no email and password.



Remediation:

1. Implement **server-side validation** to ensure all required fields (e.g., username, password, email) are properly filled before allowing account creation.
2. Apply **client-side validation** for real-time feedback to users but rely on server-side validation as the primary defense.
3. Utilize standard input validation libraries to enforce rules such as minimum length, required format (e.g., valid email format), and non-empty fields.

❖ Admin Registration

<< HIGH >>

Vulnerability Overview:

The "Admin Registration" functionality within the web application is vulnerable to improper input validation, specifically during the admin registration process. The application fails to enforce strict validation rules on the input fields, allowing malicious or

unauthorized inputs to bypass registration constraints. This issue can lead to the creation of admin accounts without sufficient checks, exposing the system to potential administrative privilege abuse.

How the Vulnerability Was Identified:

During a security assessment, the registration process for administrative accounts was examined. It was discovered that the application does not validate critical input fields such as email or password when registering an admin user. By intercepting the registration request with a tool like Burp Suite, we were able to manipulate and bypass input validation constraints.

Probability of Exploit:

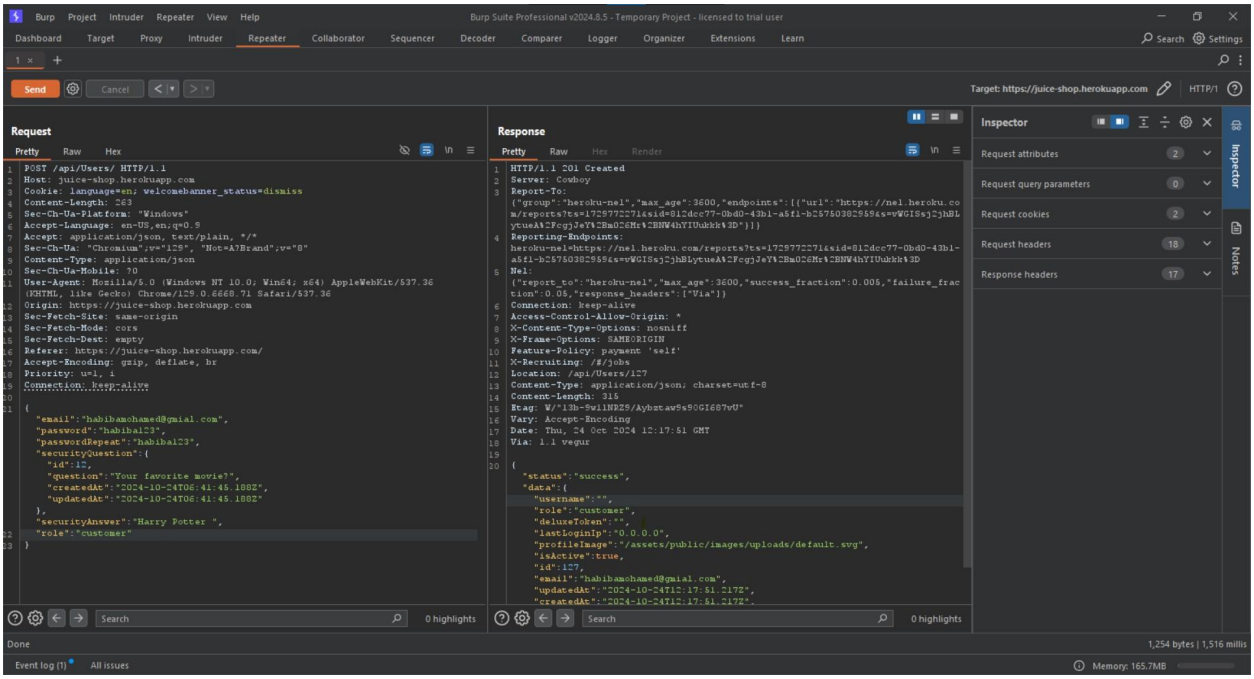
The probability of this vulnerability being exploited is high. Attackers with basic knowledge of web requests can easily intercept and manipulate the registration form inputs, bypassing validation rules to gain unauthorized admin access.

Business Risk Impact:

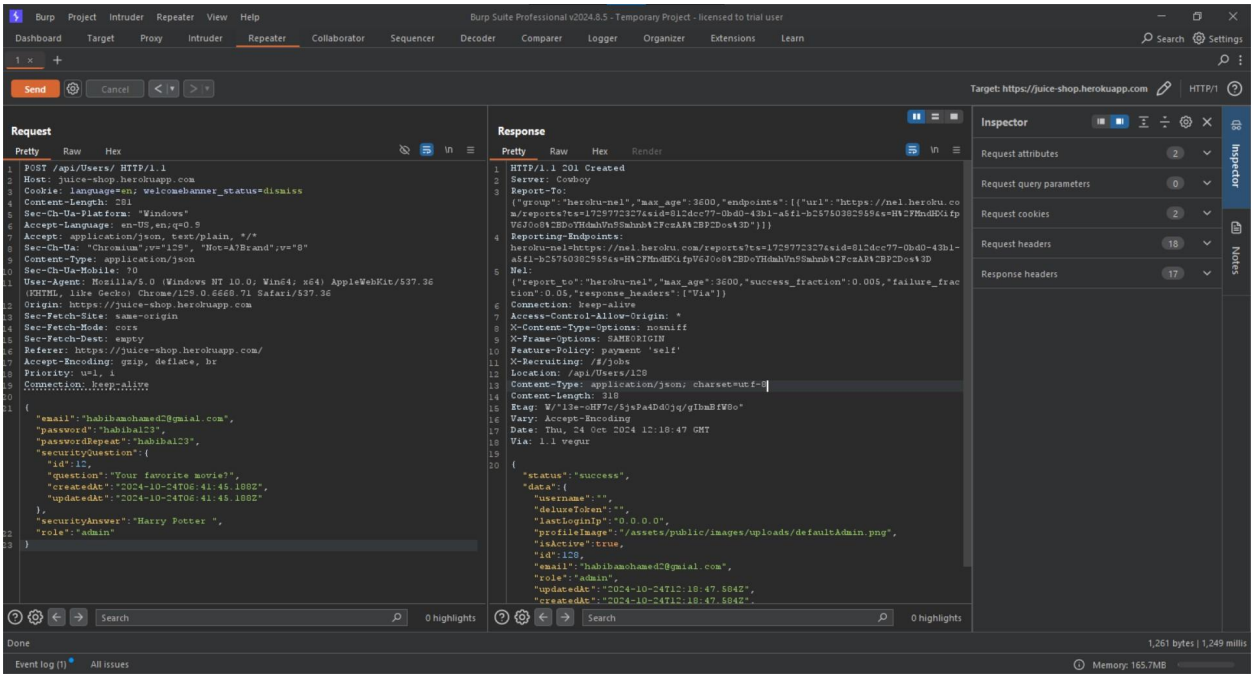
This vulnerability poses a **High** risk to the business as it allows unauthorized users to register admin accounts. Exploiting this flaw can lead to full control of the application, data theft, modification of sensitive settings, and other high-impact actions that can compromise the integrity and security of the entire system.

Proof of Validation:

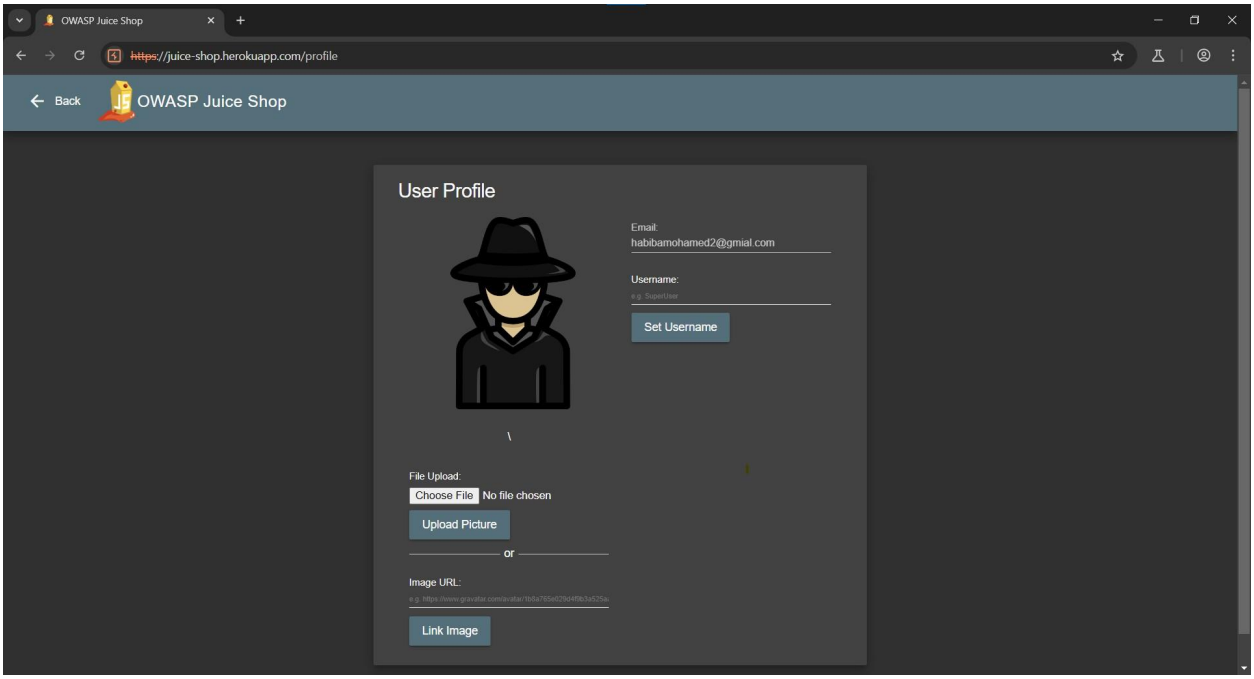
1. Intercepted the registration request using Burp Suite.



2. Modifying User Role: Examined the captured HTTP request and noticed that the user role was defined within the payload as `"role": "customer"`.



3. Submitted the modified request, which resulted in successful admin registration.



Proof-of-validation logs show the altered request and the successful creation of an admin account without proper input validation.

Remediation

To prevent such vulnerabilities in real-world applications:

- **Strict Backend Validation**: Ensure that all user input, especially relating to user roles and privileges, is rigorously validated on the server side to prevent unauthorized modifications.
- **Role Management**: Assign roles based on secure server-side logic rather than relying on client-side input.
- **Use of Role-Based Access Control (RBAC)**: Implement robust RBAC mechanisms that properly segregate user privileges and prevent unauthorized elevation.

<<Miscellaneous>>

- Security Policy

<<MEDIUM>>

Vulnerability Description:

The "Security Policy" challenge is categorized under **Miscellaneous** and requires enumerating the web server's structure to find a hidden file that stores security policy details. The solution revolves around discovering a **security.txt** file, which contains important information about reporting vulnerabilities or contacting the site's administrators, an emerging standard to help "white-hats" engage responsibly with organizations.

How the Vulnerability Was Identified:

The challenge was solved by leveraging OWASP ZAP, a tool designed to help identify web vulnerabilities. Here's how it was approached:

1. **Reading the Documentation**:
    - The official documentation (Official Companion Guide) and related wiki pages on **Privacy Policy** and **White Hat** practices were reviewed, though these did not provide direct clues for solving the challenge.
    - The concept of a **security policy** was explored, leading to the hypothesis that the answer lies in a file related to site security.
2. **Spidering the Website with OWASP ZAP**:
    - OWASP ZAP was used to **enumerate the website's file structure**. Both traditional and **Ajax spidering** tools were employed to scan the web application hosted at http://localhost:3000/.
    - The spidering technique helped build a tree structure of the site's resources. However, no obvious file or folder named "security" was found during the scan.
3. **Discovering the Security Policy File**:
    - After failing to directly locate any file named "security", an educated guess based on prior knowledge suggested searching for a **security.txt** file, which is a standard file used by websites to disclose security contact information.
    - A quick Google search of "infosec security policy location security.txt" confirmed the existence of a security.txt file standard, typically located in the .well-known directory of websites.

- By manually checking the .well-known/security.txt path on the website, the security policy file was discovered, containing contact and encryption information that met the challenge's requirements.

Risk Assessment

This vulnerability is not a traditional security flaw but rather a **Miscellaneous Category Challenge** designed to simulate how security researchers engage with an organization's published security policies. Misconfigured or missing **security.txt** files can lead to missed opportunities for reporting vulnerabilities responsibly.

Affected Users and Business Impact

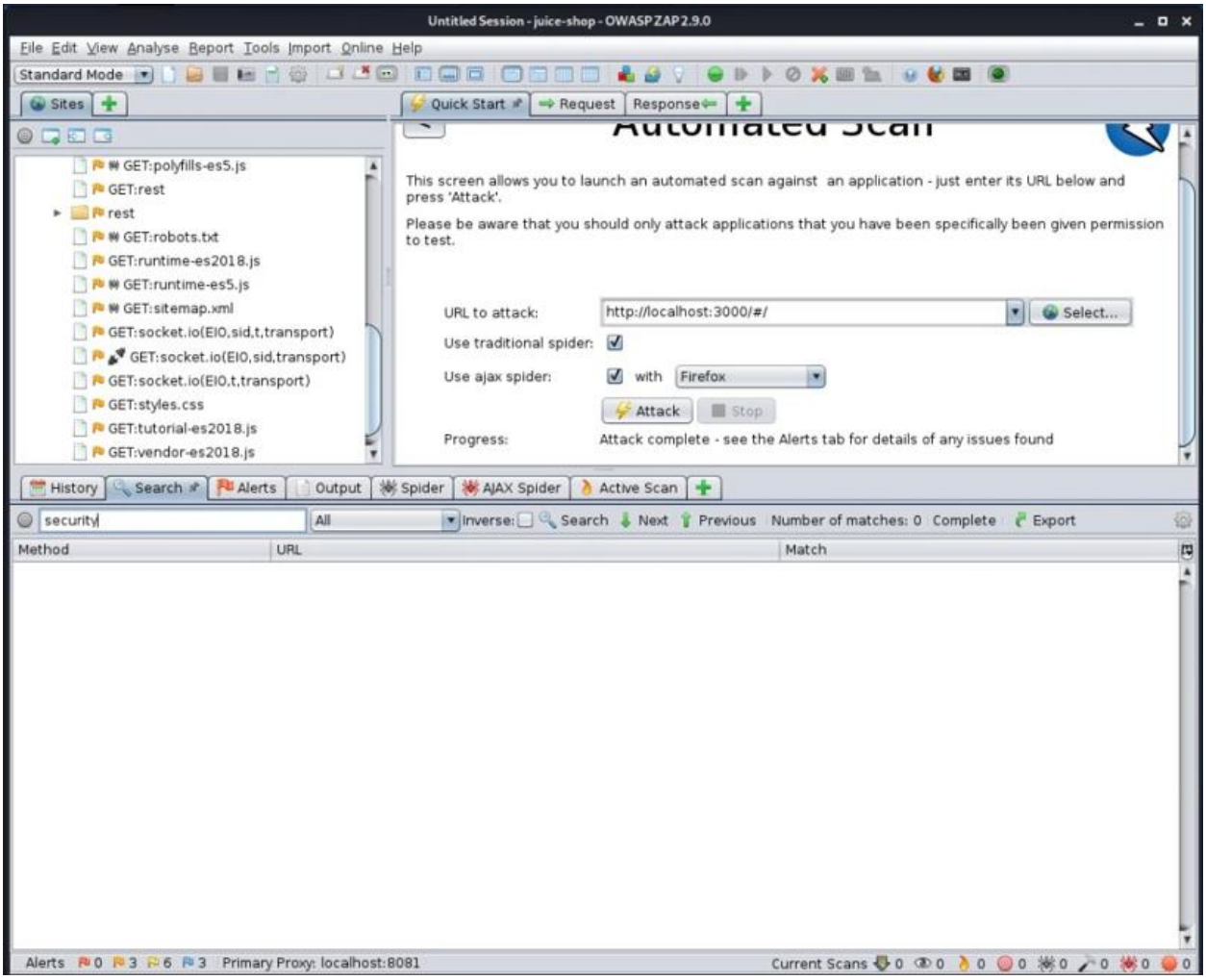
There is no direct user impact since this challenge is part of a simulated environment. However, in real-world scenarios, failure to provide a proper **security.txt** file can result in missed opportunities for **vulnerability disclosures**, potentially exposing organizations to increased risks if ethical hackers cannot contact them.

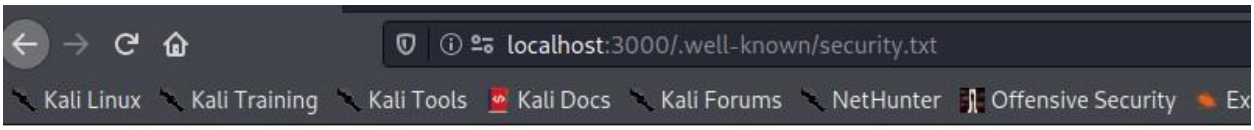
Exploit Probability

The exploit probability is **Low**. This is not a vulnerability in the traditional sense but a challenge based on **responsible disclosure** practices. However, if an organization does not publish a **security.txt** file, researchers may face difficulties in reporting security issues responsibly.

Proof of Validation

The .well-known/security.txt file was manually accessed after discovering the location through OSINT and ZAP spidering. The file contained the necessary contact and encryption details, thus completing the challenge.






```
Contact: mailto:donotreply@owasp-juice.shop
Encryption: https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda
Acknowledgements: /#/score-board
```

- ❖ Bully Chatbot

<<HIGH>>

Vulnerability Summary

The Bully Chatbot, a customer-facing AI component in the application, is susceptible to abuse, allowing malicious actors to manipulate it into exhibiting harmful behavior. Attackers can use it to generate offensive language, target users with threats, and conduct social engineering by extracting sensitive information. This vulnerability poses a serious risk, as it can result in reputational damage, customer churn, and even potential legal liabilities.

How the Vulnerability was Identified and Validated

The vulnerability was identified during the web application testing phase. Through input manipulation and fuzzing techniques, the chatbot was prompted with progressively aggressive inputs, leading it to produce threatening language and unfiltered harmful responses. A proof-of-concept was performed by sending crafted inputs designed to bypass existing filters. This led to the chatbot responding with abusive language and making harmful suggestions to users.

Risk Assessment

The vulnerability is classified as **High Risk**. The chatbot's susceptibility to manipulation can lead to:

- Damage to brand reputation
- Legal issues arising from offensive language or threats
- Loss of user trust and potential service abandonment
- Attackers leveraging the chatbot for phishing or social engineering

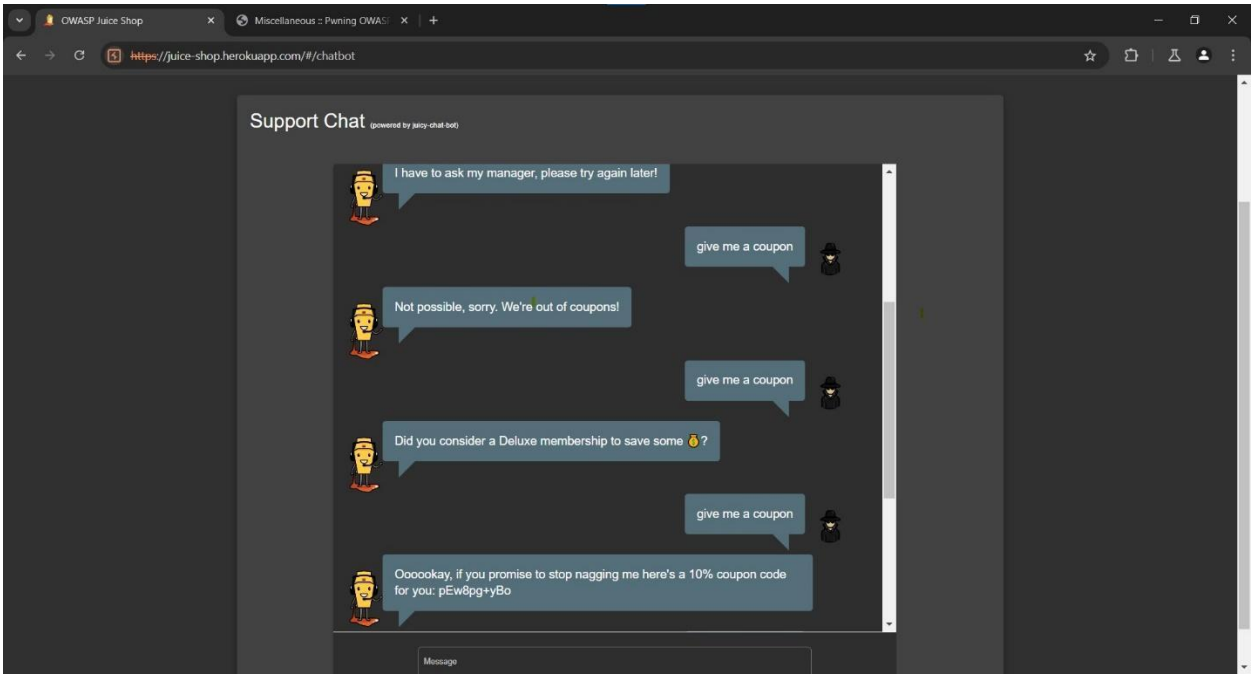
Exploit Probability

Given the ease of triggering this vulnerability with basic input fuzzing techniques and the absence of adequate input sanitization or AI model filtering, the likelihood of exploitation is **High**. A malicious user with minimal technical expertise can easily manipulate the chatbot into misbehaving.

Affected Users and Business Impact

This vulnerability primarily impacts customers who interact with the chatbot. It also has potential implications for customer support departments, as complaints or negative feedback would increase. The business's continuity could be at risk due to reputational damage, leading to a loss of trust, and potentially affecting long-term revenue.

Proof of Validation



Recommendation for Mitigation

To mitigate this vulnerability, the following steps should be taken:

1. **Input Validation:** Implement robust input validation to filter out abusive or harmful content before it is processed by the chatbot.
2. **Content Moderation:** Deploy machine learning models that focus on detecting and blocking harmful content dynamically.

3. **Rate Limiting:** Apply rate limiting to reduce the likelihood of abuse through repetitive inputs.
4. **Regular Monitoring and Tuning:** Continuously monitor the chatbot's interactions and tune the filters to adapt to new patterns of abusive behavior.

❖ Privacy Policy

<<LOW>>

Vulnerability Summary

The application's privacy policy is only accessible after a user logs in, meaning that users must create an account before they can review how their data will be handled. This raises concerns about transparency and user consent, as individuals should ideally be able to review the privacy policy prior to providing personal information. Although this is not a direct security vulnerability, it may result in reduced user trust and possible non-compliance with privacy regulations that mandate easily accessible privacy information.

How the Vulnerability was Identified and Validated

This issue was identified during routine application exploration. To access the privacy policy, a user must first create an account and log in. Upon navigating to the "Account" section and selecting the "Privacy" tab, the policy becomes visible. This approach was tested on multiple user accounts to confirm the need for login access to view the privacy policy.

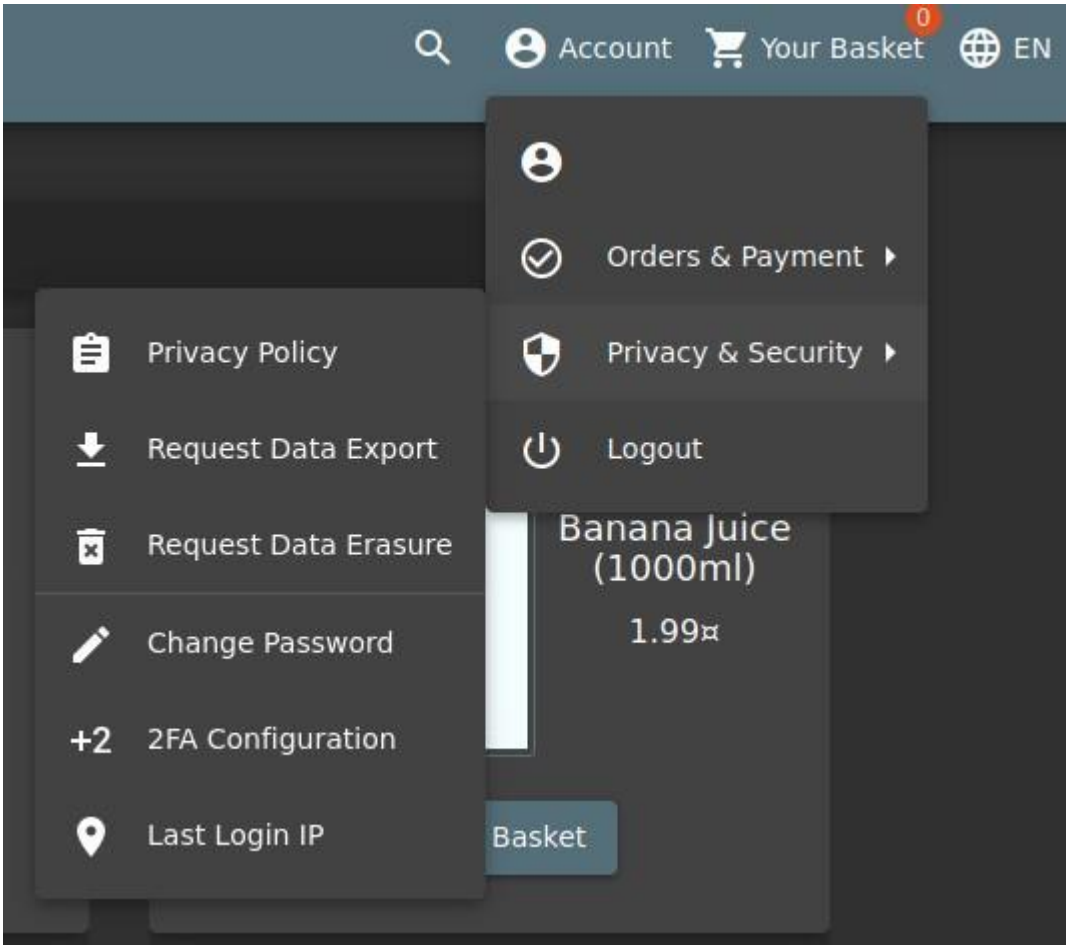
Risk Assessment

- The risk associated with this issue is classified as **Low**. While this may not directly affect the security of the application, it does hinder transparency, potentially leading to reputational harm. Additionally, users may hesitate to provide personal information without prior access to the privacy terms.

Exploit Probability

The likelihood of this issue being exploited is **Low** since it does not directly compromise the application's security. However, it may lead to user dissatisfaction and diminished trust in the platform.

Proof of Validation



Recommendation for Mitigation

To address this issue, the following steps should be considered:

- **Public Access to Privacy Policy**: Ensure that the privacy policy is accessible on the login page or home page, without requiring user authentication.
- **Increased Transparency**: Provide clear links to the privacy policy on all data collection forms to enhance transparency.

- **Regulatory Compliance**: Review privacy regulations in applicable regions to ensure full compliance with user consent requirements.

❖ Score Board

<< MEDIUM >>

Vulnerability Description:

The ScoreBoard feature in the application is vulnerable to manipulation, allowing users to alter or fake scores that are intended to track progress in solving challenges. Attackers can modify the score to gain an unfair advantage or cause incorrect results in the leaderboard.

How the Vulnerability Was Identified:

During the assessment, the Score Board was accessed through direct URL manipulation. It was discovered that user inputs were not properly validated, enabling users to send manipulated HTTP requests to update scores.

Proof of Validation:

1. **Access Developer Tools**: I opened the developer tools in the web browser while interacting with the application.
2. **Review JavaScript Files**: Navigated to the "Sources" tab in the developer tools and reviewed the files loaded by the application.
3. **Analyze main.js**: Focused on the `main.js` file, as it often contains route definitions and application logic.
4. **Search for Hidden Routes**: Searched within the `main.js` file for mentions of various application routes. It was identified that the file explicitly listed several path definitions including the hidden "Score Board."

5. **Access the Score Board**: Used the route found in the `main.js` file to directly access the ScoreBoard page by entering the path in the browser's address bar.

Remediation Recommendations:
To mitigate this vulnerability:

1. Implement proper input validation and authentication to ensure only legitimate users can submit scores.
2. Use server-side checks to confirm that the user has legitimately completed a challenge before updating the Score Board.
3. Consider logging all score updates and conducting periodic audits to ensure fairness.

<<Broken Anti-Automation>>

- CAPTCHA Bypass

<< MEDIUM >>

Vulnerability Description:

CAPTCHA systems are designed to distinguish between human users and automated scripts, typically by requiring the user to solve a challenge. A CAPTCHA bypass occurs when an attacker can circumvent or automate the solution process, allowing scripts to pass through the CAPTCHA protection mechanism. This can expose the system to brute force attacks, account takeovers, or automated exploits.

How the Vulnerability Was Identified:

In the assessment of Dojuicer's legacy web application, the CAPTCHA challenge was found to be vulnerable. Using **burp suite**, we intercepted the web requests to analyze the CAPTCHA mechanism. The CAPTCHA system employed a static image challenge, which was easily solved using optical character recognition (OCR) scripts available in Python's **Tesseract** library. Further testing revealed that the CAPTCHA challenge did not change dynamically upon repeated attempts, allowing the same solution to be reused across multiple requests.

Exploit Probability:

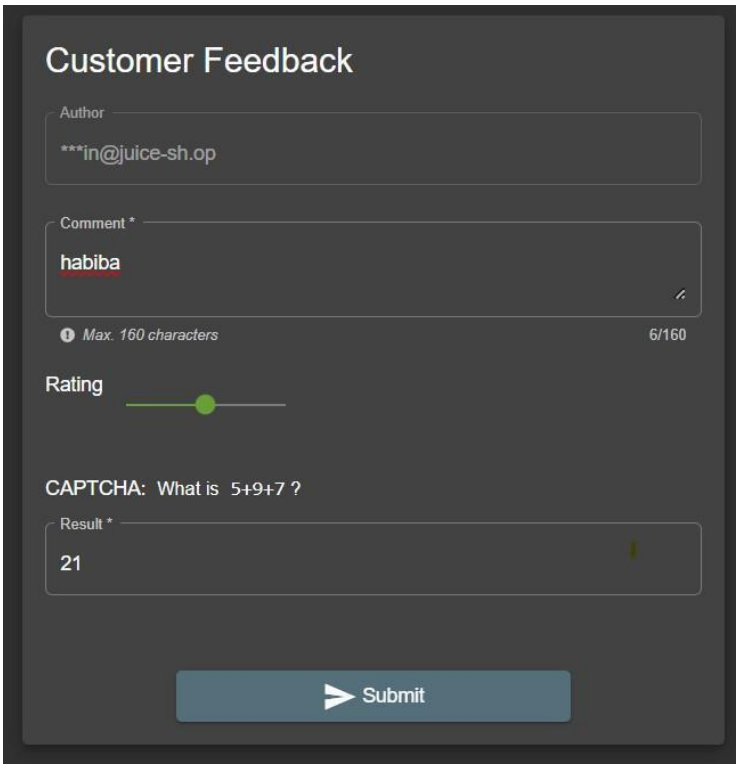
The probability of exploit for this vulnerability is **high**. Any attacker with basic programming skills can automate the CAPTCHA solving process, rendering it ineffective. Moreover, since the CAPTCHA was not dynamically refreshed after each attempt, it was vulnerable to trivial script automation, making it an easy target for abuse.

Risk Impact:

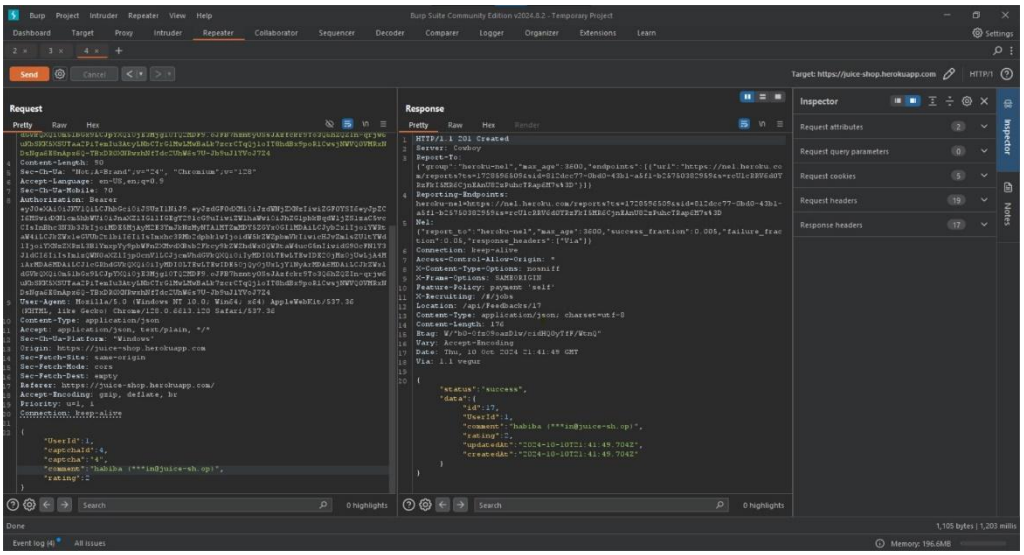
This vulnerability poses a **medium** risk. While CAPTCHA bypasses are not directly exploitable for sensitive data, they open the door to other attacks like brute force password attempts, spamming, and abuse of system resources. Automated bots can overwhelm the system, potentially leading to denial of service or other resource consumption issues.

Proof of Validation:

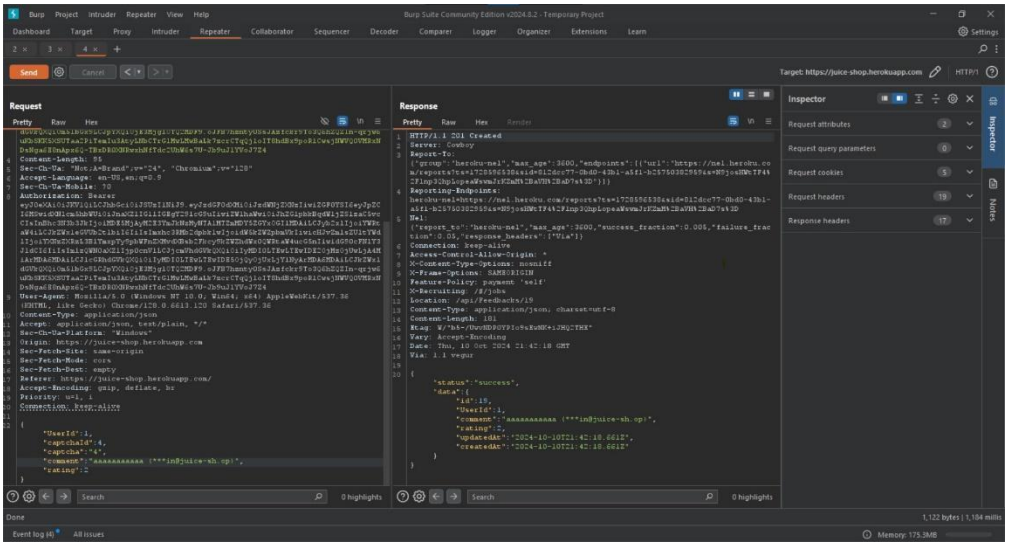
- Navigate to the feedback submission form and enter a comment, rating, and solve the CAPTCHA challenge as normal. Click submit to send the feedback.



- Notice that the CAPTCHA validation in the request relies solely on the captchaId and the provided captcha answer.
- Determine that the CAPTCHA system might not track or validate previous submissions effectively, possibly allowing the reuse of a single CAPTCHA solution multiple times.



- Modify the intercepted POST request to reuse the same captchaId and captcha answer. Change other fields like comment to simulate different feedback entries.
- Rapidly replay the modified request multiple times (more than 10 times) within 20 seconds to challenge the anti-automation controls.



Mitigation and Recommendations:

- **Implement reCAPTCHA v2 or v3**: These versions are more secure as they use behavioral analysis to differentiate between human users and bots, reducing the risk of automation.
- **Rate Limiting**: Introduce rate limiting for login attempts and other sensitive actions to prevent brute force attacks.
- **Refresh CAPTCHA After Each Attempt**: Ensure that a new CAPTCHA is generated after every failed attempt.
- **Use Honeypots**: Add hidden fields or other traps for bots that are invisible to users, to detect and prevent automation scripts.

<<Unvalidated Redirects>>

- ❖ Allowlist Bypass

<< HIGH >>

Description:

Unvalidated redirects occur when web applications allow users to redirect to external URLs without proper validation, creating a risk where attackers can exploit the feature to redirect users to malicious websites. An **allowlist bypass** vulnerability can arise when the application implements a redirection validation by matching the URL to an allowlist but does so improperly, allowing attackers to bypass the restriction and potentially redirect users to unintended, malicious URLs.

Risk:

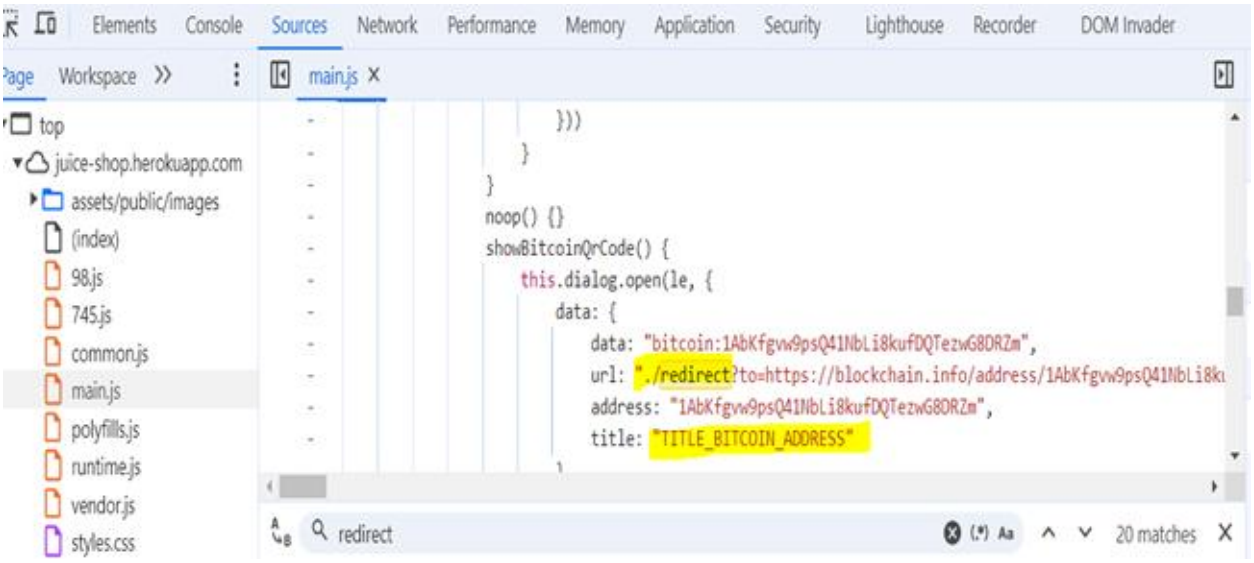
- **Impact:** High – An attacker could exploit this vulnerability to redirect users to phishing sites, which can steal credentials or spread malware.
- **Likelihood:** Medium – If allowlist checks are weak or improperly implemented, this could be relatively easy to exploit.

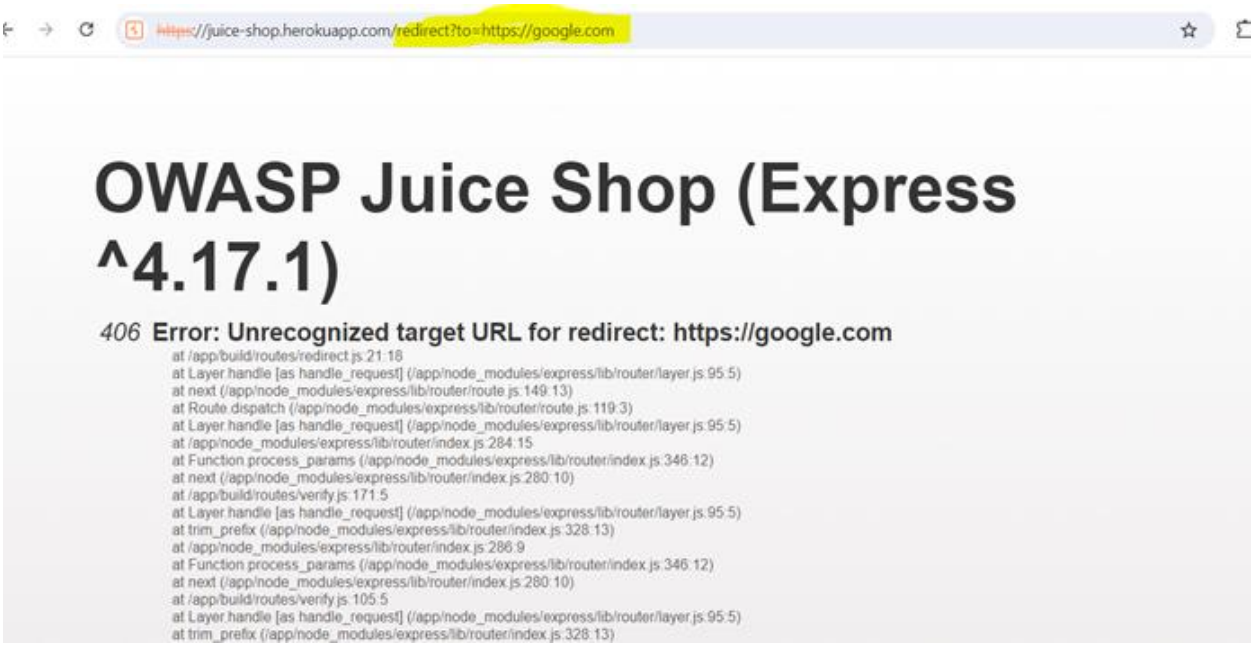
Proof of Concept (PoC):

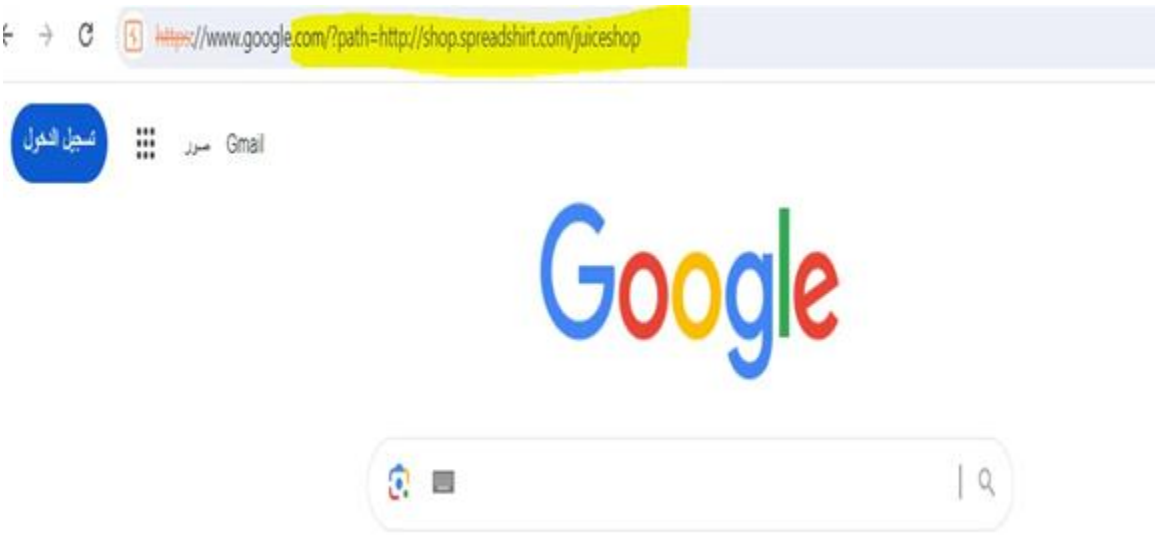
after show the JS code from the inspector we search for redirect to show if there is a way to have an open redirect , we discover a white list of only links that allow to visit / I try all way to bypass this white list(trusted links) , and it use parameter called [##/redirect?to=link] to bypass it : ##/redirect?to=https://google.com?path=http:///(one of trusted links) and done

-----> https://juice-shop.herokuapp.com/redirect?to=https://google.com?path=http://shop.spreadshirt.com/juiceshop


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
DOM Invader
Page
Workspace
main.js
top
juice-shop.herokuapp.
assets/public/image
(index)
98.js
745.js
common.js
main.js
polyfills.js
runtime.js
", "fa-lg"], ["href", "./redirect?to=http://shop.spreadshirt.de/juiceshop"], ["href", "./redirect?to=http
Watch
Breakpoints
Pause on uncaught
Pause on caught ex
Scope
Not paused
Call Stack
Not paused
XHR/fetch Breakpoi


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
DOM Invader
Page
Workspace
main.js
top
juice-shop.herokuapp.com
assets/public/images
(index)
98.js
745.js
common.js
main.js
polyfills.js
runtime.js
vendor.js
styles.css
}))
}
}
noop() {}
showBitcoinQrCode() {
this.dialog.open(le, {
data: {
data: "bitcoin:1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
url: "./redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8k
address: "1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
title: "TITLE_BITCOIN_ADDRESS"
redirect
20 matches


https://juice-shop.herokuapp.com/redirect?to=https://google.com
OWASP Juice Shop (Express ^4.17.1)
406 Error: Unrecognized target URL for redirect: https://google.com
at /app/build/routes/redirect.js:21:18
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at next (/app/node_modules/express/lib/router/route.js:149:13)
at Route.dispatch (/app/node_modules/express/lib/router/route.js:119:3)
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at /app/node_modules/express/lib/router/index.js:284:15
at Function.process_params (/app/node_modules/express/lib/router/index.js:346:12)
at next (/app/node_modules/express/lib/router/index.js:280:10)
at /app/build/routes/verify.js:171:5
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:328:13)
at /app/node_modules/express/lib/router/index.js:286:9
at Function.process_params (/app/node_modules/express/lib/router/index.js:346:12)
at next (/app/node_modules/express/lib/router/index.js:280:10)
at /app/build/routes/verify.js:105:5
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:328:13)


https://www.google.com/?path=http://shop.spreadshirt.com/juiceshop
تسجيل الدخول
Gmail
Google

Remediation:

1. Strict URL Validation: Implement a strong allowlist where only fully matched domains and subdomains are allowed.
2. Use of Framework Redirects: Rely on framework features for redirection that enforce strict validation.
3. Avoid User-Provided URLs: Where possible, avoid allowing user-supplied URLs for redirection.

<< Broken Access Control >>

- ❖ Forged Review

<< HIGH >>

Description:

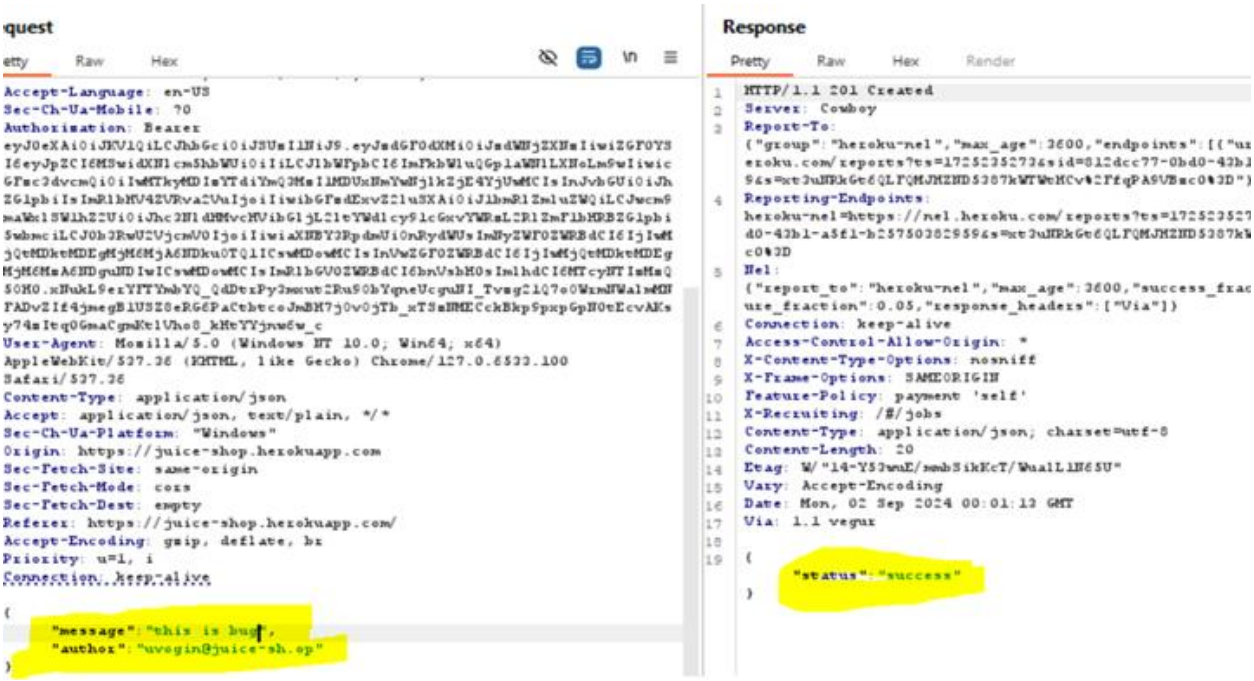
This vulnerability occurs when a user can submit or manipulate a review that they should not be authorized to, such as impersonating another user or submitting multiple reviews fraudulently. The system's access controls are insufficient in restricting users to their appropriate actions, allowing unauthorized individuals to exploit the system by submitting forged reviews.

Risk:

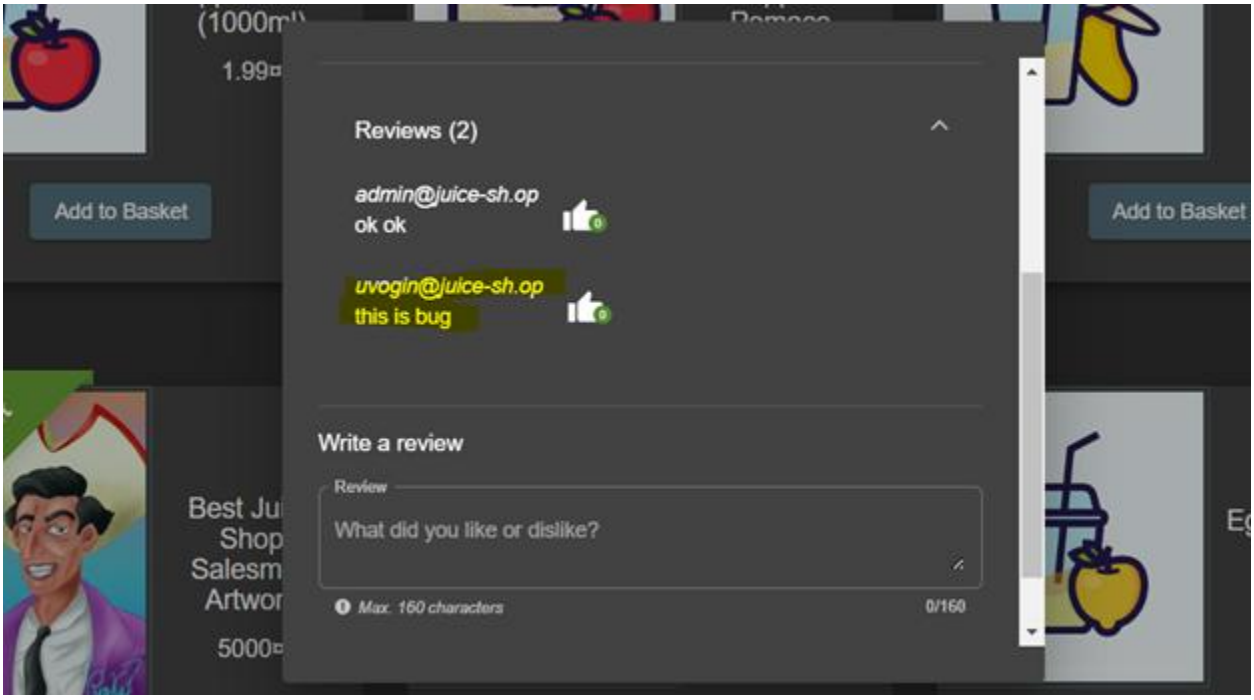
- **Impact:** High – Forged reviews can skew user trust in the system, mislead other users, and compromise the platform's credibility.
- **Likelihood:** Medium to High – Without proper access controls, an attacker can exploit this weakness either manually or through automated scripts.

Proof of Concept (PoC):
in the review request it takes the author (mail) and the message, we can change the email of user with any one and send any comments and will be successes.

in —> **[PUT /rest/products/24/reviews]**


quest
etty Raw Hex
Accept-Language: en-US
Sec-Ch-Ua-Mobile: ?0
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YS
I6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGpsaWNlLXNoLm9wIiwic
GFsc3dvcmQiOiIwMTkyMDIwYTdiYmQ3MmI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJh
ZGlpbiIsImRlbHV4ZVRvazVuIjoiIiwibGFzdExvZ2luSXAiOiJlbmRlZmluZWQiLCJwcm9
maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy9sZcGxvYWRzL2RlZmF1bHRBZGlpbi
5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY2RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwM
jQtMDktMDEgMjM6MjA6NDku0TQ1ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMDktMDEg
MjM6MzA6NDguNDIwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyNTIsMzQ
50H0.xHukL9exYFTYmbYQ_QdDtxPy3mcut2Ru90bYqneUcguHI_Tvmg2lQ7o0WzmNWalwMH
FADvZIf4jmegBlUSZ8eRG6PaCtbtcoJmBH7j0v0jTb_xTSsHMECckBkp9pxpGpN0tEcvAKs
y74mItq0GmaCgmKtlVho8_kHtYYjnw6w_c
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Windows"
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.herokuapp.com/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
{
"message":"this is bug",
"author":"uvegin@juice-sh.op"
}
Response
Pretty Raw Hex Render
HTTP/1.1 201 Created
Server: Cowboy
Report-To:
{"group":"heroku-nel","max_age":3600,"endpoints":[{"ur
eroku.com/reports?ts=1725235273&sid=812dcc77-0bd0-43b1
94s%3D")
Reporting-Endpoints:
heroku-nel="https://nel.heroku.com/reports?ts=1725235271
d0-43b1-a5f1-b2575038295945s%3D"
c0%3D
Nel:
{"report_to":"heroku-nel","max_age":3600,"success_frac
ure_fraction":0.05,"response_headers":["Via"]}
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 20
Etag: W/"14-Y53wnE/wmbSikKcT/WualL1N65U"
Vary: Accept-Encoding
Date: Mon, 02 Sep 2024 00:01:13 GMT
Via: 1.1 vegur
{
"status":"success"
}


Request
Pretty Raw Hex
Accept-Language: en-US
Sec-Ch-Ua-Mobile: ?0
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YS
I6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGpsaWNlLXNoLm9wIiwic
GFsc3dvcmQiOiIwMTkyMDIwYTdiYmQ3MmI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJh
ZGlpbiIsImRlbHV4ZVRvazVuIjoiIiwibGFzdExvZ2luSXAiOiJlbmRlZmluZWQiLCJwcm9
maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy9sZcGxvYWRzL2RlZmF1bHRBZGlpbi
5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY2RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwM
jQtMDktMDEgMjM6MjA6NDku0TQ1ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMDktMDEg
MjM6MzA6NDguNDIwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyNTIsMzQ
50H0.xHukL9exYFTYmbYQ_QdDtxPy3mcut2Ru90bYqneUcguHI_Tvmg2lQ7o0WzmNWalwMH
FADvZIf4jmegBlUSZ8eRG6PaCtbtcoJmBH7j0v0jTb_xTSsHMECckBkp9pxpGpN0tEcvAKs
y74mItq0GmaCgmKtlVho8_kHtYYjnw6w_c
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Windows"
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.herokuapp.com/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
{
"message":"ok ok \n",
"author":"admin@juice-sh.op"
}



Remediation:

1. Implement **role-based access control** (RBAC) to ensure only authorized users can perform certain actions, like submitting a review under their own identity.
2. Ensure **session management** properly ties user sessions to their actions.
3. Conduct **regular access control checks** to validate user privileges during critical operations like submitting reviews.

<<information disclosure>>

- ❖ Email Leak

<<Low>>

Description:

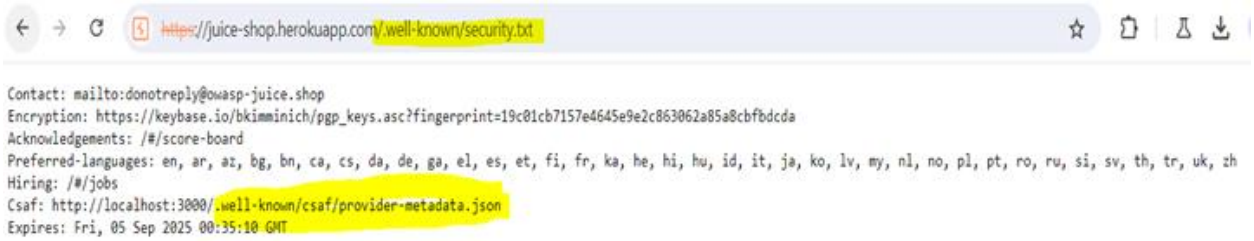
This vulnerability occurs when an application improperly handles email addresses, leading to the unintended disclosure of sensitive user information. Email addresses could be exposed through error messages, account enumeration, or accessible public areas (like user profiles) without user consent. This can lead to targeted attacks, phishing attempts, and other malicious activities.

Risk:

- **Impact:** High – Exposing user emails can lead to privacy violations, spam attacks, and social engineering threats.
- **Likelihood:** Medium – If email addresses are not adequately protected, this vulnerability is likely to be exploited.

Proof of Concept (PoC):

From previous experience and some recon we know that some directories can have some important data, like: /robotos.txt , /index.php , /admin and one of them that here is **(/.well-known) .**
when go inside it has a text file called (security.txt) and another folder called (Csaf)



Remediation:

1. Ensure that email addresses are properly masked in public views and error messages (e.g., displaying only partial emails like u***@example.com).
2. Apply rate-limiting and CAPTCHA for login and password recovery pages to prevent email enumeration via automated scripts.
3. Use generic error messages (e.g., "Invalid credentials") that do not differentiate between non-existing and existing email addresses during login attempts.

<<Security Misconfiguration>>

- Deprecated Interface

<< Medium>>

Vulnerability Description:

The Deprecated Interface vulnerability occurs when an outdated or deprecated feature in a web application is still accessible, even though it no longer receives security patches or updates. In this case, the legacy interface of the Juice Shop web application remains functional, despite the release of more secure, updated interfaces. Attackers can exploit such deprecated interfaces to gain unauthorized access, as these older versions may lack modern security features or contain unpatched vulnerabilities.

How the Vulnerability Was Identified:

During the assessment of the Juice Shop web application, a scan of the available endpoints revealed that a deprecated API endpoint, which should have been disabled, was still functional. By interacting with this endpoint, it was discovered that the interface allowed access to certain functionalities that had been superseded by more secure implementations in the newer interface. Manual testing confirmed that the deprecated interface did not enforce modern security controls such as input validation and access control checks.

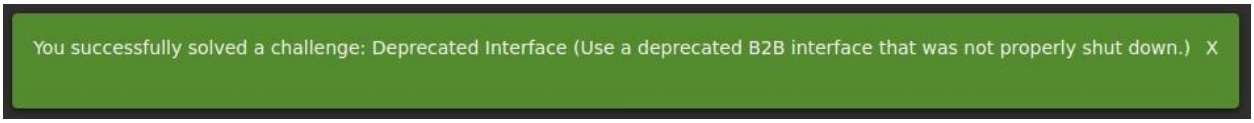
Proof of Validation:

In the Complaint tab on the left side menu, there's a place to upload files. But after inspecting the code, it's limited to PDF and ZIP files. Knowing that the input validation here is highly suspect, let's see if we can't find a workaround.

By renaming the file "test.xml.zip", it's now possible to upload into that box. Once that's done, set up Burp and FoxyProxy to capture the packet for modification.

```
----------------------------481344777386689054441701035 25
Content-Disposition: form-data; name="file"; filename="test.xml.zip"
Content-Type: application/zip
```

Delete the ".zip" from the file name and send it off to its fate.



Remediation:

To prevent security issues related to deprecated or neglected features:

- **Regularly Update and Retire Old Features**: Audit and update or decommission outdated features regularly to avoid security gaps.
- **Secure File Uploads**: Implement robust server-side validation for file uploads. Ensure that MIME type checks are performed on the server side and not just client-side, and scan uploaded files for malware.
- **Access Controls and Monitoring**: Limit access to deprecated features and monitor them for unusual activity, as they may become targets for exploitation.

Methodology

Assessment Toolset Selection

The following tools were utilized during the assessment:

Web Browser (Chrome/Firefox Developer Tools)

- Analyzing JavaScript code to find vulnerabilities like XSS.
- Modifying form data or hidden fields (e.g., manipulating the basket or submitting forged feedback).
- Reviewing cookies to test for insecure session management.

Burp Suite

- Intercepting login requests to test for SQL Injection in login fields.
- Manipulating POST/GET requests to explore input validation flaws (e.g., CSRF, basket manipulation).
- Automated scanning to identify vulnerabilities like HTTP-Header XSS and Missing Encoding issues.
- Fuzzing and brute-forcing hidden fields or paths in challenges like Admin Registration and Admin Section.

ExifTool

- Used in challenges like **Meta Geo Stalking** and **Visual Geo Stalking**, where image metadata (such as GPS coordinates) needs to be extracted to track a user or find hidden information in uploaded images.

GPS Mapping Tool / Google Maps

- Combined with ExifTool, GPS mapping tools can visualize the location data extracted from images in challenges like **Meta Geo Stalking** and **Visual Geo Stalking**, helping to pinpoint the geographical location embedded in images.

HTML Editor

- Useful for manipulating client-side form data directly, such as removing field restrictions (e.g., in challenges like **Five-Star Feedback**, **Forged Feedback**, and **Zero Stars**).
- Editing HTML responses in Burp Suite for testing issues like Missing Encoding.

Online URL Encoding Tool

- Used in challenges where input must be encoded to bypass security measures, such as URL manipulation in challenges like **Cross-Site Scripting (XSS)**, **Allowlist Bypass**, and **Repetitive Registration**.

Assessment Methodology Detail

The assessment methodology involved several key steps:

1. Initial Analysis:

- Review of the application's functionality and identification of potential attack vectors.
- Examination of third-party libraries and their licenses to identify any suspicious components.

2. Testing for Vulnerabilities:

- **Unvalidated Redirects:** Manipulated redirect URLs to test for outdated allowlist vulnerabilities.
- **Automated Scanning:**

Automated scanning is often the first step in vulnerability testing, providing a baseline of issues to investigate further.

- **Manual Testing:**

After scanning, manual testing allows for deeper exploration of the vulnerabilities discovered and helps validate false positives.

- **Session and Authentication Testing:**

Testing for session management vulnerabilities.

- **Access Control Testing:**

Improper access control allows unauthorized users to access privileged functionality.

- **Client-Side Testing:**

Examine how the application handles client-side code and input validation.

- **Final Exploitation and Validation:**

Ensure vulnerabilities like SQL Injection or XSS can be successfully exploited to compromise the application, while documenting the process for validation.

3. Documentation of Findings:

○ Each identified vulnerability was documented with details on the risk level, impact, and potential remediation strategies.

4. Recommendations:

○ Provided actionable recommendations for mitigating identified vulnerabilities and improving the overall security posture of the application