**Task 1**

**Conduct security testing on a sample web application to identify**

**vulnerabilities like SQL injection, XSS, and authentication flaws.**

**OWASP JUICE SHOP VULNERABILITY REPORT**

Vulnerable URL:<https://juice-shop.herokuapp.com/#/>

**Title:** SQL injection

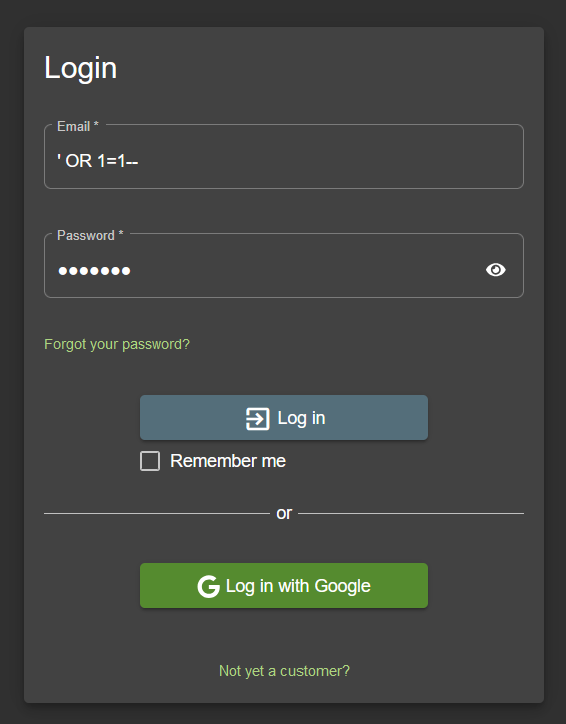
**Severity:**

**Description:** SQL Injection is a critical security vulnerability that occurs when an attacker is able to insert or manipulate SQL queries executed by an application. This happens when user input is improperly sanitized or validated, allowing malicious SQL statements to be included in the query. In this specific case, the vulnerability was identified when [specific context, e.g., an input field in the login form was found to allow the insertion of SQL code, enabling the attacker to manipulate the SQL queries sent to the database]. This can lead to unauthorized access to the database and potentially expose or manipulate sensitive data.

**Steps to Reproduce:**

Step 1- Visit the login page

Step 2- Enter sql query ’ OR 1=1-- to the username and any text as password

****

**Impact:** The impact of unauthorized access can be severe, depending on the sensitivity of the data or systems accessed. Potential impacts include exposure of sensitive data such as personal information, financial records, or proprietary information. Unauthorized actions performed on behalf of legitimate users can lead to further exploitation, such as privilege escalation, data modification, or service disruption. Additionally, there may be legal and regulatory consequences due to data breaches or non-compliance with data protection regulations.

**Mitigation/Recommendation:** To mitigate the risk of unauthorized access, it is crucial to strengthen authentication mechanisms by implementing multi-factor authentication (MFA) to add an extra layer of security, enforce strong password policies, and utilize secure authentication protocols such as OAuth, SAML, or OpenID Connect. Regularly reviewing and updating access controls to ensure the principle of least privilege and implementing role-based access control (RBAC) will help restrict access based on user roles. Ensuring continuous monitoring and auditing of access logs can detect and respond to unauthorized access attempts promptly, further safeguarding the system.

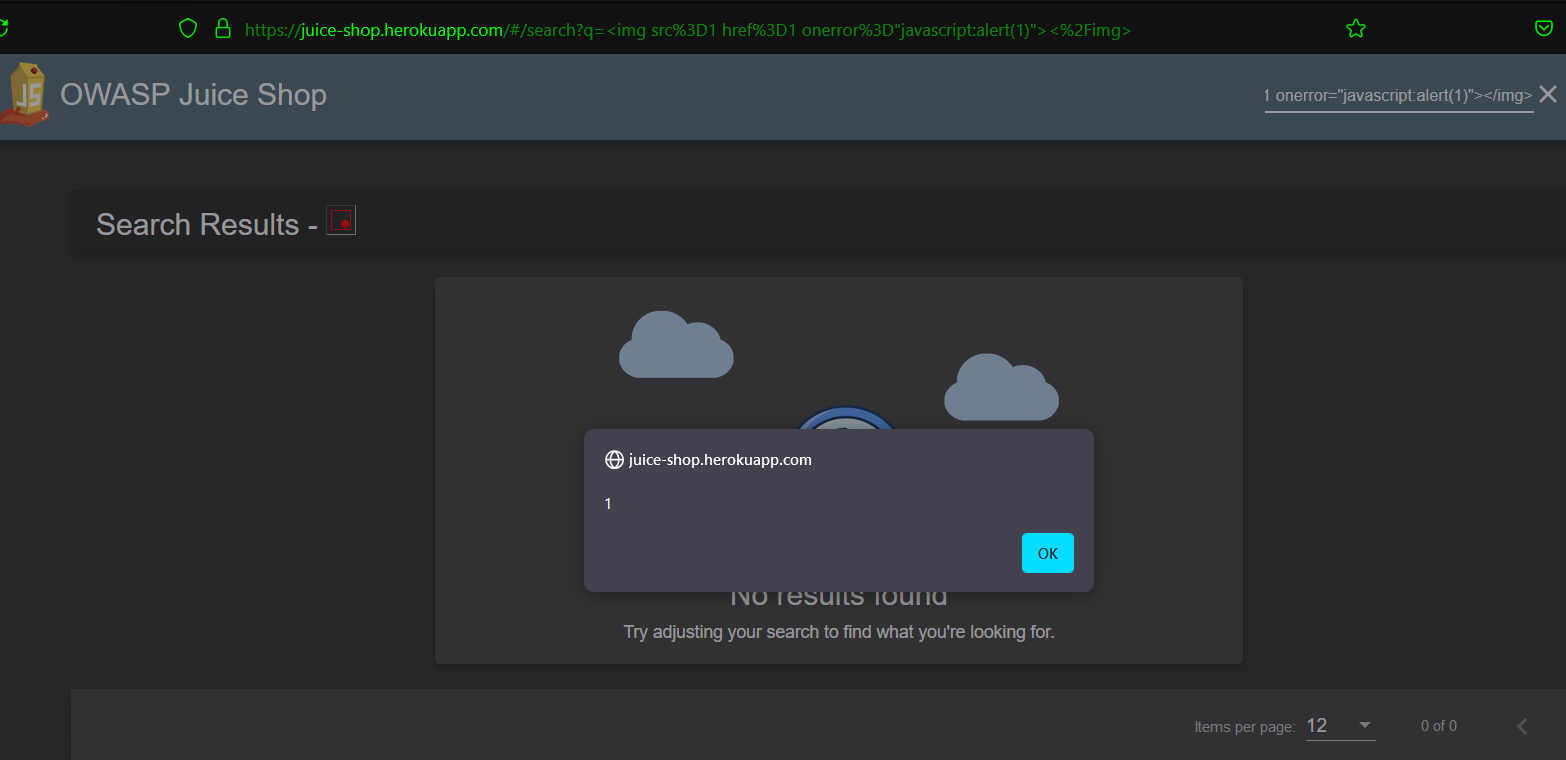
**Title:** Cross-Site Scripting (XSS)

**Description:** Stored Cross-Site Scripting (XSS) is a type of vulnerability where malicious scripts are injected and stored on a target server, typically within a database, comment field, message forum, or other data storage locations. Unlike reflected XSS, where the malicious script is reflected off a web server, stored XSS is more persistent, affecting every user who views the infected data. In this specific instance, the vulnerability was identified when [specific context, e.g., user input in the comment section was not properly sanitized, allowing JavaScript code to be stored and executed in subsequent page loads.

**Steps to Reproduce:**

Step 1: Visit the home page of the website

Step 2: Enter <img src=1 href=1 onerror="javascript:alert(1)"></img> on the search bar

****

**Impact:** The impact of stored XSS can be significant and far-reaching. When users access the infected page, the malicious script executes in their browsers, allowing attackers to steal session cookies, redirect users to malicious sites, log keystrokes, or perform actions on behalf of the user without their knowledge. This can lead to unauthorized access to user accounts, theft of sensitive information, and potential further exploitation within the application. The persistence of stored XSS makes it particularly dangerous, as the malicious script continues to pose a risk until it is identified and removed.

**Mitigation/Recommendation:** To mitigate the risk of stored XSS, it is crucial to implement robust input validation and output encoding mechanisms. All user input should be sanitized and validated to ensure that no executable scripts or malicious code can be injected. Utilizing libraries or frameworks that automatically escape output in HTML contexts can help prevent scripts from executing. Employing Content Security Policy (CSP) headers can further restrict the sources from which scripts can be loaded and executed. Regular security audits and code reviews should be conducted to identify and remediate potential XSS vulnerabilities. Additionally, educating developers on secure coding practices and the importance of validating and encoding user input can help reduce the risk of such vulnerabilities in the future.

**Title:** Html injection

**Severity:**

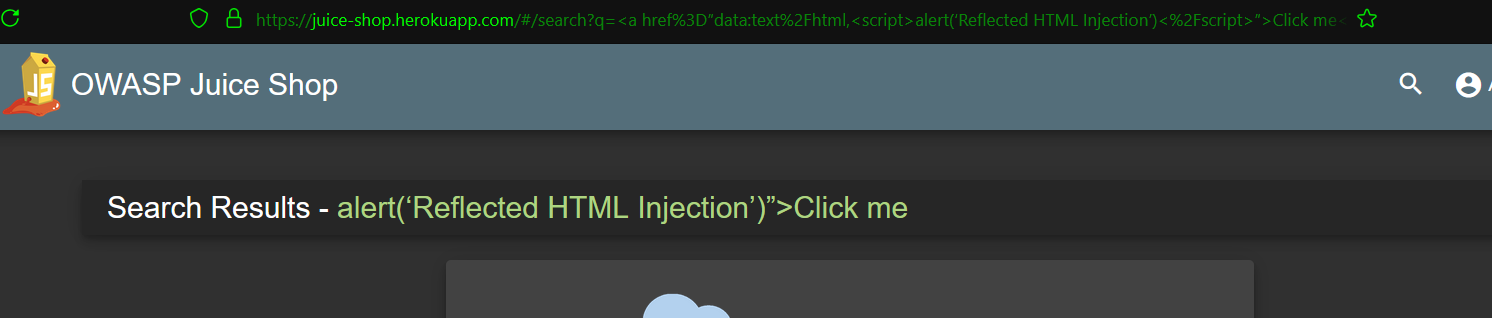
**Description:** HTML Injection is a type of security vulnerability that occurs when an application allows user-supplied data to be injected into the HTML response without proper validation or encoding. This can enable an attacker to inject arbitrary HTML code, which can manipulate the structure and content of the webpage. In this specific instance, the vulnerability was identified when [specific context, e.g., user input in the feedback form was directly included in the HTML response, allowing the injection of HTML tags and content.

**Steps to Reproduce:**

Step 1: Visit the home page of the website

Step 2: Enter <a href=”data:text/html,<script>alert(‘Reflected HTML Injection’)</script>”>Click me</a> on the search bar

<a href=”data:text/html,<script>alert(‘Reflected HTML Injection’)</script>”>Click me</a>



**Impact:** The impact of HTML Injection can vary depending on the context and the application's use of the injected HTML. Potential impacts include the alteration of webpage content, defacement of the website, and the introduction of malicious links or scripts. While HTML Injection itself may not directly lead to code execution, it can facilitate other attacks such as Cross-Site Scripting (XSS) if combined with script injection. This can result in the theft of sensitive information, unauthorized actions on behalf of users, and a general degradation of user trust and application integrity.

**Mitigation/Recommendation:** To mitigate the risk of HTML Injection, it is essential to implement thorough input validation and output encoding. All user-supplied data should be validated on the server side to ensure it conforms to expected formats and does not contain any harmful HTML tags or attributes. Additionally, output encoding should be used to ensure that any user input included in the HTML response is treated as text rather than executable code. Utilizing security libraries or frameworks that handle encoding and escaping can significantly reduce the risk. Regular security reviews and testing should be conducted to identify and address potential HTML Injection vulnerabilities. Educating developers on secure coding practices and the importance of validating and encoding user inputs is also crucial in preventing such vulnerabilities.

**Title:** Iframe Injection

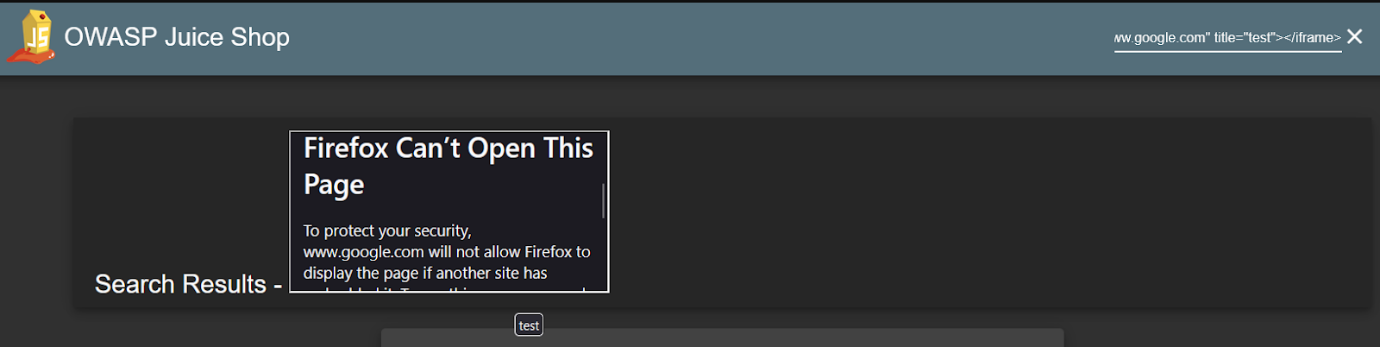
**Severity:**

**Description:** Iframe Injection occurs when an attacker is able to insert an <iframe> element into a webpage through unvalidated or unsanitized user input. This type of vulnerability typically arises when user inputs, such as those from form fields or URL parameters, are incorporated into the HTML of a page without proper filtering. In this particular case, the vulnerability was identified when [specific context, e.g., an input field that accepts user-generated content was found to allow the inclusion of iframe elements, which led to unintended embedding of external content within the target site.

**Steps to Reproduce:**

Step 1: Visit the home page of the website

Step 2: Enter <iframe src="https://www.google.com" title="test"></iframe> on the search bar

****

**Impact:** The impact of iframe injection can be quite severe. Attackers can use this vulnerability to perform various malicious activities, including creating phishing attacks by embedding deceptive content or login forms within the iframe, thereby tricking users into revealing sensitive information. It can also lead to content spoofing, where attackers manipulate the appearance of a legitimate site to mislead users. Additionally, iframe injection can be leveraged for clickjacking attacks, where users are deceived into performing actions they did not intend by interacting with invisible or disguised elements. Furthermore, if the iframe points to a malicious site, it may capture and exfiltrate sensitive data from users without their knowledge.

**Mitigation/Recommendation:** To mitigate the risk of iframe injection, it is crucial to implement comprehensive input validation and output encoding. User inputs that are reflected in the HTML output should be rigorously sanitized to ensure that any potentially harmful HTML tags, including <iframe>, are removed or escaped. Applying a Content Security Policy (CSP) can further enhance security by specifying which sources are allowed to frame content, thus preventing unauthorized iframes from being embedded. Regular security assessments, including vulnerability scanning and code reviews, should be conducted to identify and address such vulnerabilities. Additionally, educating developers about secure coding practices and maintaining up-to-date security patches will help in preventing iframe injection and similar vulnerabilities.

**Title:** Insecure Direct Object Reference (IDOR)

**Description:** Insecure Direct Object Reference (IDOR) occurs when an application exposes internal object references (e.g., file names, database keys) in such a way that unauthorized users can access or manipulate them. This vulnerability arises when user input is used to directly reference objects without proper authorization checks. For instance, if a URL or API endpoint includes a parameter that specifies an object ID, and the application does not validate whether the user has permission to access or modify the object with that ID, an attacker could exploit this by changing the ID to access other users' data. In this specific case, the vulnerability was identified when , an API endpoint that allowed users to access or modify data by changing a numeric ID without checking if the user had the appropriate permissions.

**Steps to Reproduce:**

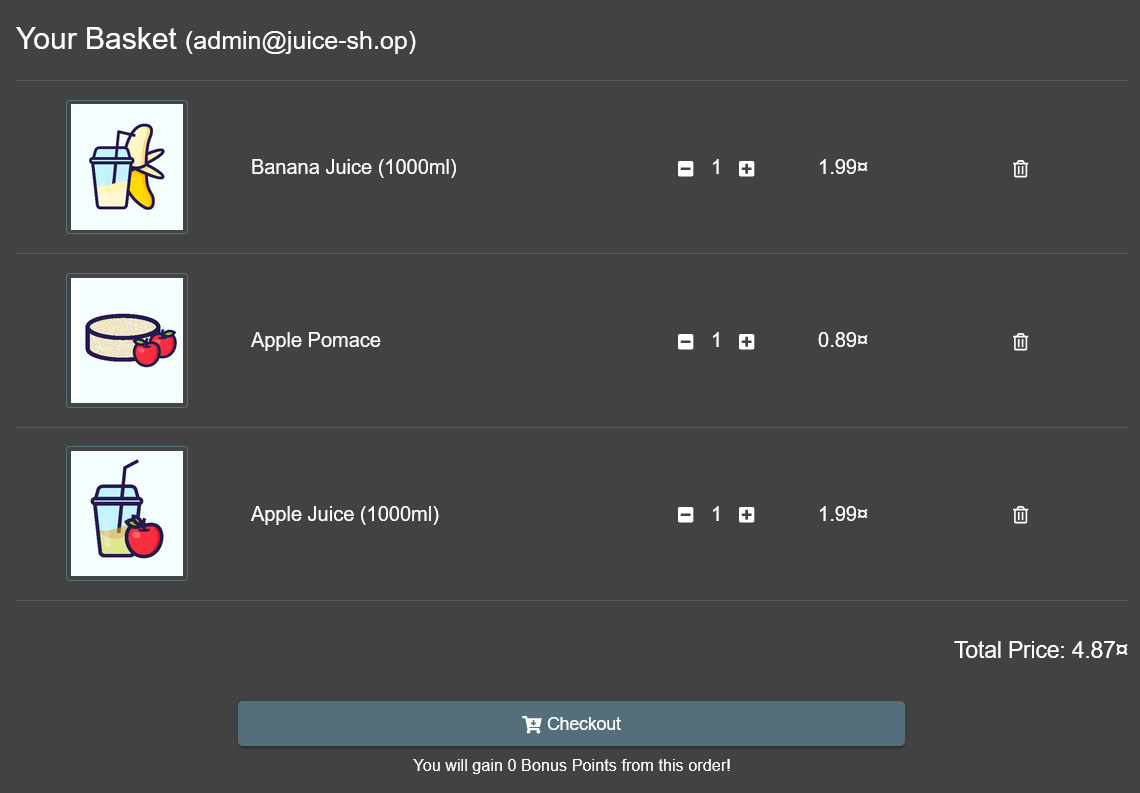
step1 : add Any items to the cart and checkout

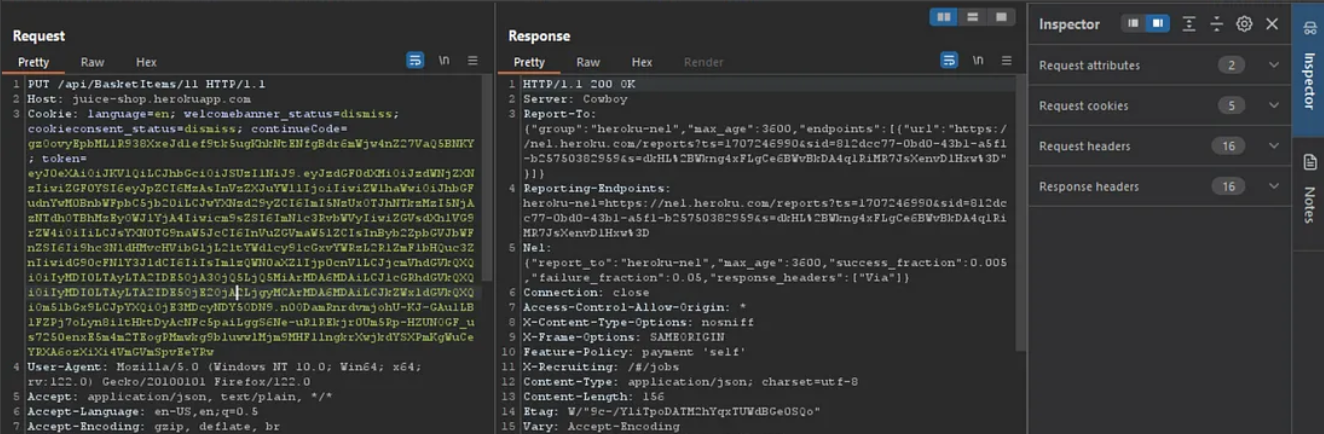
Step 2: Intercept the check out request with burp suite

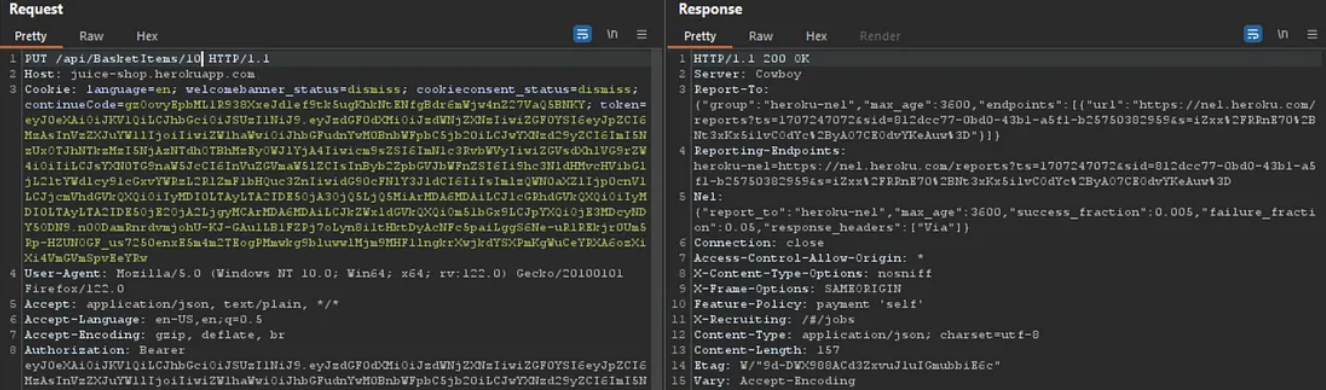
Step 3: Send the request to repeater

Step 4: on the very first line you can see BasketItem/11 change the /11 to /10 or whatever you like

Step 5: Send the edited request ahead and the response came to be 200 ok that means now my basket is someone else’s so confirming IDOR .

****

****

****

**Impact:** The impact of IDOR can be significant, as it can lead to unauthorized access to sensitive information, data leakage, or manipulation of resources. Attackers can exploit IDOR vulnerabilities to view or modify data that they should not have access to, such as personal information, financial records, or administrative data. This can result in privacy violations, data corruption, and unauthorized changes to critical data. Additionally, IDOR can facilitate further attacks by allowing unauthorized users to escalate privileges or gain access to restricted functionalities, ultimately compromising the integrity and security of the application.

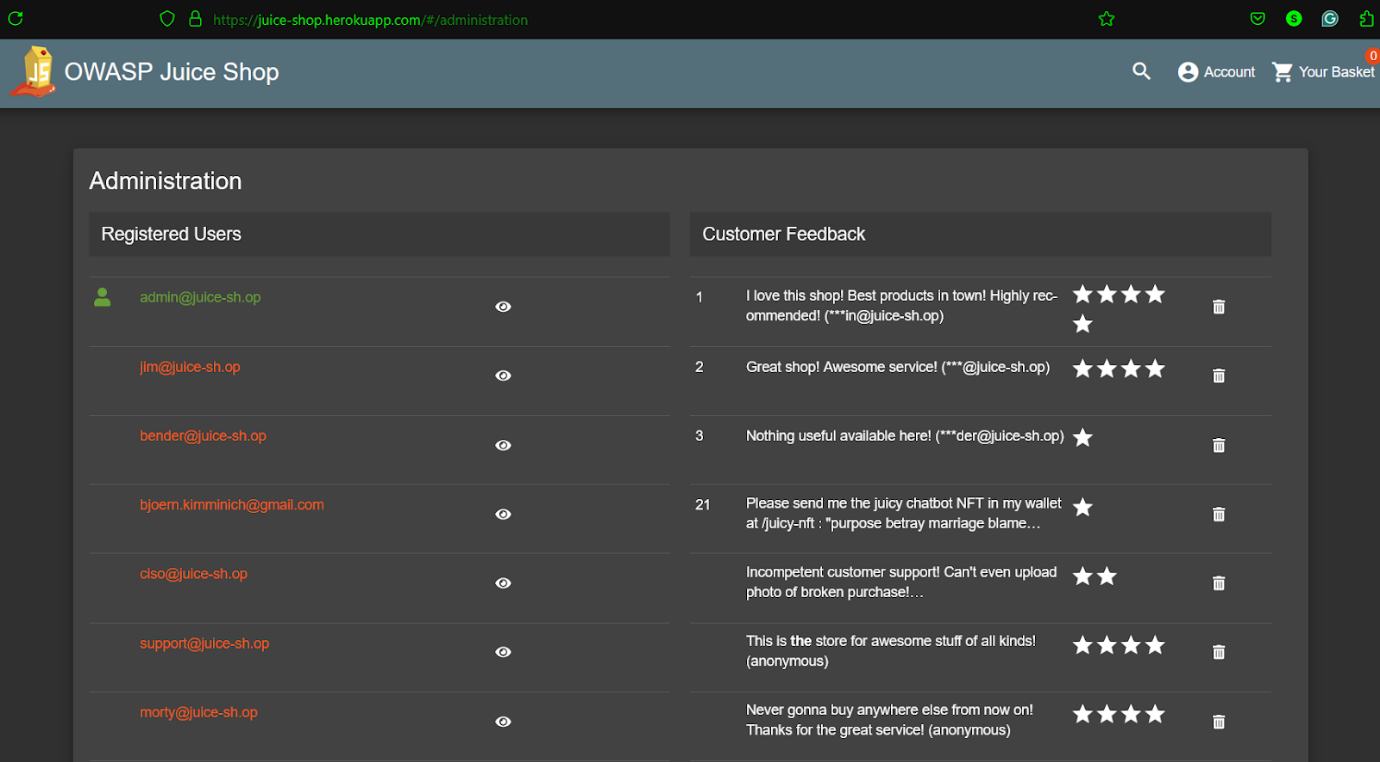
**Mitigation/Recommendation:** To mitigate the risk of IDOR, it is essential to implement robust access controls and authorization checks on all user inputs that reference internal objects. The application should ensure that users can only access or manipulate resources for which they have explicit permissions. This can be achieved by implementing access control mechanisms that validate user roles and permissions before processing requests. Additionally, avoiding the use of direct object references in URLs or API endpoints and using indirect references or unique

**Title:** Administration Page Disclosure

**Description:** Administration page disclosure occurs when sensitive administrative interfaces and pages are exposed to unauthorized users, either through direct URL access or insufficient access control mechanisms. This vulnerability typically arises due to a lack of proper authentication and authorization checks, making it possible for attackers to discover and access administrative pages. In this specific case, the vulnerability was identified when [specific context, e.g., an attacker was able to access the administration page by directly navigating to a specific URL, bypassing any login or authentication mechanism].

**Steps to Reproduce:**

**step1 :** Visit the Vulnerable website  
Step 2: Just add /administrator at the end of the URL

****

**Impact:** The impact of administration page disclosure can be significant, as it potentially grants attackers access to sensitive administrative functionalities. This can lead to unauthorized changes in application settings, user account manipulation, and access to sensitive data. If an attacker gains control of the administration interface, they can perform malicious actions such as creating or deleting user accounts, altering content, or changing security configurations. This can result in a complete compromise of the application, leading to data breaches, service disruptions, and loss of user trust.

**Mitigation/Recommendation:** To mitigate the risk of administration page disclosure, it is essential to implement robust authentication and authorization controls. Ensure that all administrative pages are protected by strong authentication mechanisms, such as multi-factor authentication (MFA), and that only authorized users have access to these pages. Implement role-based access control (RBAC) to restrict access based on user roles and permissions. Additionally, obscuring administrative URLs and implementing security mechanisms such as HTTP authentication and IP whitelisting can add an extra layer of protection. Regular security audits and penetration testing should be conducted to identify and address any weaknesses in access control mechanisms. Educating developers on secure coding practices and maintaining up-to-date security patches are also crucial for preventing unauthorized access to administrative pages.

**Title:** Unauthorized Password Change

**Severity:**

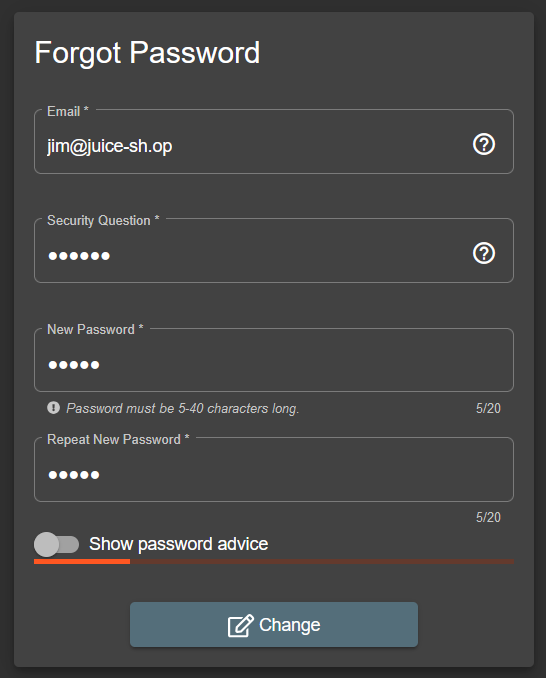
**Description:** The unauthorized password change vulnerability allows attackers to change the password of another user without proper authorization. This vulnerability typically arises when there is a lack of proper authentication and authorization checks in the password change functionality of an application. In this specific case, the vulnerability was identified when [specific context, e.g., an attacker was able to change another user's password by manipulating parameters in a password reset request without verifying the identity of the requesting user].

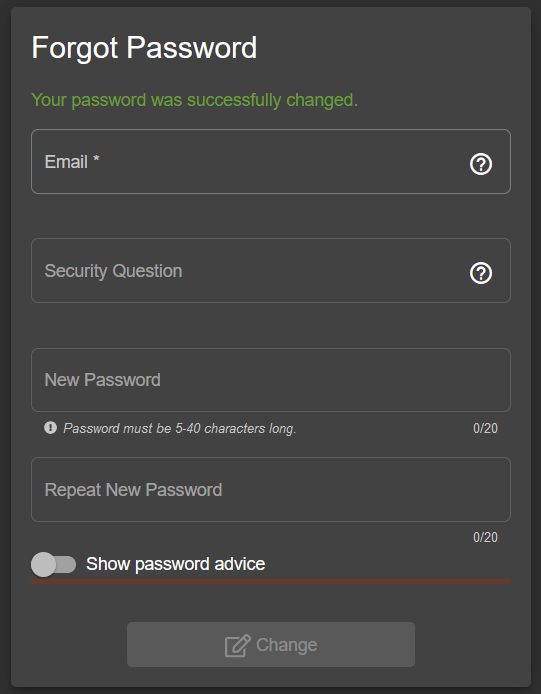
**Steps to Reproduce:**

Step 1: After finding out the administration page take any email id and copy paste it on any search engine you will get the answer to the security question

Step2 : for jim@juice-sh.op the answer was samuel

step 3 : Properly input the answers and change the password

****

****

**Impact:** The impact of an unauthorized password change can be severe, leading to account takeover and unauthorized access to sensitive user data. An attacker can exploit this vulnerability to gain control over user accounts, potentially accessing personal information, financial data, and other confidential information. This can result in identity theft, financial loss, and damage to user trust. Moreover, if administrative or privileged accounts are compromised, the attacker could gain control over the entire application or system, leading to widespread security breaches and potential data leaks.

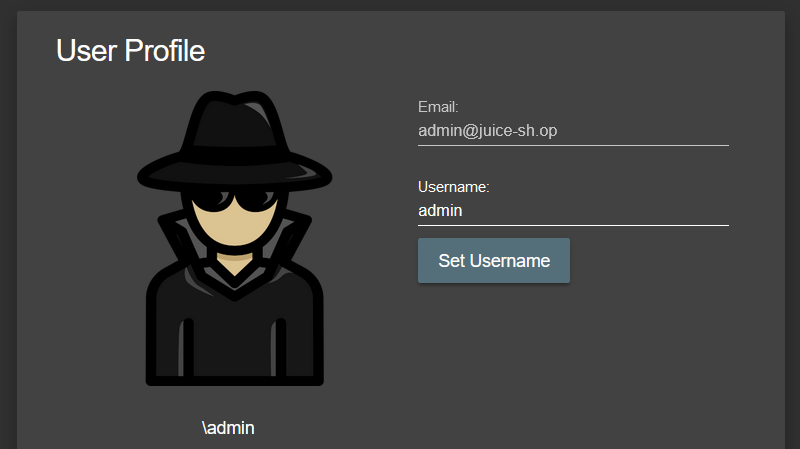
**Mitigation/Recommendation:** To mitigate the risk of unauthorized password change, it is crucial to implement robust authentication and authorization checks for all password change functionalities. Ensure that users are properly authenticated before allowing password changes, typically by requiring the current password, security questions, or multi-factor authentication. Implementing rate limiting and monitoring for unusual password change requests can also help detect and prevent unauthorized attempts. Regular security reviews and penetration testing should be conducted to identify and address such vulnerabilities. Educating developers on secure coding practices and maintaining up-to-date security patches are essential for preventing unauthorized password changes and other security issues.

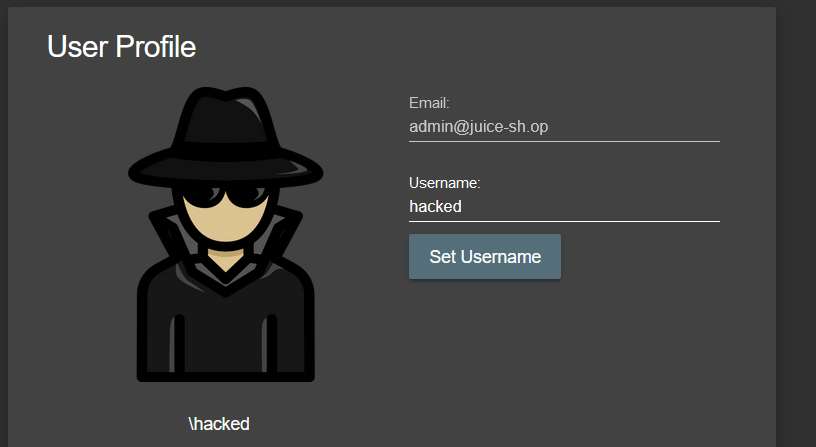
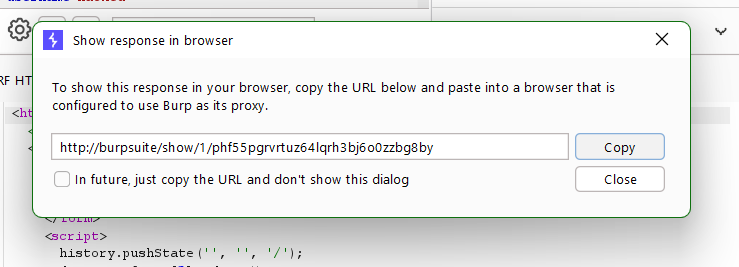
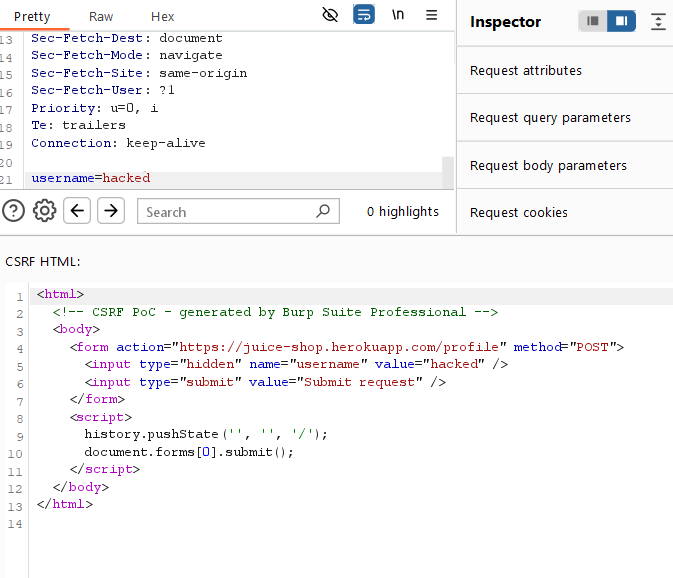
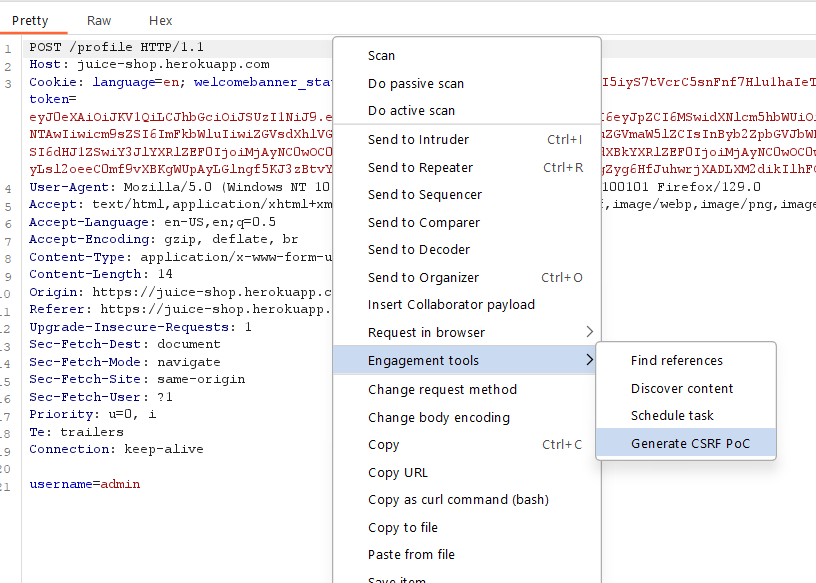
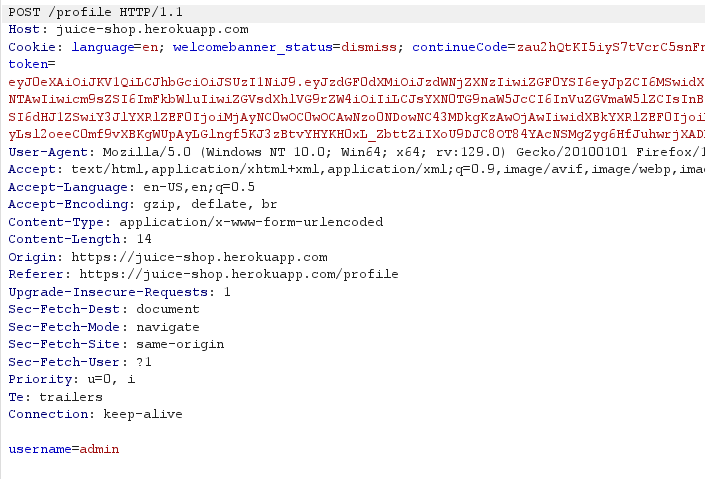
**Title: CSRF**

Severity:

Description:

Steps to Reproduce:





Impact:

Mitigation/Recommendation: