

# Penetration Testing Report

June 14<sup>th</sup>, 2018

Report For:

Digital Egypt Pioneers Initiative

Prepared by:

Abdelrahman Abdelhady Mohamed

Abdelrahman Emad Hegab

Zeyad Karim Mansoor

Sameer Mohamed Mohamed

Amr Mounir Ismael

## Table of Contents

Executive Summary .....	4
Assessment Summary .....	4
Strategic Recommendations .....	4
1 Technical Summary.....	5
1.1 Scope.....	5
1.2 Post Assessment Clean-up .....	5
1.3 Risk Ratings .....	5
1.4 Findings Overview.....	5
2 Technical Details.....	7
2.1 User Impersonation - Improper Access Control.....	7
2.2 Data Exfiltration – Improper Access Control .....	9
2.3 Privilege escalation .....	11
2.4 Unvalidated Redirect.....	13
2.5 Exposure of Data to an Unauthorized Control Sphere.....	15
2.6 URL Redirection to Untrusted Site (‘Open Redirect’).....	17
2.7 Cross-Site Request Forgery (CSRF) .....	19
2.8 Unrestricted Upload of File with Dangerous Type .....	21
2.9 Security Misconfiguration – Replay Attack.....	25
2.10 Missing Brute Force Protection.....	27
2.11 Missing ‘Strict-Transport-Security’ header.....	28
2.12 Overlay Permissive Cross-domain Whitelist.....	28
2.13 Missing Error Handling Leads to Information Exposure.....	29
2.14 Frameable response (Clickjacking).....	30
3 Appendices.....	31
3.1 Penetration Testing Methodologies .....	31
3.1.1 Web/API Application Assessment.....	31
3.1.2 External Infrastructure Assessment.....	33
3.1.3 Mobile Application Assessment .....	34

## Document Control

### Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

### Proprietary Information

The content of this document is considered proprietary information and should not be disclosed outside of the recipient organization's network.

PenTest-Hub gives permission to copy this report for the purposes of disseminating information within your organization or any regulatory agency.

---

#### Document Version Control

Issue No.	Issue Date	Issued By	Change Description
0.1	18/01/2018	[REDACTED]	Draft for internal review only
1.0	23/01/2018	[REDACTED]	Released to client

---

#### Document Distribution List

[REDACTED]	Project Sponsor, [REDACTED] ([REDACTED])
[REDACTED]	Security Consultant, PenTest-Hub
[REDACTED]	CEO, PenTest-Hub

## Executive Summary

██████ engaged PenTest-Hub (part of SecureStream group) to conduct a security assessment and penetration testing against currently developed web application project. The purpose of the engagement was to utilize active exploitation techniques in order to evaluate the security of the application against best practice criteria, to validate its security mechanisms and identify possible threats and vulnerabilities. The assessment provides insight into the resilience of the application to withstand attacks from unauthorized users and the potential for valid users to abuse their privileges and access.

This current report details the scope of testing conducted and all significant findings along with detailed remedial advice. The summary below provides non-technical audience with a summary of the key findings and section two of this report relates the key findings and contains technical details of each vulnerability that was discovered during the assessment along with tailored best practices to fix.

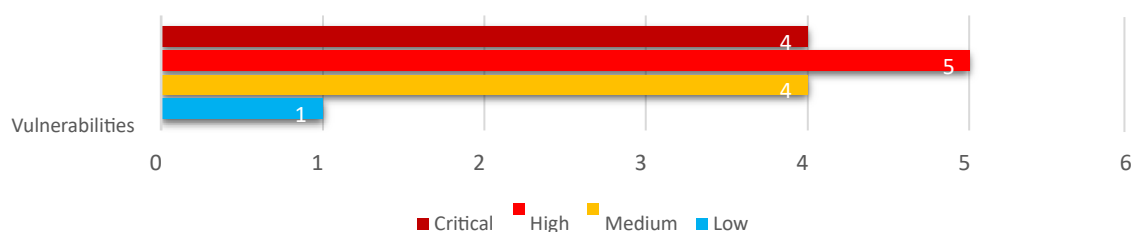
## Assessment Summary

Based on the security assessment for ██████ web applications the current status of the identified vulnerabilities set the risk at a **CRITICAL** level, which if not addressed in time (strongly recommended before going in a live production environment), these vulnerabilities could be a trigger for a cybersecurity breach. These vulnerabilities can be easily fixed by following the best practices and recommendation given throughout the report.

The following table represents the penetration testing in-scope items and breaks down the issues, which were identified and classified by severity of risk. (note that this summary table does not include the informational items):

Phase	Description	Critical	High	Medium	Low	Total
1	Web/API Penetration Testing	4	5	4	1	14
	Total	3	5	5	1	14

The graphs below represent a summary of the total number of vulnerabilities found up until issuing this current report:



## Strategic Recommendations

We recommend addressing the **CRITICAL** and **HIGH** vulnerabilities before go-live.

## 1 Technical Summary

### 1.1 Scope

The security assessment was carried out in the pre-production environment and it included the following scope:

○ ]		[IP:		○
] ○		[IP:		]
○ ]		[IP:		○
] ○		[IP:		]
		[IP:		
		[IP:		

### 1.2 Post

### Assessment Clean-up

Any test accounts, which were created for the purpose of this assessment, should be disabled or removed, as appropriate, together with any associated content.

### 1.3 Risk Ratings

The table below gives a key to the risk naming and colours used throughout this report to provide a clear and concise risk scoring system.

It should be noted that quantifying the overall business risk posed by any of the issues found in any test is outside our scope. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable by the business.

#	Risk Rating	CVSSv3 Score	Description
1	CRITICAL	9.0 - 10	A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible.
2	HIGH	7.0 – 8.9	A vulnerability was discovered that has been rated as high. This requires resolution in a short term.
3	MEDIUM	4.0 – 6.9	A vulnerability was discovered that has been rated as medium. This should be resolved throughout the ongoing maintenance process.
4	LOW	1.0 – 3.9	A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks.
5	INFO	0 – 0.9	A discovery was made that is reported for information. This should be addressed in order to meet leading practice.

### 1.4 Findings Overview

All the issues identified during the assessment are listed below with a brief description and risk rating for each issue. The risk ratings used in this report are defined in Risk Ratings Section.

Ref	Description	Risk
#####-1-1	User Impersonation - Improper Access Control	CRITICAL
#####-1-2	Data exfiltration - Improper Access Control	
#####-1-3	Privilege escalation	
#####-1-4	Unvalidated Redirect	
#####-1-5	Exposure of Data to an Unauthorized Control Sphere	HIGH
#####-1-6	URL Redirection to Untrusted Site ('Open Redirect')	HIGH
#####-1-7	Cross-Site Request Forgery (CSRF)	HIGH
#####-1-8	Unrestricted Upload of File with Dangerous Type	HIGH
#####-1-9	Security Misconfiguration – Replay Attack	MEDIUM
#####-1-10	Missing Brute Force Protection	
#####-1-11	Missing 'Strict-Transport-Security' header	
#####-1-12	Overly Permissive Cross-domain Whitelist	
#####-1-13	Missing Error Handling Leads to Information Exposure	LOW
#####-1-14	Frameable response (Clickjacking)	

## 2 Technical Details

### 2.1 Null Byte Injection to Access Restricted Files

Ref ID: [REDACTED]-1-1

The null byte injection vulnerability allows an attacker to bypass file extension restrictions and access files that are otherwise blocked by the application. By appending a null byte (%00), the backend interprets the file as its original format, while the frontend only sees the allowed extension.

In this case, the Juice Shop application restricts file downloads to .md and .pdf extensions, but by using a null byte injection, the attacker can access other file types, such as .bak, leading to exposure of sensitive information (e.g., a backup of the package.json file).

#### Vulnerability Details:

Severity:	Medium
Affects:	/ftp
Parameter(s)	File download path (e.g., /ftp/package.json.bak)
Attack Vectors	Authorization bearer, all post parameters
References:	<a href="#">OWASP: Null Byte Injection</a> <a href="#">CWE-158: Improper Neutralization of Null Byte or NUL Character</a>

#### Evidence

```
{
  "name": "juice-shop",
  "version": "6.2.0-SNAPSHOT",
  "description": "An intentionally insecure JavaScript Web Application",
  "homepage": "http://owasp-juice.shop",
  "author": "Björn Kimminich <bjoern.kimminich@owasp.org> (https://kimminich.de)",
  "contributors": [
    "Björn Kimminich",
    "Jannik Hollenbach",
    "Aashish683",
    "greenkeeper[bot]",
    "MarcRler",
    "agrawalarpit14",
    "Scar26",
    "CaptainFreak",
    "Supratik Das",
    "JuiceShopBot",
    "the-pro",
    "Ziyang Li",
    "aaryan10",
    "m4ll1c3",
    "Timo Pagel",
    "..."
  ],
  "private": true,
  "keywords": [
    "web security",
    "web application security",
    "webappsec",
    "owasp",
    "pentest",
    "pentesting",
    "security",
    "vulnerable",
    "vulnerability",
    "broken",
    "bodgeit"
  ],
  "dependencies": {
    "body-parser": "~1.18",
    "colors": "~1.1",
    "config": "~1.28",
    "cookie-parser": "~1.4",
    "cors": "~2.8",
    "dottie": "~2.0",
    "epilogue-js": "~0.7",
    "errorhandler": "~1.5",
    "express": "~4.16",
    "express-jwt": "0.1.3",
    "fs-extra": "~4.0",
```

**Steps to Reproduce:**

1. Navigate to the /ftp directory in the browser
2. Attempt to download the package.json.bak file. The application will block the download and show the following error:  
403 Error: Only .md and .pdf files are allowed!
3. Modify the URL to include a null byte and an allowed extension, replacing the original path with:  
<http://owasp-juice.shop/ftp/package.json.bak%2500.md>  
(The %2500 is the URL-encoded form of the null byte, and .md is an allowed extension.)
4. Submit the modified URL and observe that the file is downloaded successfully.

**Remediation Guidance:**

- Input Validation: Implement stricter validation on user inputs, ensuring that null bytes or other special characters are properly filtered out and sanitized. Never rely solely on client-side or extension-based filtering.
- Secure File Handling: Ensure that file names are sanitized before being processed by the server, particularly when dealing with user-supplied input or URLs.
- Restrict Sensitive File Access: Consider using authentication and authorization checks before serving any file, especially backup files, to prevent unauthorized access.
- File Extension Checks: Instead of relying on file extension-based restrictions, validate the actual content type of the file before allowing downloads (e.g., using MIME type validation).



## 2.2 Unauthorized Access through Weak Credentials

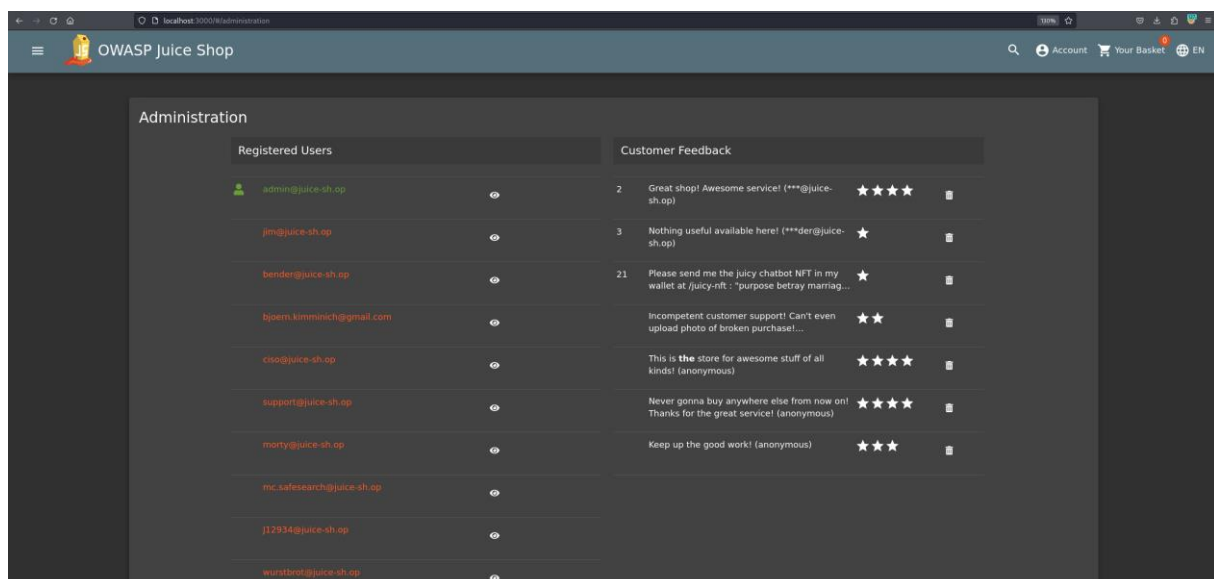
Ref ID: [REDACTED]-1-2

The Juice Shop application exposes users' emails within the product review section, which includes sensitive accounts like the **admin**. Combined with weak password policies, an attacker can use a brute-force attack to guess the admin's credentials and gain unauthorized access to privileged areas of the application, such as the **administration panel**.

### Vulnerability Details:

<b>Severity:</b>	Critical
<b>Affects:</b>	Product reviews section, displaying user email addresses (including the admin's) publicly.
<b>Parameter(s)</b>	Admin's email and weak password authentication.
<b>Attack Vectors</b>	<ul style="list-style-type: none"> <li>The <b>admin email</b> (admin@juice-sh.op) was revealed publicly in the reviews section.</li> <li>Using this email, an attacker could perform a <b>brute-force attack</b> on the login page with a password wordlist to guess the admin password.</li> <li>The <b>password</b> turned out to be "<b>admin123</b>", a weak and easily guessable password.</li> <li>After successful login as the admin, the attacker can access the <b>/administration</b> page, which is the admin panel for the application, leading to full control over the system.</li> </ul>
<b>References:</b>	<ul style="list-style-type: none"> <li><a href="#">OWASP: Authentication and Credential Management</a></li> <li><a href="#">CWE-521: Weak Password Requirements</a></li> </ul>

### Evidence



### Steps to Reproduce:

1. Navigate to the Product Reviews section on the Juice Shop website.
2. Locate a review written by the admin with the email [admin@juice-sh.op](mailto:admin@juice-sh.op).
3. Go to the login page of Juice Shop and enter the admin email as the username.
4. Using a password brute-force tool (e.g., Hydra, Burp Suite Intruder) with a common wordlist, attempt to guess the password. In this case, the password was found to be admin123.
5. Upon successful login, navigate to /administration in the browser.
6. The admin panel will be accessible, allowing the attacker full administrative control.

**Remediation Guidance:**

1. Remove Sensitive Information: Do not expose sensitive information like email addresses in public areas of the website, especially for high-privilege accounts such as the admin.
2. Enforce Strong Password Policies:
  - Implement stronger password requirements, including a minimum length of at least 12 characters, and enforce the use of complex passwords (upper and lower case letters, numbers, special characters).
  - Prevent the use of commonly used passwords (e.g., admin123, password, etc.) by cross-referencing against a list of known weak passwords.
3. Rate Limiting and Account Lockout:
  - Implement rate limiting on login attempts to prevent brute-force attacks.
  - Use account lockout mechanisms after a set number of failed login attempts, with an option for users to reset their passwords through email.
4. Multi-Factor Authentication (MFA):
  - Add MFA to the admin and other high-privilege accounts to reduce the likelihood of unauthorized access, even if credentials are compromised.
5. Role-Based Access Control:
  - Ensure that even if attackers gain access to an account, they have limited permissions. Avoid granting administrative privileges based solely on username.

## 2.3 Insecure Direct Object Reference (IDOR) in Basket Access

Ref ID: [REDACTED]-1-3

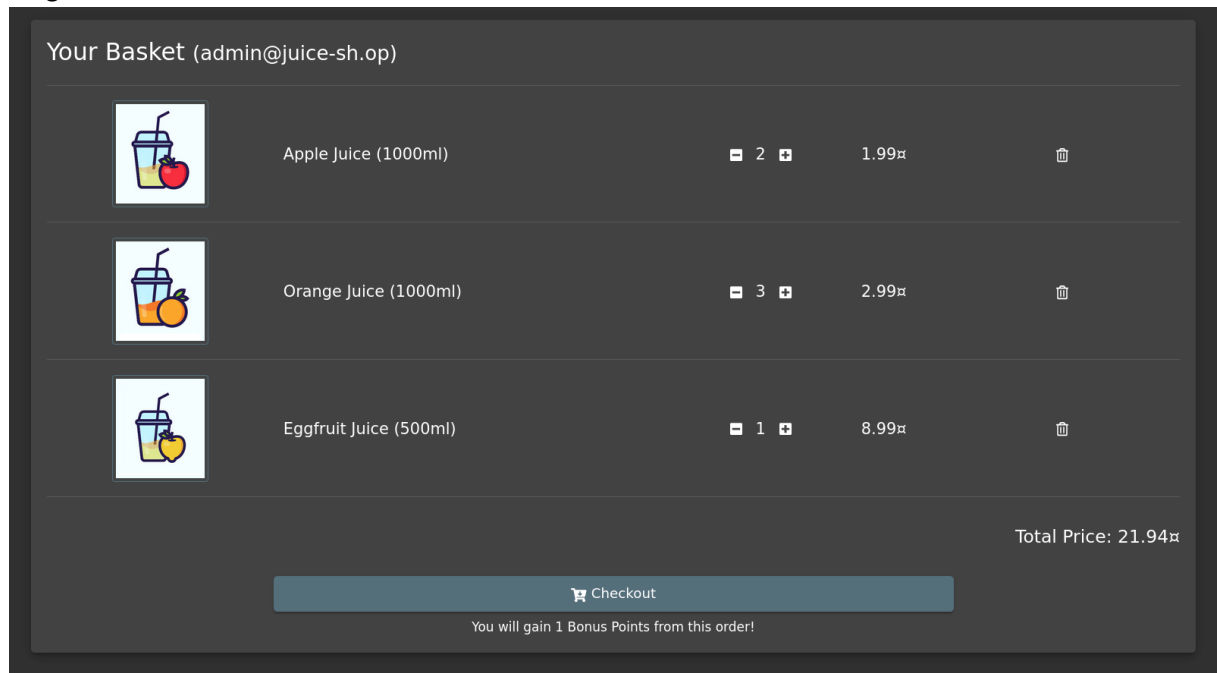
An Insecure Direct Object Reference (IDOR) vulnerability occurs when an application uses user-supplied input (such as an ID) to directly access objects (such as user accounts or baskets) without performing proper access control checks. In this case, by manipulating the basket ID in the URL, an attacker can gain access to other users' baskets and view their contents.

### Vulnerability Details:

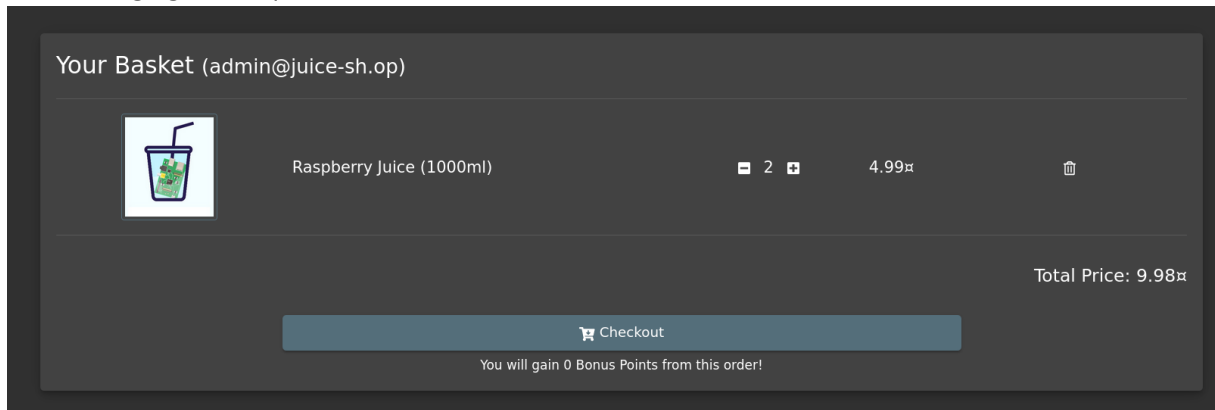
<b>Severity:</b>	High
<b>Affects:</b>	/rest/basket/{id}
<b>Parameter(s)</b>	id (the basket ID)
<b>User Role Affected:</b>	All users with baskets are vulnerable, allowing anyone to view another user's basket by modifying the id parameter.
<b>Attack Vectors</b>	<ul style="list-style-type: none"> <li>A request is sent when accessing the basket</li> <li>An attacker can intercept and modify the request using a tool like <b>Burp Suite</b> to change the id value</li> <li>If proper authorization checks are not in place, the server responds with the basket details of the user with the modified ID, leading to exposure of personal data, product selections, or order history.</li> </ul>
<b>References:</b>	<ul style="list-style-type: none"> <li><a href="#">OWASP: Insecure Direct Object Reference (IDOR)</a></li> <li><a href="#">CWE-200: Exposure of Sensitive Information to an Unauthorized Actor</a></li> </ul>

### Evidence

#### Original Basket



### After changing the Request



### Steps to Reproduce:

1. Log into your account on Juice Shop and add an item to your basket.
2. Open Burp Suite and enable Intercept.
3. Navigate to your basket and observe the request being sent to `/rest/basket/1` (or whatever your basket ID is).
4. In Burp Suite, modify the id in the request URL from `/rest/basket/1` to `/rest/basket/2` and forward the request.
5. Observe that the basket of another user is returned in the server's response, confirming the IDOR vulnerability.

### Remediation Guidance:

1. Implement Object-Level Authorization Checks:
  - Ensure that access to sensitive resources like baskets is restricted based on the authenticated user's session. When processing a request, verify that the user ID matches the resource owner (e.g., the basket owner) before providing access.
2. Use Indirect Object References:
  - Replace direct object references (e.g., numerical IDs) with indirect references such as tokens or hashes that are mapped internally to actual objects. This prevents attackers from guessing or manipulating IDs.
3. Parameter Validation:
  - Validate and sanitize user input at both the client and server levels. Ensure that users can only interact with resources they are authorized to access.
4. Log and Monitor Suspicious Access:
  - Implement logging and monitoring for abnormal requests, such as multiple requests for different basket IDs, which could indicate an attempt to exploit IDOR vulnerabilities.

## 2.4 Improper Order's Value Validation

#####-1-3

Ref ID: 

The application allows users to enter negative amounts for items in the cart, which can lead to unauthorized discounts or refunds.

### Vulnerability Details:

Severity:	High
Affects:	Minus Value
Parameter(s)	Quantity
User Role Affected:	All users with baskets are vulnerable, allowing anyone to view another user's basket by modifying the id parameter.
Attack Vectors	Burp Suite: Manipulation of request parameters (POST parameters)
References:	<ul style="list-style-type: none"><li>• <a href="#">OWASP: Input Validation and Encoding</a></li><li>• <a href="#">CWE-20: Improper Input Validation</a></li></ul>

### Evidence

## OWASP Juice Shop

### Order Confirmation

Customer: gugtenterf+yxlim@gmail.com

Order #: 5965-f4e31bf953961d82

Date: 2024-10-19

-1000x Melon Bike (Comeback-Product 2018 Edition) ea. 2999 = -2999000

Delivery Price: 0.99

**Total Price: -2998999.01**

**Bonus Points Earned: -300000**

(The bonus points from this order will be added 1:1 to your wallet -fund for future purchases!)

Thank you for your order!

### Steps to Reproduce

1. Open Burp Suite and configure it to intercept traffic from your browser.
2. Navigate to the Juice Shop application and add an item to your cart.
3. Intercept the request sent when you proceed to checkout.
4. In Burp Suite, locate the parameter responsible for the quantity. And it is "quantity"
5. Modify the value of the quantity parameter to a negative number (e.g., -1000).
6. Forward the modified request.
7. Observe the application response to confirm the exploit (e.g., a negative total or successful transaction).

Ref ID: 

### Remediation Guidance:

- **Input Validation:** Implement validation to ensure that quantity fields do not accept negative values.
- **User Feedback:** Provide clear error messages for invalid input instead of allowing silent failures.

## 2.5 Deluxe Fraud

#####-1-4

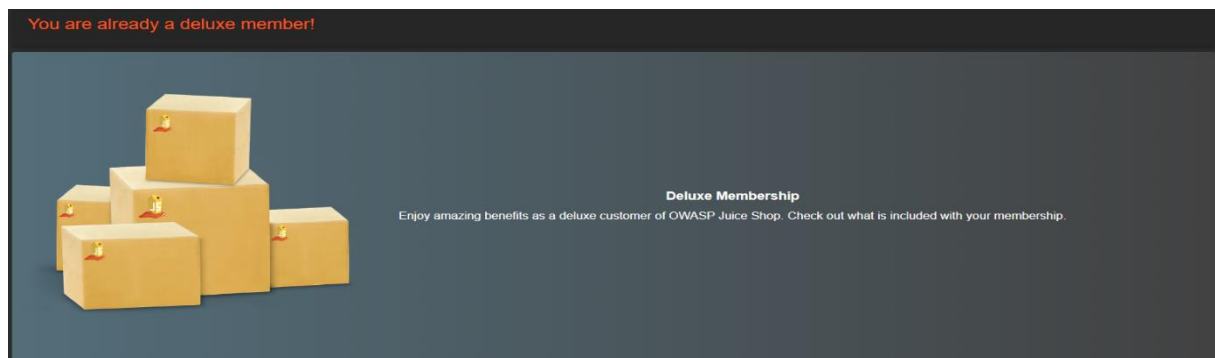
Ref ID: 

Users can exploit vulnerabilities in the membership registration process to gain unauthorized deluxe membership benefits. By manipulating UI elements and altering the payment Mode parameter, they can bypass payment requirements. This leads to potential revenue loss and undermines the integrity of the system.

### Vulnerability Details:

<b>Severity:</b>	High
<b>Affects:</b>	Membership registration process
<b>Parameter(s)</b>	Disabled button, payment Mode
<b>Attack Vectors</b>	<ol style="list-style-type: none"><li>1. UI Manipulation: Removing the disabled state from the membership button in the front end to submit the registration without payment. Request</li><li>2. Manipulation: Changing the payment Mode parameter in the POST request from "wallet" to "paid", allowing unauthorized access.</li></ol>
<b>References:</b>	<ul style="list-style-type: none"><li>• OWASP: Input Validation and Encoding</li><li>• CWE-20: Improper Input Validation</li></ul>

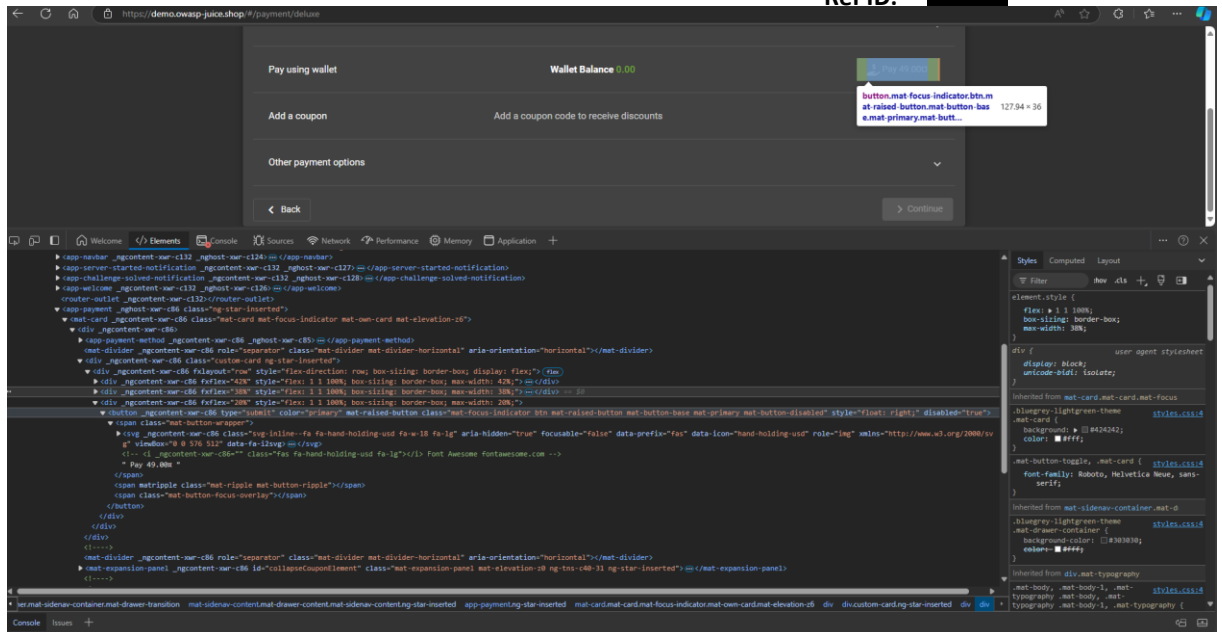
### Evidence



### Steps to Reproduce:

1. **Capture the Request:** Use a tool like Burp Suite to intercept the request to `/rest/deluxe-membership`.
2. **UI Manipulation:**
  - Inspect the button using browser developer tools.
  - Remove the block on button by removing from UI: `disabled=true` attribute and `mat-button-disabled` from the membership button.
  - Submit the form.
3. **Request Manipulation:**
  - Modify the `paymentMode` in the JSON body of the intercepted request from "wallet" to "" .
  - Send the modified request.
4. **Observe the Response:** Check if the application processes the membership registration successfully without any payment.

Ref ID:



## Remediation Guidance:

- **Input Validation:** Implement strict server-side validation for both the paymentMode parameter and any front-end inputs to prevent unauthorized actions.
- **Business Logic Enforcement:** Ensure all membership registration processes adhere to defined business rules and payment requirements.
- **Client-Side Controls:** Enhance UI controls to prevent manipulation and ensure that important validations are enforced on the server side.
- **Monitoring and Logging:** Track all membership registration requests and changes to detect and respond to suspicious activities.



## 2.6 CAPTCHA Bypass

#####-1-5

Ref ID: [REDACTED]

The CAPTCHA mechanism in Juice Shop is flawed, allowing attackers to bypass it with minimal effort. CAPTCHA is intended to verify that the user is human, but this implementation can be bypassed because the server does not enforce the CAPTCHA validation strictly. Attackers can send requests without solving the CAPTCHA, and the server accepts them as valid.

### Vulnerability Details:

<b>Severity:</b>	Medium
<b>Affects:</b>	Comment submission and user interaction processes
<b>Parameter(s)</b>	CAPTCHA ID and verification token
<b>Attack Vectors</b>	Reusing the same CAPTCHA ID and associated token to submit multiple comments without re-validation.
<b>References:</b>	<ul style="list-style-type: none"> <li>- OWASP: Input Validation and Encoding</li> <li>- CWE-20: Improper Input Validation</li> </ul>

### Evidence

```
"captchaId":5,
"captcha":"5",
"comment":"sameer123 (**94f6051@mailmaxy.one) ",
"rating":5,
"UserId":23
```

<div> <div>Send</div> <div>Cancel</div> <div>&lt;</div> <div>&gt;</div> </div> <div> <div>Request</div> <div>Response</div> </div> <div> <div>Pretty</div> <div>Raw</div> <div>Hex</div> <div>Render</div> </div>			
<pre>1 HTTP/1.1 201 Created 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Location: /api/Feedbacks/84 8 Content-Type: application/json; charset=utf-8 9 Content-Length: 183 10 ETag: W/"b7-lbtHNLb8iAkGjI44XB3IGS10JjU" 11 Vary: Accept-Encoding 12 Date: Sat, 19 Oct 2024 12:26:52 GMT 13 Connection: keep-alive 14 Keep-Alive: timeout=5 15 16 {   "status": "success",   "data": {     "id": 85,     "comment": "sameer123 (**94f6051@mailmaxy.one) ",     "rating": 5,     "UserId": 23,     "updatedAt": "2024-10-19T12:26:52.155Z",     "createdAt": "2024-10-19T12:26:52.155Z"   } }</pre>		<pre>1 HTTP/1.1 201 Created 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Location: /api/Feedbacks/84 8 Content-Type: application/json; charset=utf-8 9 Content-Length: 183 10 ETag: W/"b7-lbtHNLb8iAkGjI44XB3IGS10JjU" 11 Vary: Accept-Encoding 12 Date: Sat, 19 Oct 2024 12:25:56 GMT 13 Connection: keep-alive 14 Keep-Alive: timeout=5 15 16 {   "status": "success",   "data": {     "id": 84,     "comment": "sameer (**94f6051@mailmaxy.one) ",     "rating": 5,     "UserId": 23,     "updatedAt": "2024-10-19T12:25:56.898Z",     "createdAt": "2024-10-19T12:25:56.898Z"   } }</pre>	

### Steps to Reproduce:

Ref ID: 

1. **Submit a Valid Comment:** Complete the CAPTCHA challenge and submit a comment to obtain a valid CAPTCHA ID.
2. **Capture the Request:** Use tools like Burp Suite to intercept the request containing the CAPTCHA ID.
3. **Reuse CAPTCHA ID:** Modify subsequent requests to reuse the same CAPTCHA ID for additional comments.
4. **Submit Multiple Comments:** Send the modified requests to the server.
5. **Evaluate Responses:** Verify whether the application accepts multiple submissions without requiring new CAPTCHA validation.

### Remediation Guidance:

- **Implement Unique CAPTCHA Tokens:** Generate a new CAPTCHA ID for each submission to ensure that previous IDs cannot be reused.
- **Validate CAPTCHA on Each Request:** Ensure that the CAPTCHA verification process is enforced for every comment submission.
- **Monitor Submission Patterns:** Track user submissions for unusual activity, allowing for proactive responses to potential abuse.

## 2.7 Changing another user's Password

HIGH

**-1-6**

Ref ID:

Ref ID:

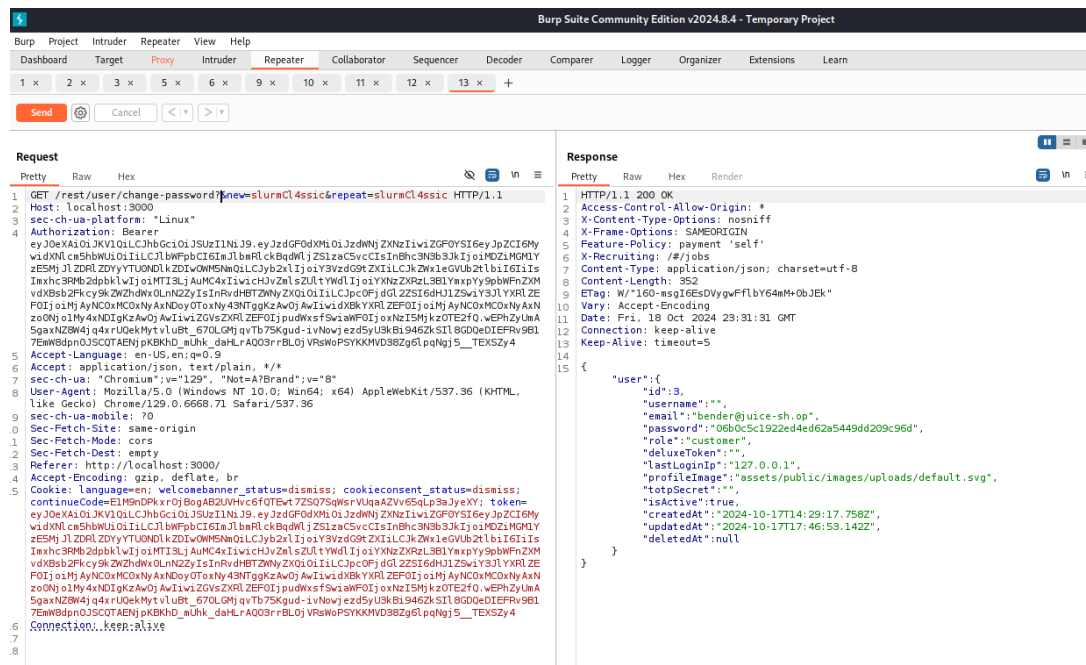
alter

The password change functionality in the application allows unauthorized users to the password of any user, bypassing proper authentication and authorization checks.

### Vulnerability Details:

<b>Severity:</b>	Critical
<b>Affects:</b>	Password Change Function
<b>Parameter(s)</b>	Password and User ID
<b>Attack Vectors</b>	The attacker can intercept the password change request using tools like Burp Suite and modify the User ID parameter to change the password for any user
<b>References:</b>	OWASP: Authentication and Session Management CWE-287: Improper Authentication

## Evidence



### Steps to Reproduce:

1. Open **Burp Suite** and configure it to intercept traffic from your browser.
2. Log in to the application and navigate to the "Change Password" feature.
3. Intercept the password change request when submitting a password change.
4. Modify the User ID parameter in the intercepted request to another user's ID.
5. Forward the modified request.
6. Check if the password for the other user has been successfully changed by attempting to log in with their account and the new password.

**Remediation Guidance:**

- 1- **Implement Proper Authentication and Authorization:** Ensure that authenticated users can request password changes, and verify User ID parameter matches the logged-in user's ID. [Redacted] only  
Ref ID: [Redacted] that the
- 2- **Implement Multi-factor Authentication (MFA):** Require MFA for sensitive actions like password changes to add an extra layer of security.
- 3- **Log and Monitor Password Change Attempts:** Enable logging and monitoring for password change requests to detect any suspicious activities.

## 2.8 SQL Injection retrieving all available products, including hidden

offers

#####-1-7

Ref ID:

The application contains a hidden Christmas special offer that is not displayed on the main page. By intercepting the product-related requests using Burp Suite, an attacker can exploit a SQL Injection vulnerability. By manipulating the Product ID parameter with a crafted SQL query, the attacker can retrieve all available products, including hidden offers such as the Christmas special. This allows the attacker to order hidden products without proper authorization or visibility.

### Vulnerability Details:

Severity:	High
Affects:	Product and Offer Discovery System
Parameter(s)	Product ID
Attack Vectors	<ul style="list-style-type: none"><li>• Using Burp Suite, the attacker intercepts a request related to product listings or special offers.</li><li>• By injecting a SQL query into the Product ID parameter, the attacker retrieves a list of all available products, including hidden offers (like the Christmas Special).</li><li>• The attacker can then order hidden products that are not displayed on the webpage.</li></ul>
References:	<ul style="list-style-type: none"><li>- <a href="#">OWASP: SQL Injection</a></li><li>- <a href="#">CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</a></li></ul>

Evidence

Pretty	Raw	Hex	Render
--------	-----	-----	--------

**Burp** Project **Intruder** **Repeater** View Help

Dashboard Target **Proxy** Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

Intercept HTTP history WebSockets history Match and replace Proxy settings

Intercept on  Forward all  Drop

Request to http://localhost:3000 [127.0.0.1]

Time	Type	Direction	Host	Method	URL	Status code
20:19:06 18 Oct 2024	WebSocket	→ To server	localhost		http://localhost:3000/socket.io/?EIO=4&transport=websocket&sid=CbnfaffNkEPAAci	
20:19:06 18 Oct 2024	HTTP	→ Request	localhost	POST	http://localhost:3000/api/basketitems/	
20:19:14 18 Oct 2024	HTTP	→ Request	localhost	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PAY988C	
20:19:28 18 Oct 2024	HTTP	→ Request	localhost	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PAY98ah	
20:19:37 18 Oct 2024	HTTP	→ Request	localhost	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PAY9JHf	
20:20:01 18 Oct 2024	HTTP	→ Request	localhost	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PAY9JWl	
20:20:26 18 Oct 2024	HTTP	→ Request	localhost	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PAY9Pdp	

Pretty Raw Hex

[illegible]

Request attributes

Request query parameters

[Request cookies](#)

Ref ID: 

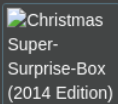
## Your Basket (bender@juice-sh.op)



Raspberry Juice (1000ml)

- 1 +

4.99€



Christmas Super-Surprise-Box (2014 Edition)

- 1 +

29.99€



Total Price: 34.98€

 Checkout

You will gain 3 Bonus Points from this order!

**Steps to Reproduce:**

1. Open Burp Suite and configure it to intercept traffic from your browser.
2. Navigate to the product page or any page that retrieves products.
3. Intercept the request related to product listings (e.g., Product ID request).
4. Inject an SQL query into the Product ID parameter.
5. Forward the modified request.
6. Check the response to see if all products, including hidden ones (e.g., Christmas special offers), are listed.
7. Attempt to order a hidden product by selecting it from the retrieved product list and proceeding to checkout.

**Remediation Guidance:**

1. Use Prepared Statements or Parameterized Queries: Ensure that SQL queries properly parameterized to prevent SQL injection vulnerabilities. [REDACTED] are  
Ref ID: [REDACTED]
2. Implement Input Validation: Validate and sanitize all user input, particularly for parameters like Product ID, to prevent malicious SQL code from being executed.
3. Restrict Sensitive Product Information Access: Ensure that hidden or special products, such as the Christmas special, are properly restricted from unauthorized access by checking user permissions.
4. Conduct Regular Security Audits: Regularly audit the application's code and database queries
5. to identify and mitigate potential SQL injection risks.
6. Use Web Application Firewalls (WAF): Deploy WAFs to detect and block SQL injection attempts.



## 2.9 File Upload Vulnerability –Text File

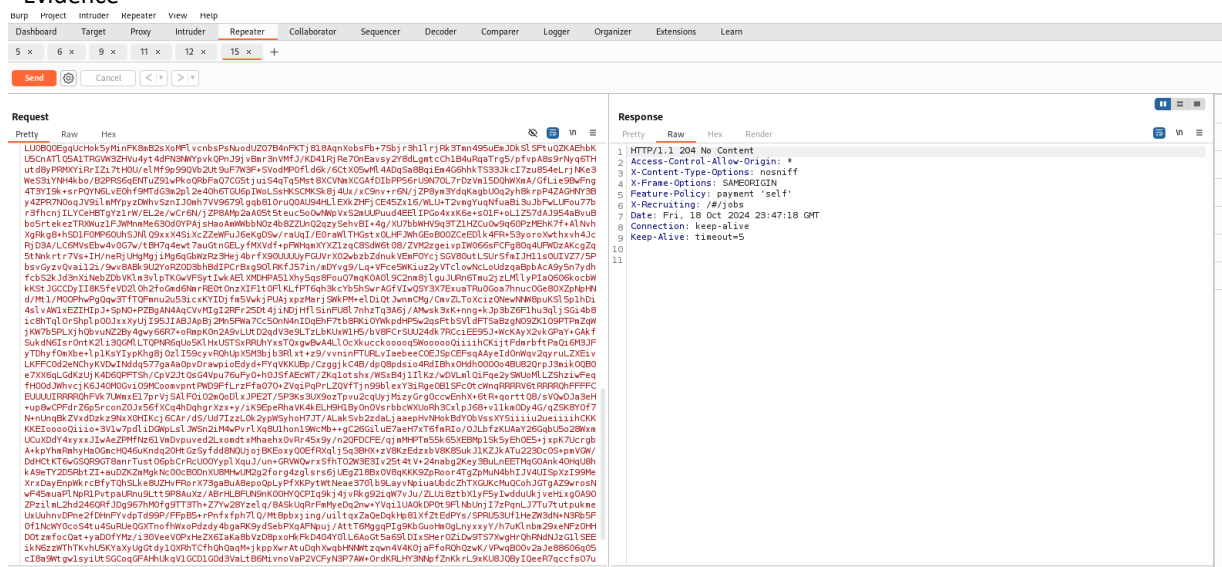
#####-1-8

The file upload feature allows users to upload text files, but the application fails to validate the content and type of the files being uploaded. This can lead to the execution of malicious scripts or injection of harmful code.

### Vulnerability Details:

<b>Severity:</b>	Critical
<b>Affects:</b>	File Upload Feature
<b>Parameter(s)</b>	Uploaded File Content
<b>Attack Vectors</b>	Attackers use Burp Suite to upload text files (e.g. image) containing malicious scripts, bypassing validation, leading to potential execution of malicious code on the server.
<b>References:</b>	<ul style="list-style-type: none"> <li>OWASP: Unrestricted File Upload</li> <li>CWE-434: Unrestricted Upload of File with Dangerous Type</li> </ul>

### Evidence



The screenshot shows the Burp Suite interface with a request and response. The request is a POST to /upload with a file named '1.txt' containing a shell command. The response is a 204 No Content.

### Steps to Reproduce:

1. Open Burp Suite and configure it to intercept traffic from your browser.
2. Navigate to the file upload feature and upload a text file that can be uploaded.
3. Prepare a text file (image decode base64).
4. Intercept the file upload request and review the parameters to ensure the file content is being sent.
5. Forward the request to upload the file.
6. Monitor the application response to see if the image is accepted and processed without proper validation.

Ref  
ID:



### **Remediation Guidance:**

- Implement File Type and Content Validation: Ensure that uploaded files are strictly validated by content and type. Reject any file that doesn't meet the expected criteria (e.g., no script or executable code).
- Use File Sanitization Tools: Apply file sanitization mechanisms that strip harmful content or encoding before accepting and processing files on the server.
- Store Files Securely: Use non-web accessible directories to store uploaded files and ensure that they are not executed or served directly.
- Limit Permissions: Apply strict file permissions, ensuring that uploaded files cannot be executed on the server.
- Log and Monitor File Upload Activities: Enable logging for all file upload activities and monitor for unusual or malicious behavior.

Ref  
ID:

#####-1-9

## 2.10 Client-Side XSS Protection Bypass

: The Juice Shop application relies solely on client-side protections (such as the browser's XSS filter) to prevent Cross-Site Scripting (XSS) attacks. By disabling these protections, an attacker can bypass XSS protections and execute malicious scripts.

### Vulnerability Details:

**Affects:** https://[REDACTED]/

**References:** <https://cwe.mitre.org/data/definitions/307.html>  
[https://www.owasp.org/index.php/Blocking\\_Brute\\_Force\\_Attacks](https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks)

### Evidence

Request	Payload	Status	Error	Timeout	Length	Comment
235	aXaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
236	GXaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
237	RXaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
238	XXaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
239	YXaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
240	ZXaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
241	aYaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
242	GYaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
243	RYaG	200	<input type="checkbox"/>	<input type="checkbox"/>	57010	
244	XYaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
245	YYaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
246	ZYaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
247	aZaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
248	GZaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	
249	eZaG	302	<input type="checkbox"/>	<input type="checkbox"/>	808	

Request	Response
	<div>Raw Headers Hex HTML Render</div> <pre> HTTP/1.1 200 OK Date: Tue, 30 Apr 2019 11:23:13 GMT Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/7.3.4 X-Frame-Options: SAMEORIGIN X-Powered-By: PHP/7.3.4 Cache-Control: no-cache, no-store, must-revalidate Pragma: no-cache Expires: 0 Connection: close Content-Type: text/html; charset=UTF-8 Content-Length: 56659  &lt;!DOCTYPE html&gt; &lt;html lang="ro"&gt; &lt;head&gt;   &lt;!-- Required meta tags --&gt;   &lt;meta charset="utf-8"&gt;   &lt;meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"&gt; </pre>

### Remediation Guidance:

We recommend implementing a CAPTCHA system to limit the risk carried by possible brute force attacks against the platform.

## 2.11 Missing 'Strict-Transport-Security' header

Ref ID: [REDACTED]###-

1-10

It has been discovered that the affected application is using HTTPS, however does not use the HSTS header. The HTTP protocol by itself is clear text, meaning that any data that is transmitted via HTTP can be captured and the contents viewed. To keep data private and prevent it from being intercepted, HTTP is often tunnelled through either Secure Sockets Layer (SSL) or Transport Layer Security (TLS). When either of these encryption standards are used, it is referred to as HTTPS.

HTTP Strict Transport Security (HSTS) is an optional response header that can be configured on the server to instruct the browser to only communicate via HTTPS. This will be enforced by the browser even if the user requests a HTTP resource on the same server.

Cyber-criminals will often attempt to compromise sensitive information passed from the client to the server using HTTP. This can be conducted via various Man-in-The-Middle (MitM) attacks or through network packet captures.

### Vulnerability Details:

<b>Affects:</b>	https://[REDACTED]
<b>Parameter(s)</b>	Header
<b>Attack Vectors</b>	(shown within the evidence subsection)
<b>References:</b>	<a href="https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet">https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet</a> <a href="http://cwe.mitre.org/data/definitions/200.html">http://cwe.mitre.org/data/definitions/200.html</a>

### Evidence

#### Raw HTTP response

```
HTTP/1.1 200 OK
Content-Length: 3
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Server: Microsoft-IIS/8.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-Powered-By: ASP.NET Set-Cookie:
ARRAffinity=8c19a282e209d49c6b7b0bf1c8a4a453f3a5033961a3b5dea76f375e3d9c9f67;Path=/;HttpOnly;Domain=[REDACTED].com
Date: Fri, 22 Dec 2017 17:06:47 GMT
```

### Remediation Guidance:

Consider implementing the HSTS header.

Depending on the framework being used the implementation methods will vary, however it is advised that the Strict-Transport-Security header be configured on the server.

One of the options for this header is max-age, which is a representation (in milliseconds) determining the time in which the client's browser will adhere to the header policy. Depending on the environment and the application this time period could be from as low as minutes to as long as days.

## 2.12 Overlay Permissive Cross-domain Whitelist

Ref ID: [REDACTED]###-

1-11

Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform "cross-domain" requests using the XMLHttpRequest L2 API in a controlled manner. In the past, the XMLHttpRequest L1 API only allowed requests to be sent within the same origin as it was restricted by the same origin policy.

Cross-Origin requests have an Origin header, that identifies the domain initiating the request and is always sent to the server. CORS defines the protocol to use between a web browser and a server to determine whether a cross-origin request is allowed. In order to accomplish this goal, there are a few HTTP headers involved in this process, that are supported by all major browsers and we will cover below including: Origin, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Origin, Access-Control-AllowCredentials, Access-Control-Allow-Methods, Access-Control-Allow-Headers.

The CORS specification mandates that for non-simple requests, such as requests other than GET or POST or requests that uses credentials, a pre-flight OPTIONS request must be sent in advance to check if the type of request will have a bad impact on the data. The pre-flight request checks the methods, headers allowed by the server, and if credentials are permitted, based on the result of the OPTIONS request, the browser decides whether the request is allowed or not.

#### **Vulnerability Details:**

<b>Affects:</b>	https://[REDACTED]/login
<b>Parameter(s)</b>	Header
<b>Attack Vectors</b>	(shown within the evidence subsection)
<b>References:</b>	<a href="https://www.owasp.org/index.php/Test_Cross_Origin_Resource_Sharing_(OTG-CLIENT-007)">https://www.owasp.org/index.php/Test_Cross_Origin_Resource_Sharing_(OTG-CLIENT-007)</a> <a href="https://cwe.mitre.org/data/definitions/942.html">https://cwe.mitre.org/data/definitions/942.html</a>

#### **Evidence**

##### **Raw HTTP request.**

```
GET /ids/login HTTP/1.1
Host: [REDACTED].com
Accept: application/json, text/plain, */*
Origin: https://www.pentest-hub.com
Connection: close
```

##### **Raw HTTP response.**

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, max-age=0, private
Pragma: no-cache
Access-Control-Allow-Origin: https://www.pentest-hub.com
```

#### **Remediation Guidance:**

CORS should be using a stricter policy of allowed domains and methods and validate the origin.

### **2.13 Missing Error Handling Leads to Information Exposure**

Ref ID: [REDACTED]###-

1-13

It has been found that the application is exposing sensitive information about internal resources with unhandled errors.

#### **Vulnerability Details:**

<b>Affects:</b>	https://[REDACTED]
<b>Parameter(s)</b>	All input fields
<b>Attack Vectors</b>	Unexpected input, untreated errors
<b>References:</b>	<a href="https://cwe.mitre.org/data/definitions/544.html">https://cwe.mitre.org/data/definitions/544.html</a>

#### Evidence

```
POST /api/v1/resource/venue HTTP/1.1
Host: api.pentest.[REDACTED].com
Accept: application/json, text/plain, */*
Referer: https://pentest.[REDACTED].com/venues/new
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJlNmNmJlYy0xNTU1LTRhZDMtYWY5Ny00Z Content-Type: application/json;charset=utf-8
Content-Length: 347
Origin : https://pentest.[REDACTED].com/
Connection: close
```

```
{
  "id": "1",
  "name": "<img src=\\\"data:image/bmp;base64,Qk1+AA\\\">[REDACTED] Bay Plaza",
  "city": "[REDACTED]",
  "country": "[REDACTED]",
  "region": "[REDACTED]",
  "vendor_id": "07f9aa4b-570a-46b5-b734-253a0af97d8d",
  "building": "",
  "floor": "",
  "room_no": "",
  "seats_no": 1,
  "seat_arrangement": 1,
  "has_whiteboard": false,
  "has_pcs": false,
  "has_projector": false,
  "touchpoint_email": "fsdfsdfsdf"
}
```

#### Raw HTTP response

```
{
  "error": {
    "message": "Unexpected token a in JSON at position 61",
    "stack": "SyntaxError: Unexpected token a in JSON at position 61\n    at JSON.parse (<anonymous>)\n    at parse (/home/centos/[REDACTED]/node_modules/body-parser/lib/types/json.js:89:19)\n    at /home/centos/[REDACTED]/node_modules/bodyparser/lib/read.js:121:18\n    at invokeCallback (/home/centos/[REDACTED]/node_modules/raw-body/index.js:224:16)\n    at done (/home/centos/[REDACTED]/node_modules/raw-body/index.js:213:7)\n    at IncomingMessage.onEnd (/home/centos/[REDACTED]/node_modules/raw-body/index.js:273:7)\n    at IncomingMessage.emit (events.js:180:13)\n    at IncomingMessage.emit (domain.js:421:20)\n    at endReadableNT (_stream_readable.js:1101:12)\n    at args.(anonymous function) (/home/centos/.npm/versions/node/v9.9.0/lib/node_modules/pm2/node_modules/event-loop-inspector/index.js:138:29)\n    at process._tickCallback (internal/process/next_tick.js:114:19)",
    "expose": true,
    "statusCode": 400,
    "status": 400,
    "body": "{
      \"id\": \"1\",
      \"name\": \"<img src=\\\\\\\"data:image/bmp;base64,Qk1+AA\\\\\\\">[REDACTED] Bay Plaza\",
      \"city\": \"[REDACTED]\",
      \"country\": \"[REDACTED]\",
      \"region\": \"[REDACTED]\",
      \"vendor_id\": \"07f9aa4b-570a-46b5-b734-253a0af97d8d\",
      \"building\": \"\",
      \"floor\": \"\",
      \"room_no\": \"\",
      \"seats_no\": 1,
      \"seat_arrangement\": 1,
      \"has_whiteboard\": false,
      \"has_pcs\": false,
      \"has_projector\": false,
      \"touchpoint_email\": \"fsdfsdfsdf\"
    }\",
    \"type\": \"entity.parse.failed\"
  }
}
```

### Remediation Guidance:

Ensure custom error handling for all possible errors.

## 2.14 Frameable response (Clickjacking)

Ref ID: [REDACTED]###-

1-14

If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to

unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defences against cross-site request forgery, and may result in unauthorized actions.

Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defence is normally ineffective and can usually be circumvented by a skilled attacker.

#### **Vulnerability Details:**

<b>Affects:</b>	https://[REDACTED]/authentication/redirect;login=true
<b>Parameter(s)</b>	Header
<b>Attack Vectors</b>	Page framed
<b>References:</b>	<a href="https://cwe.mitre.org/data/definitions/693.html">https://cwe.mitre.org/data/definitions/693.html</a>

#### **Remediation Guidance:**

To effectively prevent framing attacks, the application should return a response header with the name X-FrameOptions and the value DENY to prevent framing altogether, or the value SAMEORIGIN to allow framing only by pages on the same origin as the response itself. Note that the SAMEORIGIN header can be partially bypassed if the application itself can be made to frame untrusted websites.

## 3 Appendices

### 3.1 Penetration Testing Methodologies

#### 3.1.1 Web/API Application Assessment

Web application assessments can be performed either remotely or on site, depending on the exposure of the application. The purpose of the assessment is to identify any vulnerabilities that can be exploited in order to attack the system or other users, bypass controls, escalate privileges, or extract sensitive data.

During the assessment, the consultants will use proven non-invasive testing techniques to quickly identify any weaknesses. The application is viewed and manipulated from several perspectives, including with no credentials, user credentials, and privileged user credentials.

The primary areas of concern in web application security are authentication bypass, injection, account traversal, privilege escalation, and data extraction.

Our methodology covers all of the OWASP Top 10 web application security risks and more.

---

Ref	Risk	Description
<b>A1</b>	Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
<b>A2</b>	Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other

---

implementation flaws to assume other users' identities temporarily or permanently.

**A3 Sensitive Data Exposure**

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit.

**A4 XML External Entities (XXE)** Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

**A5 Broken Access Control**

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

**A6 Security Misconfiguration** Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

**A7 Cross-Site Scripting (XSS)**

XSS flaws occur whenever an application includes untrusted data in a web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

**A8 Insecure Deserialization** Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

**A9 Using Known Vulnerable Components**

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.

**A10 Insufficient Logging and Monitoring**

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to other systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach



is over 200 days, typically detected by external parties rather than internal processes or monitoring.

### 3.1.2 External Infrastructure Assessment

An external infrastructure assessment checks for the vulnerabilities on which a majority of attacks are based. Infrastructure layer vulnerabilities are usually introduced via misconfiguration or an insufficient patching process.

The assessment is divided into four distinct phases: profiling, discovery, assessment, and exploitation.

Profiling of the corporate Internet-facing infrastructure using non-invasive techniques including OSINT frameworks, but not limited to.

PenTest-Hub engineers use a variety of scanning tools and techniques to locate live hosts and services within the target IP range and perform a comprehensive assessment against all IP addresses in scope:

- UDP and TCP port scanning – commonly done using standard port-scanning tools.
- Operating system fingerprinting.
- Service identification – service identification tools are used to analyse all live systems.
- User enumeration – dependent on what services are offered. - Network mapping – Hping, traceroute, IP fingerprinting.

Once the automated discovery is completed, the results will be investigated in a manual test to identify possible attack vectors. Manual assessment of all live hosts and exposed services focuses on:

- Host and service configuration – misconfigurations and poor build processes can leave insecure services available. These often allow a trivial route to achieve system compromise.
- Patching vulnerabilities – lack of a stringent patching strategy can leave hosts vulnerable; efforts will be made to locate out-of-date services and operating- system-wide missing patches.
- Use of insecure credentials or protocols such as Telnet and FTP may increase the risk of compromise. Any use of these protocols will be highlighted. Default and easy-to-guess passwords will be attempted.

All exploitation is done in strict accordance with agreed rules of engagement. It should be noted that exploitation is highly dependent on circumstances. Once exploits have succeeded, we use any access and privileges gained to attempt to escalate access rights to the highest level possible. Detailed records are kept of all data recovered and copies are taken before changes are made to any files. All exploits are risk- assessed to minimize disruption to live systems.

### 3.1.3 Mobile Application Assessment

Mobile applications, and the devices upon which they run, have quickly become a core part of everyday technology. With a surge in mobile application development, and developers under time pressure to provide new functionality, attacks and breaches have dramatically increased.

The purpose of Mobile application assessments is to identify any vulnerabilities that can be exploited in order to attack the system or other users, bypass controls, escalate privileges, or extract sensitive data.

The primary areas of concern in mobile application security are weak server-side controls, lack of binary protections, insecure data storage and insufficient transport layer protection.

Our methodology covers all of the OWASP Top 10 mobile security risks and more.

Ref	Risk	Description
<b>M1</b>	Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system.
<b>M2</b>	Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.
<b>M3</b>	Insecure Communication	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.
<b>M4</b>	Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: <ul style="list-style-type: none"><li>- Failing to identify the user at all when that should be required</li><li>- Failure to maintain the user's identity when it is required</li><li>- Weaknesses in session management</li></ul>
<b>M5</b>	Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3.
<b>M6</b>	Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.).
<b>M7</b>	Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other codelevel mistakes where the solution is to rewrite some code that's running on the mobile device.
<b>M8</b>	Code Tampering	This category covers binary patching, local resource modification,

method hooking, method swizzling, and dynamic memory modification.

Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.

#### **M9** Reverse Engineering

This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.

**M10** Extraneous Functionality Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.