

---

# OWASP Juice-Shop

## Penetration Test Report

---

Version	1.0
Author	Mohamed Mostafa, 5000284 Youssef Mostafa, 2000653
Issue Date	Thursday, December 29 <sup>th</sup> , 2022

# Table of Contents

## 1.0 OWASP Juice-Shop Penetration Test Report 2

1.1 Introduction 2

1.2 Scope 2

## 2.0 Executive Summary 3

2.1 - Recommendations 3

## 3.0 Risk Assessment ..... 4

3.1 - Likelihood 4

3.2 - Impact 4

## 4.0 Findings Summary ..... 5

## 5.0 Vulnerability and Remediation Report 6

## 6.0 Information gathering ..... 7

## 7.0 High Findings..... 8

7.1 - Login Admin Vulnerability SQL Injection

7.2 - Admin Section

7.3 - Confidential Document

7.4 - Login Bender

7.5 - Admin Registration

7.6 - Password Strength

7.7 - Poison Null Byte

7.8 - Deluxe Fraud

7.9 - Payback Time

## 8.0 Medium Findings.....

8.1 - Dom Based XSS Vulnerability

8.2 - Forged Feedback

8.3 - Vulnerable Library

8.4 - Ephemeral Accountant

## 8.5 - Exposed Metrics

### **9.0 Low Findings**.....45

#### 9.1 - Reflected XSS Vulnerability

#### 9.2 - Meta Geo Stalking

#### 9.3 - Visual Geo Stalking

#### 9.4 - Repetitive Registration

#### 9.5 - CAPTCHA Bypass

#### 9.6 - 0-star rating

# 1.0 OWASP Juice-Shop Penetration Test Report

## 1.1 Introduction

Subject of this document is a summary of penetration tests performed against web applications owned by OWASP company. Test was conducted according to rules of engagement defined and approved at the beginning by both parties – customer and contractor. Black-box pen-testing assignment was requested.

Black-box penetration test classification means that penetration tester has no internal knowledge about target system architecture. He/she can use information commonly available on the Internet. More data can be collected during the reconnaissance phase based on observation of target system behavior. Black-box penetration test results give overview of vulnerabilities exploitable from outside the company network. It shows how unauthenticated actors can take advantage of weaknesses existing in tested web applications.

Time frame defined:

Penetration test start: Wed, December 14<sup>th</sup> 08:30:00

Penetration test end: Wed, December 14<sup>th</sup> 17:00:00

## 1.2 Scope

To perform a Black Box Web Application Penetration Test against the web applications of the organization named OWASP Juice-Shop.

This is what the client organization defined as scope of the tests:

- ☒ Dedicated Web Server: localhost
- ☒ Domain: http://localhost:3000
- ☒ Subdomains: all subdomains

## 2.0 – EXECUTIVE SUMMARY

The conducted penetration test of the OWASP Juice-Shop web application identified a total of 20 vulnerabilities, including 9 high severity, 5 medium severity, and 6 low severity issues. The vulnerabilities were primarily due to outdated patches and poor security configurations, and we were able to gain access to multiple vulnerabilities during the testing. One of the most significant vulnerabilities identified was Admin Registration, which allows an attacker to create a user with admin privileges. In addition, we were able to successfully various sensitive data, which could allow an attacker to take over some accounts, and Forge data, which allows an attacker to purchase products with invalid amounts. Other vulnerabilities identified included SQL Injections and XSS, which have a huge impact on the website.

Overall, the penetration test identified a number of vulnerabilities that need to be addressed in order to improve the security of the OWASP Juice-Shop web application.

### 2.1 – Recommendations

To remediate these vulnerabilities, we recommend implementing the following measures: Apply patches to address known vulnerabilities Implement input validation to ensure that only allowed characters and formats are accepted in form submissions or query string parameters Use output encoding and sanitization techniques to prevent malicious code from being executed in the page Use a web application firewall (WAF) to monitor and block malicious traffic Regularly update software and applications to ensure that they are protected against known vulnerabilities. In addition to the identified vulnerabilities, the penetration test also identified a number of best practices and security measures that should be implemented during the testing. These measures should be effective in protecting the web application from potential attacks.

Implement input validation to ensure that only allowed characters and formats are accepted in form submissions or query string parameters, in order to prevent vulnerabilities. Use output encoding and sanitization techniques to prevent malicious code from being executed in the page, in order to prevent vulnerabilities. Use parameterized queries and stored procedures to reduce the risk of SQL injection attacks. Implement a web application firewall (WAF) to monitor and block malicious traffic, in order to prevent vulnerabilities. Regularly update software and applications to ensure that they are protected against known vulnerabilities, in order to prevent vulnerabilities. Remove EXIF and unnecessary data from images or files before posting them publicly, in order to prevent vulnerabilities. Implement password security measures, such as requiring strong passwords and enforcing password expiration, in order to prevent vulnerabilities. Validate user input and implement appropriate access controls to prevent unauthorized access to sensitive data, in order to prevent vulnerabilities.

Implement two-factor authentication (2FA) or multi-factor authentication (MFA) to add an extra layer of security to user accounts and prevent unauthorized access. Implement network segmentation to limit access to sensitive data and systems to only authorized users and devices. Use encrypted connections and protocols, such as HTTPS and SFTP, to protect data in transit. Use security monitoring tools and services to continuously monitor the network and identify potential security threats. Conduct regular security assessments and penetration tests to identify and address vulnerabilities on an ongoing basis. Develop and maintain an incident response plan to effectively handle security incidents and minimize the impact on the organization. Implement security awareness training programs to educate employees on how to recognize and prevent potential security threats. Use security controls and measures, such as firewalls, intrusion detection and prevention systems (IDPS), and antivirus software, to protect against external threats. Regularly update and patch software and systems to ensure that they are protected against known vulnerabilities.

## 3.0 – RISK ASSESSMENT

### 3.1 Likelihood

The likelihood is a measurement of the capacity to carry out an attack. The factor will be the difficulty or skill required to carry out the attack.

Risk	Description
Critical	An attacker is near-certain to carry out the threat event
High	An untrained user could exploit the vulnerability. The vulnerability is obvious or easily accessed
Medium	The vulnerability required some hacking knowledge to carry out the attack
Low	The vulnerability required significant time, skill , access and other resource to carry out the attack

### 3.2 Impact

The impact is a measurement of the adverse effect carrying out an attack would have on the organization.

Risk	Description
Critical	An attack would cause catastrophic or severe effect on operation, asset or other organization
High	An attack would severely degrade mission capability. The attack may result in damage to asset( data exposure)
Medium	An attack would degrade the mission capability. An attack would allow for primary function to application to resume, but at reduced effectiveness
Low	An attack would degrade mission capability in a limited capacity. The attack may result in marginal damage to assets

## 4.0 -- FINDINGS SUMMARY

Findings	Likelihood	Impact	Rating	Status
Login Admin Vulnerability SQL Injection	Medium	Critical	High	Open
Admin Section	High	Critical	High	Open
Confidential Document	High	High	High	Open
Login Bender	Medium	Critical	High	Open
Admin Registration	Low	Critical	High	Open
Password Strength	Low	Critical	High	Open
Poison Null Byte	Low	Critical	High	Open
Deluxe Fraud	Medium	Critical	High	Open
Dom Based XSS Vulnerability	Medium	Medium	Medium	Open
Forged Feedback	Medium	Low	Medium	Open
Vulnerable Library	Low	High	Medium	Open
Ephemeral Accountant	Low	High	Medium	Open
Exposed Metrics	Critical	Low	Medium	Open
Reflected XSS Vulnerability	Medium	Low	Low	Open
Meta Geo Stalking	Low	Low	Low	Open
Visual Geo Stalking	Low	Low	Low	Open
Repetitive Registration	Medium	Low	Low	Open
CAPTCHA Bypass	Medium	Low	Low	Open
0-star rating	Medium	Low	Low	Open
Payback Time	Medium	Critical	High	Open



## **5.0 – VULNERABILITY & REMEDIATION REPORT**

Penetration test finding classification, description and recommendations mentioned in the report are taken mostly from OWASP TOP 10 project documentation available on site:

[https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project).

The OWASP TOP 10 is a list of definitions of web application vulnerabilities that pose the most significant security risks to organizations when exploited.

## 6.0 – INFORMATION GATHERING

The information gathering portion of a penetration test focuses on identifying the scope of the penetration test. I started the pen-test with finding all subdomains.

I have used multiple tools to make sure that I haven't missed any domain.

Tools Used:

- 1) WFUZZ
- 2) Virustotal
- 3) Sublist3r
- 4) Burp Suite
- 5) Synk
- 6) ExIfTool
- 7) FoxyProxy
- 8) GIMP

## 7.0 High Findings

---

### 7.1 Title: Login Admin Vulnerability SQL Injection

**Rating:** High

**URL:** http://127.0.0.1:3000/#/login

**Description:** The Login Admin vulnerability allows an attacker to gain unauthorized access to the administrator account.

**Proof of Concept:**

Step 1: Open the OWASP Juice-Shop application in a web browser.

Step 2: Navigate to the login page by clicking on the "Sign in" button in the top right corner of the page.

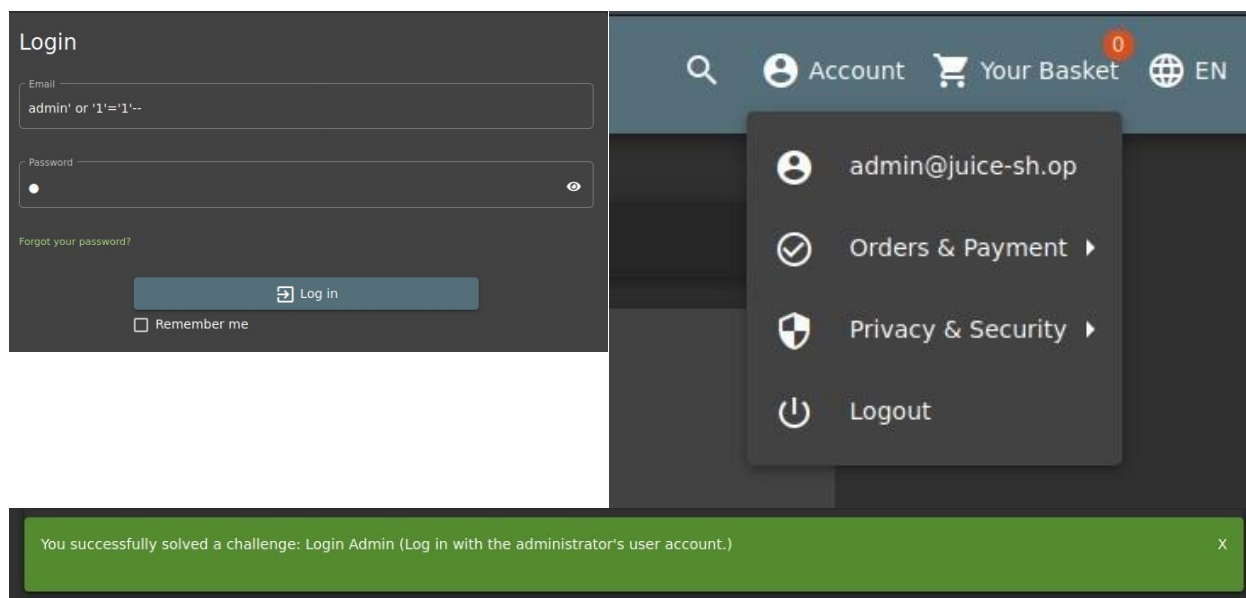
Step 3: In the username field, enter "admin' OR 1 = 1 --" and in the password field, enter any password.

Step 4: Click on the "Sign in" button.

This vulnerability can be exploited by an attacker to gain unauthorized access to the administrator account and potentially gain access to sensitive information or perform malicious actions within the application.

To mitigate the risk of SQL injection attacks, there are a few best practices that you can follow:

- Use parameterized queries: Parameterized queries are a type of SQL statement that allows you to pass in parameters, rather than embedding them directly into the statement. This can help to prevent attackers from injecting malicious code into your queries.
- Validate user input: Make sure to validate all user input to ensure that it is appropriate and does not contain any malicious code.
- Use stored procedures: Stored procedures are pre-compiled SQL statements that can be executed on the server. They can help to reduce the risk of SQL injection attacks because the SQL code is not visible to the user.
- Use a web application firewall (WAF): A WAF is a security solution that monitors and analyzes incoming traffic to a web application and blocks malicious traffic based on a set of rules. This can help to prevent SQL injection attacks by blocking traffic that is attempting to inject malicious code into your application.
- Regularly update your software: Make sure to keep your software and applications up to date with the latest security patches. This can help to prevent vulnerabilities that attackers can exploit.



## 7.2 Title: Admin Section

**Rating:** High

**URL:** <http://127.0.0.1:3000/#/administration>

**Description:** This vulnerability allows an attacker to access the admin section of the website and control sensitive aspects of the website using the admin dashboard.

### Proof of Concept:

Step 1: Log in as an admin ([admin@juice-sh.op](mailto:admin@juice-sh.op) or any other admin account with admin roles). This can be done, for example, through section 7.1 of this document.

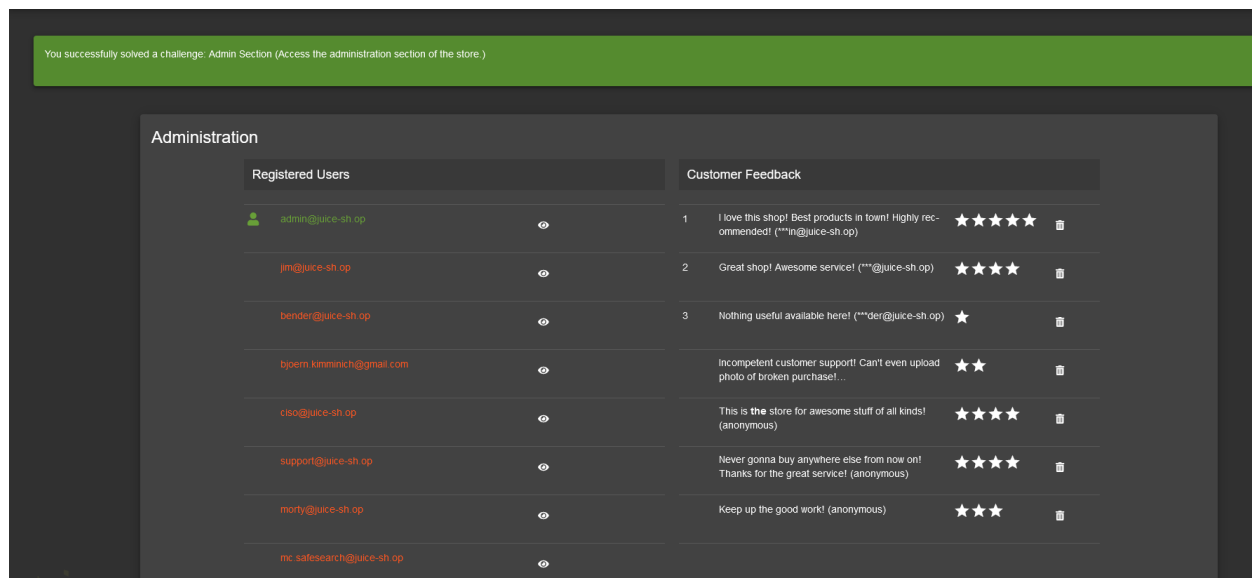
Step 2: Try famous admin directories that may lead to the admin dashboard.

Step 3: Notice that the URL entered above will send the attacker to the admin dashboard.

Step 4: The attacker can use the dashboard to see the registered users or read and delete customer Feedbacks

### Remediation Steps:

- Change the URL to something more secure such as using a base 64 encoded password.
- Enable multifactor authentication and use an IP whitelist



### 7.3 Title: Confidential Document

**Rating:** High

**URL:** http://127.0.0.1:3000/ftp

**Description:** This vulnerability allows an attacker to access the ftp folder and retrieve sensitive information.

#### Proof of Concept:

Step 1: Use WFUZZ for enumerating the website

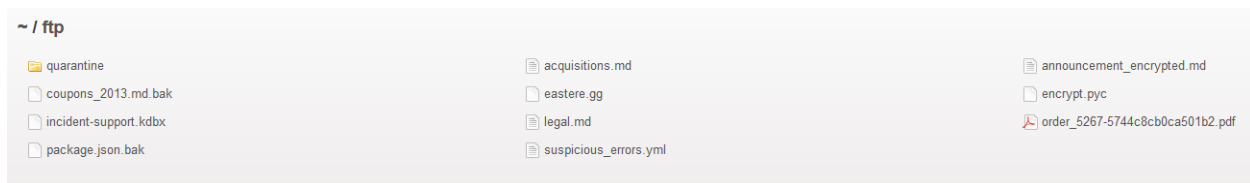
Step 2: An ftp directory will be found

Step 3: Access the ftp directory through the URL provided above

Step 4: access confidential documents with sensitive information. E.g. acquisitions.md.

#### Remediation Steps:

- In case an ftp folder must be included in the site, Access Controls such as IP whitelists and passwords are very important.



## 7.4 Title: Login Bender

**Rating:** High

**URL:** <http://127.0.0.1:3000/#/login>

**Description:** This vulnerability allows an attacker to access the account of a User, in this case, Bender's account

### Proof of Concept:

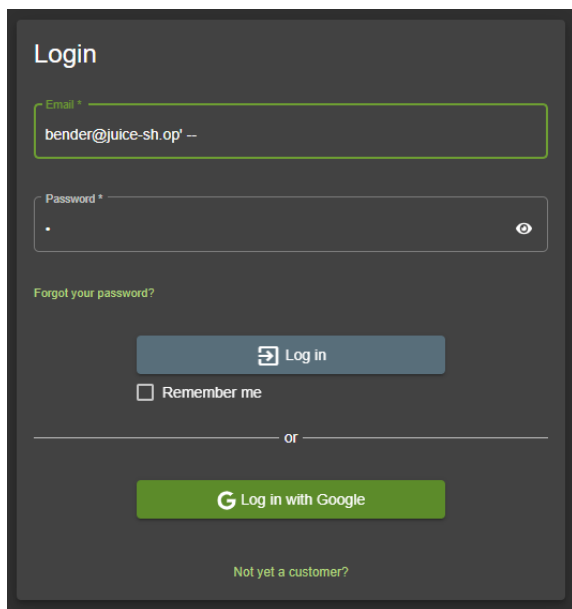
Step 1: Head to the admin section through sections 7.1 and 7.2 to retrieve Bender's email

Step 2: add a single quote after the email followed by a double dash "--" to comment what's after this injection

Step 3: enter any password and press login and the attacker should have the account at his disposal.

### Remediation Steps:

- Use parameterized queries: Parameterized queries are a type of SQL statement that allows you to pass in parameters, rather than embedding them directly into the statement. This can help to prevent attackers from injecting malicious code into your queries.
- Validate user input: Make sure to validate all user input to ensure that it is appropriate and does not contain any malicious code.
- Use stored procedures: Stored procedures are pre-compiled SQL statements that can be executed on the server. They can help to reduce the risk of SQL injection attacks because the SQL code is not visible to the user.
- Use a web application firewall (WAF): A WAF is a security solution that monitors and analyzes incoming traffic to a web application and blocks malicious traffic based on a set of rules. This can help to prevent SQL injection attacks by blocking traffic that is attempting to inject malicious code into your application.
- Regularly update your software: Make sure to keep your software and applications up to date with the latest security patches. This can help to prevent vulnerabilities that attackers can exploit.



You successfully solved a challenge: Login Bender (Log in with Bender's user account.)

## 7.5 Title: Admin Registration

**Rating:** High

**URL:** <http://127.0.0.1:3000/#/login>

**Description:** This vulnerability allows an attacker to register an admin account.

### Proof of Concept:

Step 1: Head to the registration section and create a regular user.

Step 2: Before sending out the post request, open Burp Suite and allow Interceptions

Step 3: Send the packet to Burp and examine the JSON data.

Step 4: Because we had data for the user authentication details that were gathered when the vulnerability in section 7.2, we know that we can add a “role” tag that can have a value of “admin” such that this registered user can be an admin. All that needs to be done is adding the role tag after the “passwordRepeat” tag and the packet can be forwarded. The packet should be accepted and a new admin user is created.

### Remediation Steps:

- [https://cheatsheetseries.owasp.org/cheatsheets/Mass\\_Assignment\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html)

```
{
  "email": "test4@test.com",
  "password": "12345",
  "passwordRepeat": "12345",
  "role": "admin",
  "securityQuestion": {
    "id": 12,
    "question": "Your favorite movie?",
    "createdAt": "2022-12-30T11:03:49.962Z",
    "updatedAt": "2022-12-30T11:03:49.962Z"
  },
  "securityAnswer": "777"
}
```

You successfully solved a challenge: Admin Registration (Register as a user with administrator privileges.)

## 7.6 Title: Password Strength

**Rating:** High

**URL:** <http://127.0.0.1:3000/#/register>

**Description:** This vulnerability allows an attacker to access the account of the admin, “admin@juice-sh.op”, WITHOUT using SQL Injections or changing the admin password.

### Proof of Concept:

Step 1: Since we already know the email of the admin account by executing section 7.5 and entering the administration section in section 7.2, we can get the email for the original admin

Step 2: Go to the login page and enter the admin email and password.

Step 3: Before sending the data to the server, open Burp Suite and allow interceptions.

Step 4: Send the request to Burp and examine the email and password.

Step 5: Get a suitable wordlist that can contain admin passwords such as “rockyou.txt” and execute a sniper attack and wait for the status of a password to be 200.



Step 6: Copy and paste the password with status 200 in the value of the “password” tag and forward the packet.

## Remediation Steps:

- Take password security seriously
- Use unique passwords for each account

The screenshot displays the Burp Suite interface. At the top, there are tabs for Dashboard, Target, Proxy, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, and Project options. Below these, there are tabs for Target, Positions, Payloads, and Options. The 'Payload Positions' tab is active, showing a configuration for an attack type 'Sniper'. The attack is a POST request to /rest/user/login with a JSON payload: {"email": "admin@juice-sh.op", "password": "5test5"}. The status is 200, and the response length is 1162. A comment indicates 'Contains a JWT'. Below the attack configuration, there is a table with columns: Request, Payload, Status, Error, Timeout, Length, and Comment. The table shows 14 requests, with the 8th request (admin123) highlighted in blue, indicating it was the successful login attempt. At the bottom, a green banner states: 'You successfully solved a challenge: Password Strength (Log in with the administrator's user credentials without previously changing them or applying SQL Injection.)'

Request	Payload	Status	Error	Timeout	Length	Comment
0		401			362	
1	aaa	401			362	
2	abc123	401			362	
3	acc	401			362	
4	access	401			362	
5	adffxc	401			362	
6	adm	401			362	
7	admin	401			362	
8	admin123	200			1162	Contains a JWT
9	admin2	401			362	
10	admin_1	401			362	
11	administrator	401			362	
12	adminstat	401			362	
13	adminstrator	401			362	
14	admintrd	401			362	

## 7.7 Title: Poison Null Byte

**Rating:** High

**URL:** http://127.0.0.1:3000/ftp

**Description:** This vulnerability allows an attacker to access files that are not .md or .pdf in the ftp folder

### Proof of Concept:

Step 1: Access the ftp folder and select a file e.g. "package.json.bak"

Step 2: Notice that an error is given that only reading .md or .pdf files is allowed.

Step 3: To bypass this, resend this packet to Burp Suite and change the path of the packet to URL encode a comment which can bypass the checker as seen in line 1 of the provided Burp intercept image

### Step 4: Forward the packet and it should get downloaded

### Remediation Steps:

- [https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)

```
Pretty Raw Hex
```

```
1 GET /ftp/package.json.bak%2500.md HTTP/1.1  
2 Host: localhost:3000  
3 sec-ch-ua: "Not?A Brand";v="8", "Chromium";v="108"  
4 sec-ch-ua-mobile: ?0  
5 sec-ch-ua-platform: "Windows"  
6 Upgrade-Insecure-Requests: 1  
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,  
9 Sec-Fetch-Site: same-origin  
0 Sec-Fetch-Mode: navigate  
1 Sec-Fetch-User: ?1  
2 Sec-Fetch-Dest: document  
3 Referer: http://localhost:3000/ftp/  
4 Accept-Encoding: gzip, deflate  
5 Accept-Language: en-US,en;q=0.9  
6 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOnBxMDUwYmNjlkZjE4YjYUMCIsInRlbnQGaWUiOih2LGlphbiIsImRlbHV4ZWVudCI6JCJob3RwU2VjcmlvdjoiIiwiaXBNEY3RpdmcUiOnRydWUsImNyZWFOZSbmH0siOmldcCEMTYzMjQWNjY1OCwiZXhwIjoxNDIONjU4fQ.CgAv9tAcplP0Sk2U49OSUWPqTn_hns2LfNgEvXk; continueCookieName=  
7 If-None-Match: W/"1174-1850f9c539d"  
8 If-Modified-Since: Wed, 14 Dec 2022 07:49:41 GMT  
9 Connection: close  
0
```

You successfully solved a challenge: Poison Null Byte (Bypass a security control with a Poison Null Byte to access a file not meant for your eyes.)

### 7.8 Title: Deluxe Fraud

**Rating:** High

**URL:** <http://127.0.0.1:3000/#/deluxe-membership>

**Description:** This vulnerability allows an attacker to pay for their membership or any payment without spending any money

## Proof of Concept:

Step 1: Login into an account and add money to the wallet by deducting the money from a card that the attacker owns.

Step 2: Head to the URL provided and apply for membership.

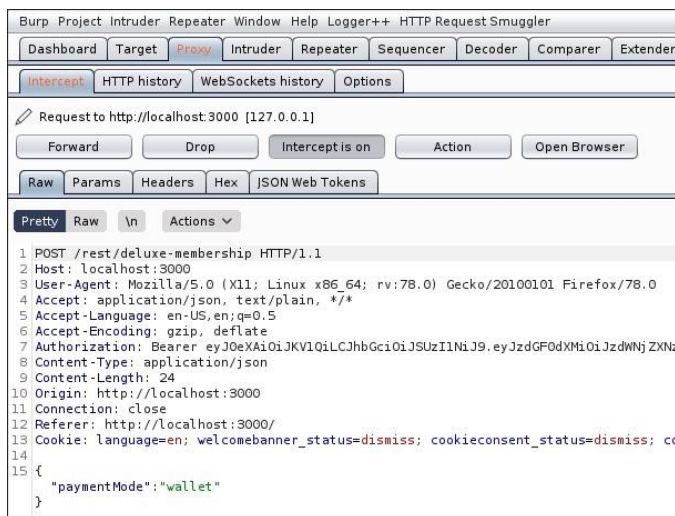
Step 3: Before pressing the pay with wallet button, open Burp Suite and allow interceptions.

Step 4: press the pay button and send to Burp.

Step 5: change the value of the “paymentMode” tag to “none” and forward the packet.

## Remediation Steps:

- Validate relevant information on the server side before issuing a deluxe token or changing a user's role.
- Verify that payment has been received before taking any action.
- Do not take action without verifying that a mode of payment has been selected.
- This type of flaw should be rare in the real world.



You successfully solved a challenge: Deluxe Fraud (Obtain a Deluxe Membership without paying for it.)

## 7.8 Title: Payback Time

**Rating:** High

**URL:** http://127.0.0.1:3000/#/basket

**Description:** This vulnerability allows an attacker to gain money by purchasing a negative amount of items

**Proof of Concept:**

Step 1: Head to the main page

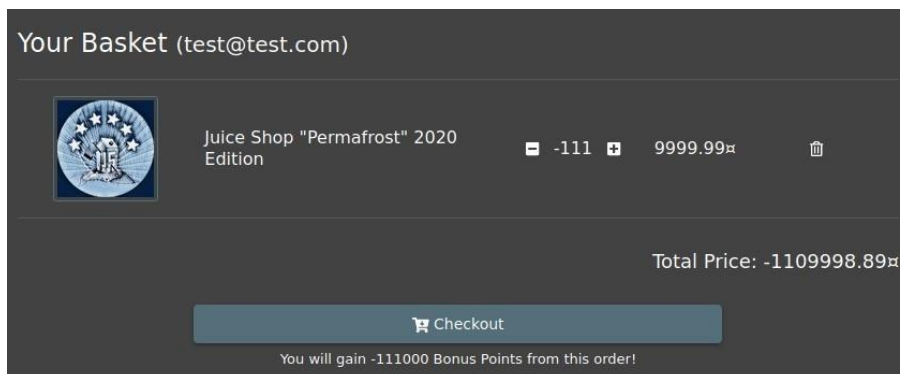
Step 2: Intercept the packets using Burp Suite when adding an item to the basket

Step 3: Change the quantity to a negative amount

Step 4: Proceed to pay for the item

**Remediation Steps:**

- [https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)



You successfully solved a challenge: Payback Time (Place an order that makes you rich.) X

## 8.0 Medium Findings

---

### 8.1 Title: Dom Based XSS Vulnerability

**Rating:** Medium

**URL:** <http://127.0.0.1:3000/#/search?q=>

**Description:** This vulnerability allows an attacker to send malicious payloads in the search-bar of the website. This can lead to a website defacement.

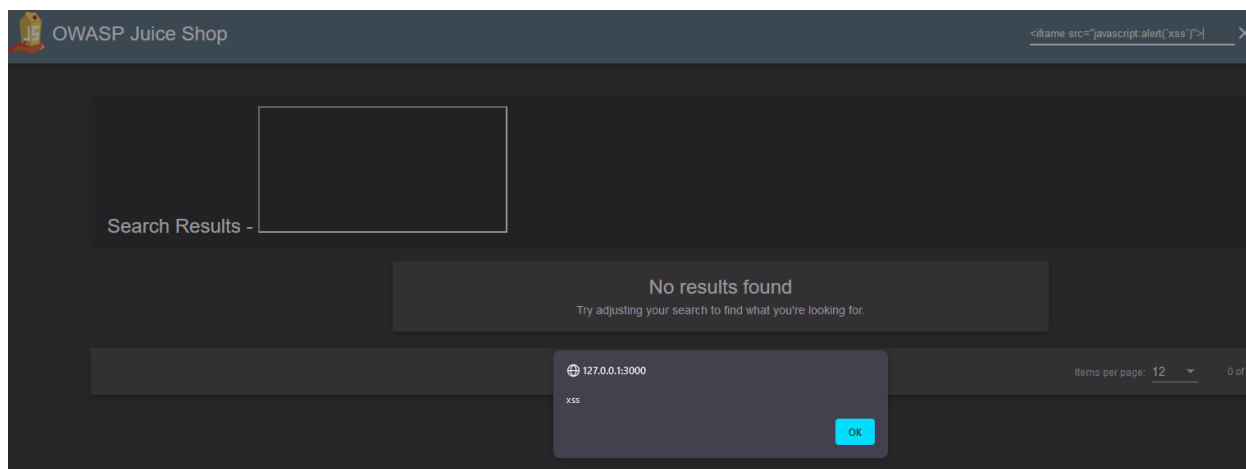
**Proof of Concept:**

Step 1: Try to send “<iframe src=“javascript:alert(`xss`)”>” through the search bar (without the double quotes at the very beginning and at the very end)

Step 2: since step 1 worked, try sending “<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto\_play=true&hide\_related=false&show\_comments=true&show\_user=true&show\_reposts=false&show\_teaser=true"></iframe>” through the search bar (without the double quotes at the very beginning and at the very end) to deface the website.

**Remediation Steps:**

- Validate and sanitize user input: Make sure to validate and sanitize all user input that is used in the DOM to prevent attackers from injecting malicious code.
- Use content security policies (CSP): CSP is a security feature that helps to mitigate XSS attacks by specifying which sources are allowed to load resources on a webpage.
- Escape special characters: When rendering user input in the DOM, make sure to escape any special characters such as <, >, and & to prevent attackers from injecting malicious code.
- Use a library or framework: Use a library or framework that automatically escapes user input and prevents XSS attacks.
- Regularly test for vulnerabilities: Regularly test your application for vulnerabilities, including XSS vulnerabilities, to ensure that it is secure.



## 8.2 Title: Forged Feedback

**Rating:** Medium

**URL:** <http://127.0.0.1:3000/#/contact>

**Description:** This vulnerability allows an attacker to send customer-feedbacks using other people's ids

### Proof of Concept:

Step 1: Fill the form, send the request to Burp Suite and examine the data.

Step 2: Change the Value of the "User Id" tag to any other existing User Id and then forward the request.

### Remediation Steps:

- Match user's cookie data to JSON fields to prevent unauthorized submissions
- Enforce text size limits on the server side to prevent large texts from being submitted into fields not designed to handle them

```
{
  "UserId":3,
  "captchaId":3,
  "captcha":"420",
  "comment":"wallah ma a3ref (***) juice-sh.op)",
  "rating":1
}
```

You successfully solved a challenge: Forged Feedback (Post some feedback in another user's name.)

### 8.3 Title: Vulnerable Library

**Rating:** Medium

**URL:** <http://127.0.0.1:3000/#/contact>

**Description:** This vulnerability allows a user to send feedback on vulnerable libraries that the website uses.

#### Proof of Concept:

Step 1: Execute section 7.7 to get the dependencies.

Step 2: Use synk to check each library and get a score of 1-10 in which a 1 is very secure and a 10 is a severe. Getting a score of 8+ means that the library is unsecure.

Step 3: After noting each library's score, submit a feedback on the vulnerable libraries.

#### Remediation Steps:

- [https://cheatsheetseries.owasp.org/cheatsheets/Vulnerability\\_Disclosure\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Vulnerability_Disclosure_Cheat_Sheet.html)

Vulnerability DB > npm > marsdb

### Arbitrary Code Injection

Affecting marsdb package, ALL versions

[Report new vulnerabilities](#)

---

Do your applications use this vulnerable package?

[Test your applications](#)

#### Overview

marsdb is a MarsDB is a lightweight client-side database.

Affected versions of this package are vulnerable to Arbitrary Code Injection. In the DocumentMatcher class, selectors on \$where clauses are passed to a Function constructor unsanitized. This allows attackers to run arbitrary commands in the system when the function is executed.

#### Remediation

There is no fixed version for marsdb.

#### References

- NPM Security Advisory

CVSS SCORE

**9.8** HIGH SEVERITY

ATTACK VECTOR	ATTACK COMPLEXITY
Network	Low
PRIVILEGES REQUIRED	USER INTERACTION
None	None
SCOPE	CONFIDENTIALITY
Unchanged	High
INTEGRITY	AVAILABILITY
High	High

**Customer Feedback**

Author  
anonymous

Comment  
marsdb~0.6, grunt ~1.0, expressjwt 0.1.3, js-yaml 3.10, sequelize ~4, sanitize-html 1.4.2

Max. 160 characters 89/160

Rating ☐

CAPTCHA: What is 5\*4\*8 ?

Result  
160

You successfully solved a challenge: Vulnerable Library (Inform the shop about a vulnerable library it is using. (Mention the exact library name and version in your comment)) X

#### 8.4 Title: Ephemeral Accountant

**Rating:** Medium

**URL:** <http://127.0.0.1:3000/#/login>

**Description:** This vulnerability allows an attacker to login with an account that was never registered.

#### Proof of Concept:

Step 1: Write the code in the picture in the email tab

Step 2: Enter any password and press login

#### Remediation Steps:

- [https://cheatsheetseries.owasp.org/cheatsheets/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)

```
' UNION SELECT * FROM (SELECT 1000 as 'id', '' as 'username', 'acc0unt4nt@juice-sh.op' as 'email', 'asdfasdf' as 'password', 'accounting' as 'role', '' as 'deluxeToken', '127.0.0.1' as 'lastLoginIp', 'default.svg' as 'profileImage', '' as 'totpSecret', 1 as 'isActive', '2020-08-30 11:12:13.456 +00:00' as 'createdAt', '2020-08-30 11:12:13.456 +00:00' as 'updatedAt', null as 'deletedAt')--
```

You successfully solved a challenge: Ephemeral Accountant (Log in with the (non-existing) accountant acc0unt4nt@juice-sh.op without ever registering that user.) X



## 8.5 Title: Exposed Metrics

Rating: Medium

URL: <http://127.0.0.1:3000/metrics>

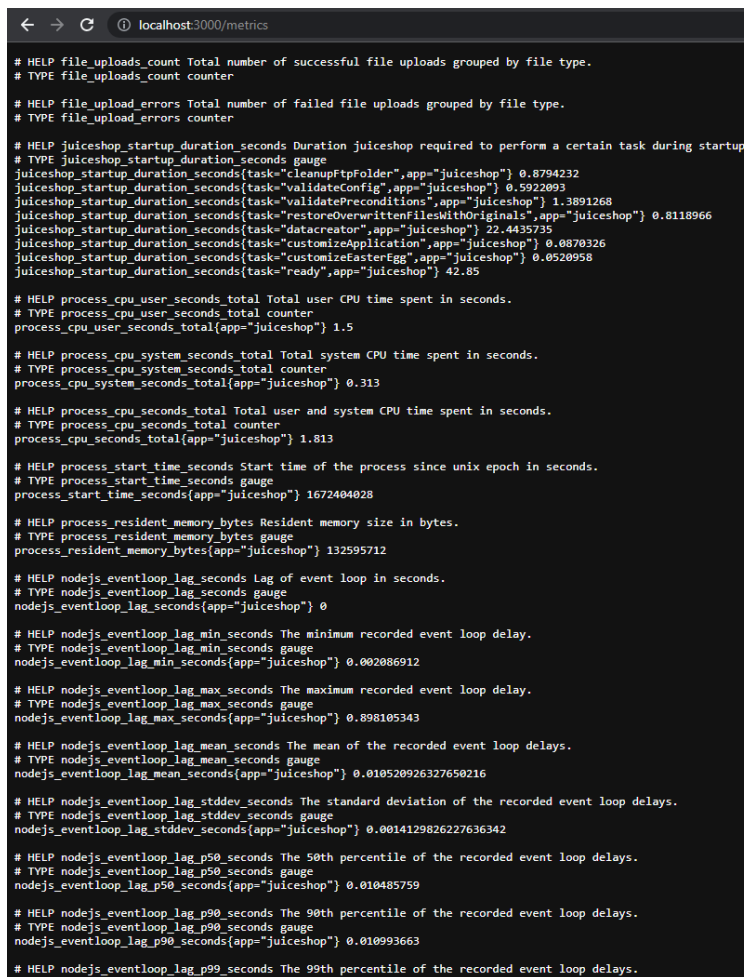
Description: This vulnerability allows an attacker to view the monitoring system of the website.

### Proof of Concept:

Step 1: Just by guessing the URL, the vulnerability worked.

### Remediation Steps:

- Change the link to a more secure URL
- Restrict access to whitelisted IP addresses
- Obfuscate the link by encoding a password and using it as the link (optional)
- Do not allow users to access use metrics



```
localhost:3000/metrics

# HELP file_uploads_count Total number of successful file uploads grouped by file type.
# TYPE file_uploads_count counter

# HELP file_upload_errors Total number of failed file uploads grouped by file type.
# TYPE file_upload_errors counter

# HELP juiceshop_startup_duration_seconds Duration juiceshop required to perform a certain task during startup
# TYPE juiceshop_startup_duration_seconds gauge
juiceshop_startup_duration_seconds{task="cleanupFtpFolder",app="juiceshop"} 0.8794232
juiceshop_startup_duration_seconds{task="validateConfig",app="juiceshop"} 0.5922893
juiceshop_startup_duration_seconds{task="validatePreconditions",app="juiceshop"} 1.3891268
juiceshop_startup_duration_seconds{task="restoreOverwrittenFilesWithOriginals",app="juiceshop"} 0.8118966
juiceshop_startup_duration_seconds{task="datacreator",app="juiceshop"} 22.4435735
juiceshop_startup_duration_seconds{task="customizeApplication",app="juiceshop"} 0.0870326
juiceshop_startup_duration_seconds{task="customizeEasterEgg",app="juiceshop"} 0.0520958
juiceshop_startup_duration_seconds{task="ready",app="juiceshop"} 42.85

# HELP process_cpu_user_seconds_total Total user CPU time spent in seconds.
# TYPE process_cpu_user_seconds_total counter
process_cpu_user_seconds_total{app="juiceshop"} 1.5

# HELP process_cpu_system_seconds_total Total system CPU time spent in seconds.
# TYPE process_cpu_system_seconds_total counter
process_cpu_system_seconds_total{app="juiceshop"} 0.313

# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total{app="juiceshop"} 1.813

# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds{app="juiceshop"} 1672404028

# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes{app="juiceshop"} 132595712

# HELP nodejs_eventloop_lag_seconds Lag of event loop in seconds.
# TYPE nodejs_eventloop_lag_seconds gauge
nodejs_eventloop_lag_seconds{app="juiceshop"} 0

# HELP nodejs_eventloop_lag_min_seconds The minimum recorded event loop delay.
# TYPE nodejs_eventloop_lag_min_seconds gauge
nodejs_eventloop_lag_min_seconds{app="juiceshop"} 0.002086912

# HELP nodejs_eventloop_lag_max_seconds The maximum recorded event loop delay.
# TYPE nodejs_eventloop_lag_max_seconds gauge
nodejs_eventloop_lag_max_seconds{app="juiceshop"} 0.898105343

# HELP nodejs_eventloop_lag_mean_seconds The mean of the recorded event loop delays.
# TYPE nodejs_eventloop_lag_mean_seconds gauge
nodejs_eventloop_lag_mean_seconds{app="juiceshop"} 0.010520926327650216

# HELP nodejs_eventloop_lag_stddev_seconds The standard deviation of the recorded event loop delays.
# TYPE nodejs_eventloop_lag_stddev_seconds gauge
nodejs_eventloop_lag_stddev_seconds{app="juiceshop"} 0.0014129826227636342

# HELP nodejs_eventloop_lag_p50_seconds The 50th percentile of the recorded event loop delays.
# TYPE nodejs_eventloop_lag_p50_seconds gauge
nodejs_eventloop_lag_p50_seconds{app="juiceshop"} 0.010485759

# HELP nodejs_eventloop_lag_p90_seconds The 90th percentile of the recorded event loop delays.
# TYPE nodejs_eventloop_lag_p90_seconds gauge
nodejs_eventloop_lag_p90_seconds{app="juiceshop"} 0.010993663

# HELP nodejs_eventloop_lag_p99_seconds The 99th percentile of the recorded event loop delays.
```

## 9.0 Low Findings

---

### 9.1 Title: Reflected XSS Vulnerability

**Rating:** Low

**URL:** http://127.0.0.1:3000/#/

**Description:** This vulnerability allows an attacker to insert malicious code by changing the order id in the Track Order section and for the result to be reflected after refreshing the page.

#### Proof of Concept:

Step 1: Make an order and pay for it.

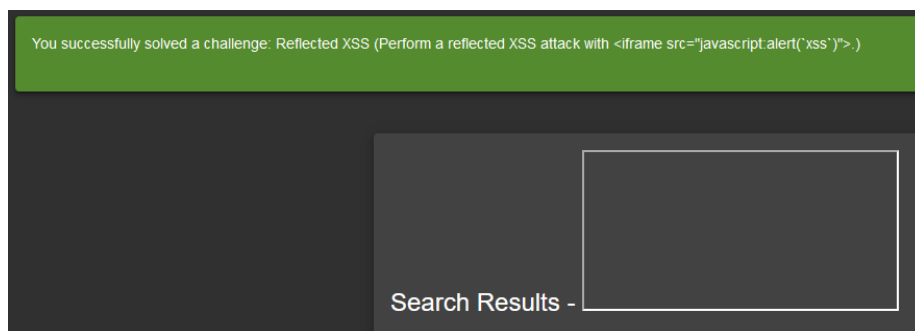
Step 2: Go to the Track Order section and select an order

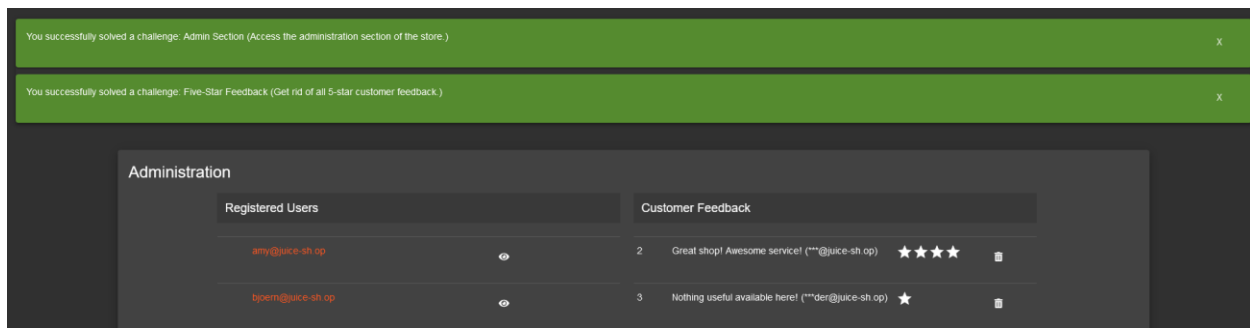
Step 3: Change the order id in the url with a payload such as `<iframe src="javascript:alert(`xss`) ">`

Step 4: Observe that the payload is reflected in the page and executes when the page is loaded

#### Remediation Steps:

- Implement input validation to ensure that only allowed characters and formats are accepted in form submissions or query string parameters
- Use output encoding and sanitization techniques to prevent malicious code from being executed in the page





## 9.2 Title: Meta Geo Stalking

**Rating:** Low

**URL:** <http://127.0.0.1:3000/#/forgot-password>

**Description:** This vulnerability requires an attacker to obtain an image sent by a user and read its metadata in order to find the answer to the user's security question to perform the "Forgot Password" function correctly: in this case, obtaining John's security question

### Proof of Concept:

Step 1: Head to the admin section through sections 7.1 and 7.2 to retrieve John's email.

Step 2: head to the "Forgot Password" section in the login form.

Step 3: paste John's email in the "Forgot Password" form to see the security question. The question is "What's your favorite place to go hiking?".

Step 4: To find the answer, access the photo-wall and find a pic submitted by John.

Step 5: Save the image to the system and use ExifTool to read the metadata.

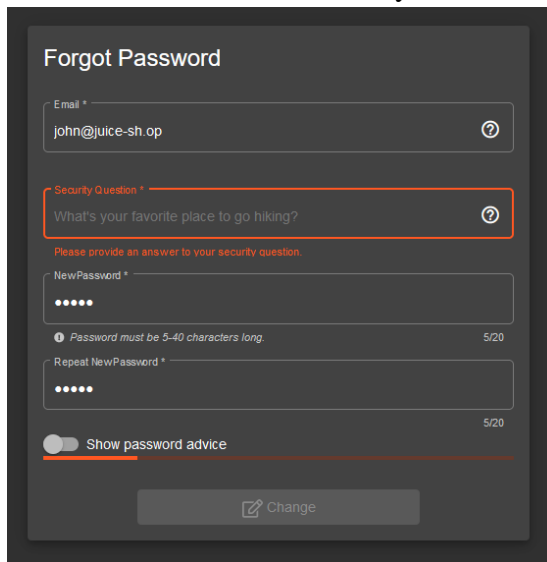
Step 6: The caption of the image is "I love going hiking here..." so the place must be the coordinates of where the picture was taken.

Step 7: Take the coordinates and paste them on google maps to find the location.

Step 8: Use Burp Suite repeater to attempt locations that suit the place. The answer in the end is "Daniel Boone National Forest"

## Remediation Steps:

- Remove EXIF and unnecessary data from images or files before posting them publicly



Forgot Password

Email \*  
john@juice-sh.op

Security Question \*  
What's your favorite place to go hiking?

Please provide an answer to your security question.

New Password \*  
.....

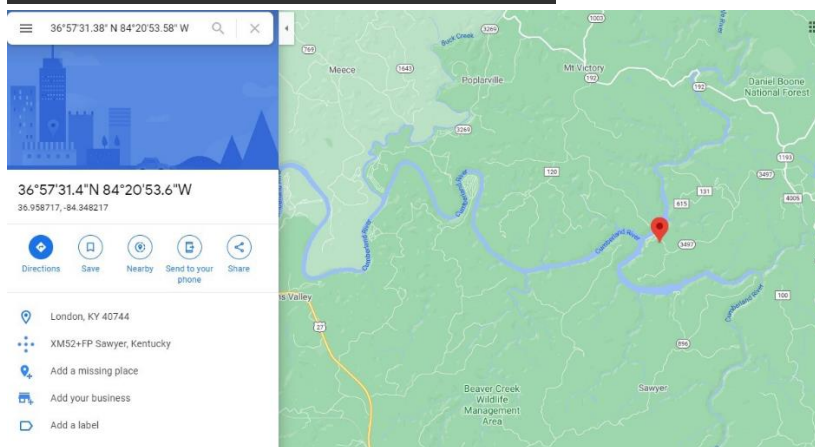
Password must be 5-40 characters long. 5/20

Repeat New Password \*  
.....

5/20

☐ Show password advice

Change



You successfully solved a challenge: Meta Geo Stalking (Determine the answer to John's security question by looking at an upload of him to the Photo Wall and use it to reset his password via the Forgot Password mechanism.)

### 9.3 Title: Visual Geo Stalking

Rating: Low

URL: <http://127.0.0.1:3000/#/forgot-password>

**Description:** This vulnerability requires an attacker to obtain an image sent by a user and check the visual details in the picture, in this case Emma's picture, in order to find the answer to her security question, "company you first work for as an adult".

#### Proof of Concept:

Step 1: Head to the admin section through sections 7.1 and 7.2 to retrieve Emma's email.

Step 2: head to the “Forgot Password” section in the login form.

Step 3: paste Emma’s email in the “Forgot Password” form to see the security question. The question is “Company you first work for as an adult”.

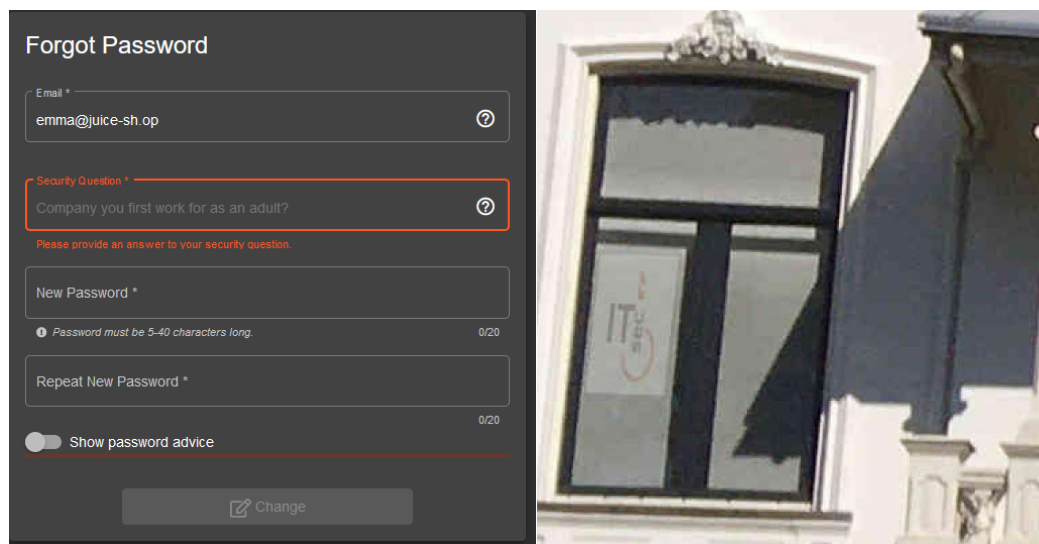
Step 4: To find the answer, access the photo-wall and find a pic submitted by Emma.

Step 5: Save Emma’s image, head to a photo-editor such as Gimp or Photoshop, and insert the image.

Step 6: Looking closely at the image, there’s a window in the building that gives away the name of the company, ITsec.

### Remediation Steps:

- Don’t add sensitive information in photos that are posted publicly.

The image shows a 'Forgot Password' form on the left and a photograph of a building window on the right. The form has fields for 'Email \*' (emma@juice-sh.op), 'Security Question \*' (Company you first work for as an adult?), 'New Password \*', and 'Repeat New Password \*'. A red box highlights the 'Security Question' field. Below the form is a 'Change' button. The photograph shows a window with a sign that says 'ITsec'.

You successfully solved a challenge: Visual Geo Stalking (Determine the answer to Emma's security question by looking at an upload of her to the Photo Wall and use it to reset her password via the Forgot Password mechanism.)

### 9.4 Title: Repetitive Registration

**Rating:** Low

**URL:** <http://127.0.0.1:3000/#/register>

**Description:** This vulnerability allows an attacker to register a user without having to confirm the password.

**Proof of Concept:**

Step 1: Create a user and put the confirm password.

Step 2: Before sending the post request to the server, open Burp Suite and allow packet interception.

Step 3: Send the packet and in Burp, change the value of the “PasswordRepeat” to an empty string and then forward the packet. The website should accept it and create the user.

### Remediation Steps:

- If the client-side checks that both passwords match, there is no reason to send the repeat password to the server.

The image shows a 'User Registration' form on the left and a JSON payload on the right. The form has fields for Email, Password, Repeat Password, Security Question, and Answer. The Email field contains 'test3@test.com'. The Password field contains '12345' and has a hint 'Password must be 5-40 characters long.' The Repeat Password field contains '12345' and has a hint '5/40'. The Security Question is 'Your favorite movie?' and the Answer is '777'. A 'Register' button is at the bottom. The JSON payload on the right is a POST request body with the following structure:

```
{  "email": "test3@test.com",  "password": "12345",  "passwordRepeat": "",  "securityQuestion": {    "id": 12,    "question": "Your favorite movie?",    "createdAt": "2022-12-30T11:03:49.962Z",    "updatedAt": "2022-12-30T11:03:49.962Z"  },  "securityAnswer": "777"}
```

You successfully solved a challenge: Repetitive Registration (Follow the DRY principle while registering a user.)

## 9.5 Title: CAPTCHA Bypass

**Rating:** Low

**URL:** <http://127.0.0.1:3000/#/contact>

**Description:** This vulnerability allows an attacker to bypass the CAPTCHA of a customer feedback

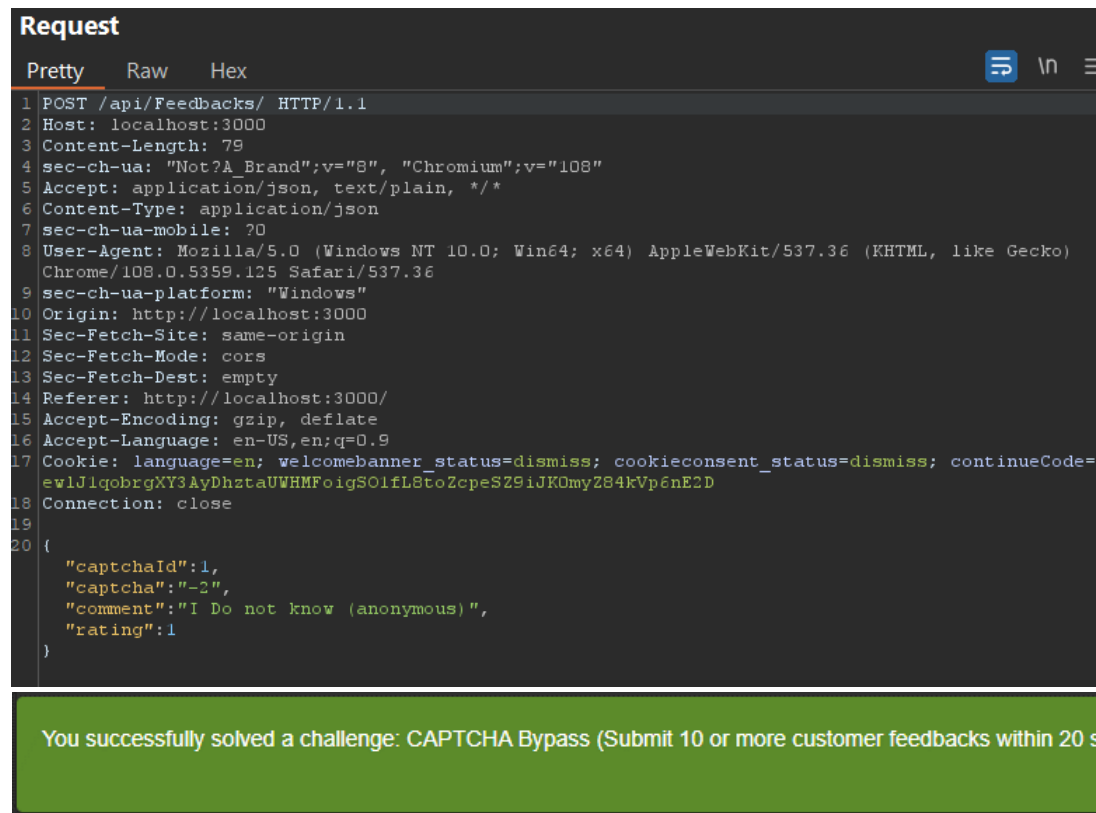
### Proof of Concept:

Step 1: fill the customer-feedback form and send the packet to Burp and then the repeater.

Step 2: Spam the send button many times and the ignore the fact that the “CAPTCHA” was made to limit the rate of submissions.

### Remediation Steps:

- Use actual CAPTCHA to verify that users are human.
- Only allow registered users to submit feedback.
- Limit the number of feedback comments that each registered user is allowed to submit in a given time span.



### 9.6 Title: 0-star rating

**Rating:** Low

**URL:** http://127.0.0.1:3000/#/contact

**Description:** This vulnerability allows an attacker to submit a review with 0 stars

### Proof of Concept:

Step 1: fill the customer-feedback form and send the packet to Burp.

Step 2: Change the rating from 0 to 1.

### Remediation Steps:

- Use input Validation.

You successfully solved a challenge: Zero Stars (Give a devastating zero-star feedback to the store.) X