CYBER DOJO

# SECURITY ASSESSMENT

Automated Full Scan + Manual Pentest

Submitted to: OWASP
Pentester Name: Ahmed Alaa-Eldin Mahdi
Ahmed Ali Abd El-Rahman
Youssef Mustafa Fawzy Gadallah
Ahmed Mohamed Abdelazize Elngar
Mostafa Abdelhamed Ahmed

Sprints group: 5

Date of Testing: 10 OCT 2024
Date of Report Delivery: 24 OCT 2024

# Table of Contents

## Contents

# Security Engagement Summary

## Engagement Overview

The Dojuicer organization engaged group #5 to conduct vulnerability assessment for a legacy web application to help them understand what security risks the web-application is posing the organization to, and propose solutions to mitigate the findings.

## Scope

The assessment was performed within the predefined scope of this engagement as listed below.

| Type | Name | Scope |
|---|---|---|
| Web-application | OWASP Juice shop | http://localhost:3000/ |

## Executive Risk Analysis

According to our assessment the client is **not secure.** Below is a table summarizing found vulnerabilities and their score

| No | Vulnerability | Risk | Impact | Attack Difficulty |
|---|---|---|---|---|
| 1 | Login page is vulnerable to SQL injection | Critical | Bypassing Authentication | Easy |
| 2 | FTP page is vulnerable to null byte injection | Critical | Bypassing restrictions and downloading files | Easy |
| 3 | Cryptographic failure | Critical | Exposes sensitive data | Easy |
| 4 | Login page does not lockout accounts | Critical | Web app becomes vulnerable to brute force | Easy |
| 5 | Allowing extremely week passwords | Critical | Passwords become easy to guess | Easy |

## Executive Recommendation

Our security assessment identified significant vulnerabilities requiring immediate attention. These issues present material risks to business operations, customer data, and company reputation.

Critical Findings and Recommendations:

**1. Database Security Vulnerability (High Risk)**

**Business Impact:**

- Potential unauthorized access to customer data

- Risk of data breach and associated costs
- Possible regulatory compliance violations
- Reputational damage

**Recommendation:**

Allocate development resources to implement industry-standard database security controls. Estimated timeline: 3-4 weeks.

Estimated cost: Medium

Priority: Immediate

**2. Data Protection Weaknesses (High Risk)**

**Business Impact:**

- Exposure of sensitive business and customer information
- Non-compliance with data protection regulations
- Potential financial penalties
- Loss of customer trust

**Recommendation:**

Upgrade data protection systems to current security standards and implement robust key management processes. Estimated timeline: 4-5 weeks.

Estimated cost: Medium to High

Priority: Immediate

**3. File System Security Issues (Medium Risk)**

**Business Impact:**

- Potential unauthorized access to system files
- Risk of data manipulation
- System integrity concerns

**Recommendation:**

Enhance file system security controls and implement proper access management. Estimated timeline: 2 weeks.

Estimated cost: Low to Medium

Priority: High

**4. Security Monitoring Gaps (Medium Risk)**

**Business Impact:**

- Delayed incident detection and response
- Inability to track unauthorized access attempts
- Challenges in meeting audit requirements
- Increased investigation costs during security events

**Recommendation:**

Implement comprehensive security monitoring solution and establish a security operations process. Estimated timeline: 6 weeks.

Estimated cost: Medium to High

Priority: High

**Next Steps:**

1. Approve resource allocation for remediation
2. Prioritize fixes based on risk level
3. Establish project timeline with IT team
4. Schedule follow-up security assessment

Total Estimated Implementation Timeline: 8-12 weeks

Our security team is available to provide detailed technical specifications and support the remediation process.

## Significant Vulnerability Summary

•

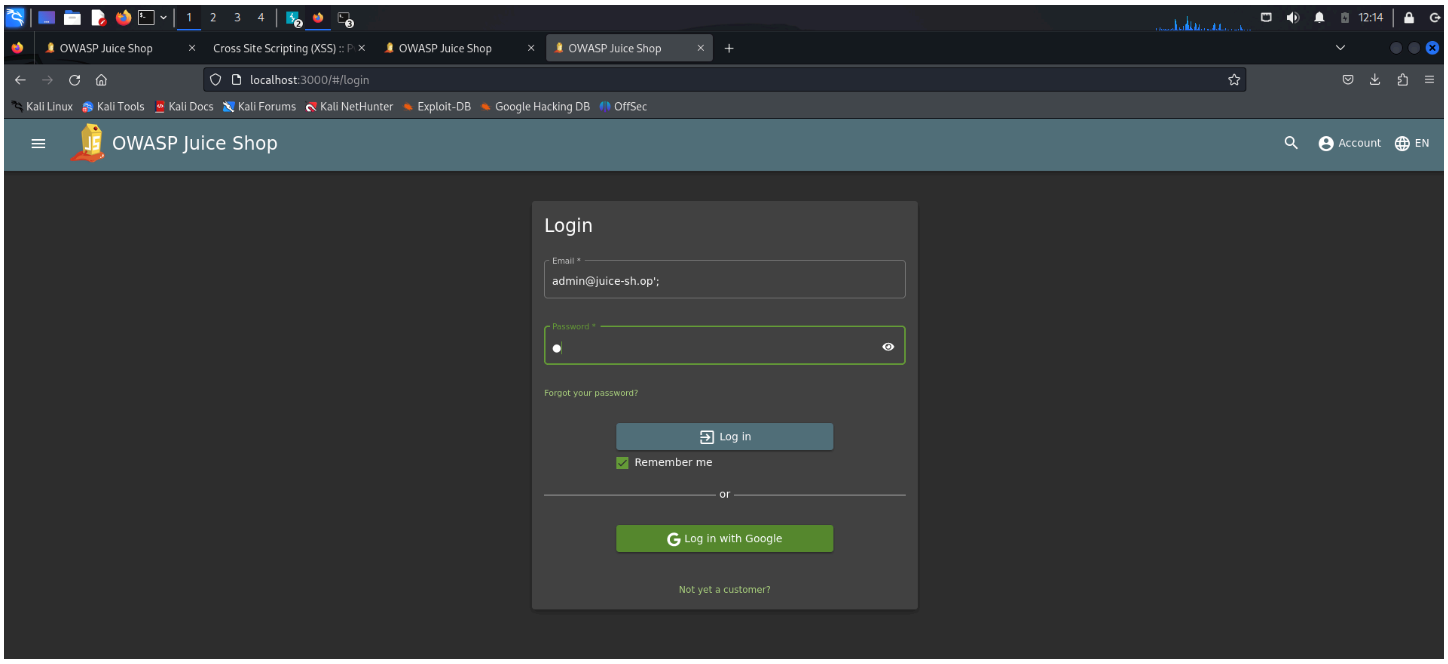
| No | Vulnerability | Risk | score | Attack Difficulty |
|---|---|---|---|---|
| 1 | SQL Injection | Critical | 9.3 | Easy |
| 2 | Cryptographic failure | High | 8.8 | Easy |
| 3 | Security logging and monitoring failure | High | 8.8 | Easy |
| 4 | Null byte injection | High | 8.7 | Easy |
| 5 | Week password policy | High | 8 | Easy |
| 6 | DOM XSS | High | 7.1 | Easy |

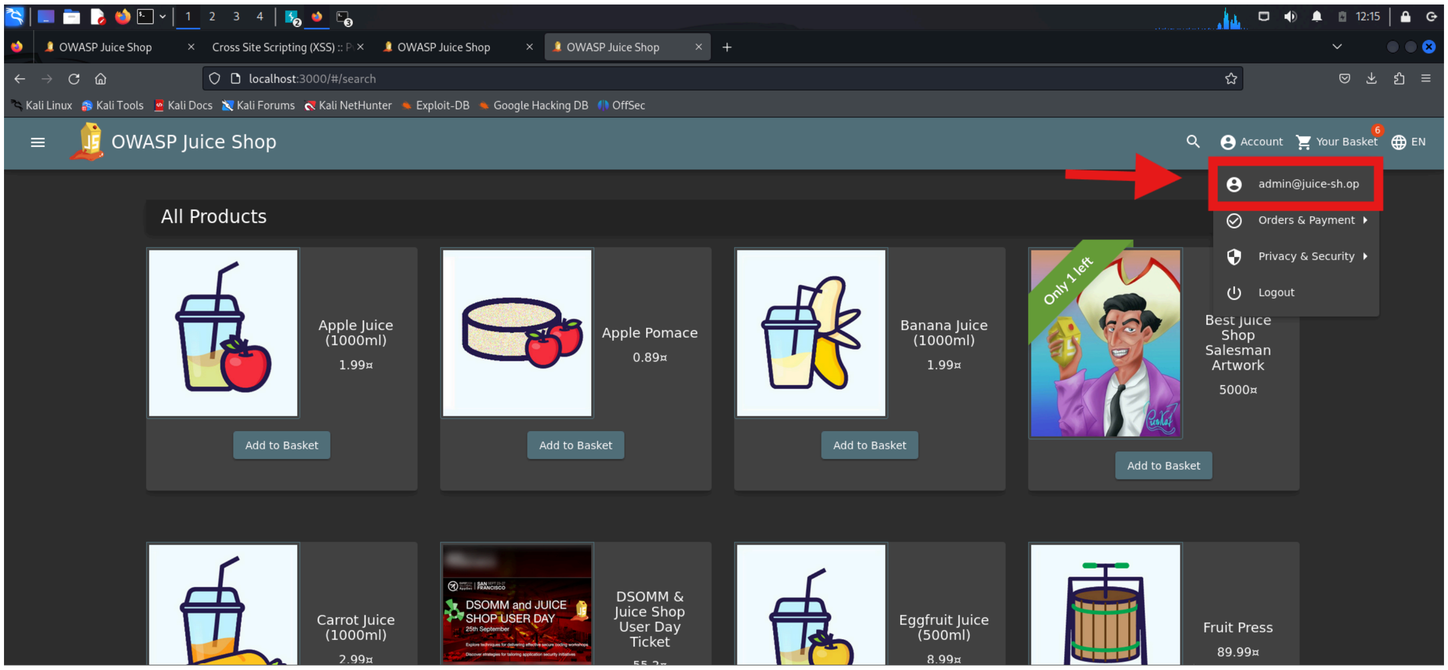
## Significant Vulnerability Detail

### SQL Injection

**RISK LEVEL Critical**

The vulnerability was identified by testing SQL injection payloads against the web app login page. The vulnerability was validated by logging in as one of the customers with having to authenticate.





## Probability of exploiting

This attack has a high probability of being exploited as it’s easy to perform.

## Impacted party

Every user on the web application.

## Remediation

- The preferred option is to use a safe API, which avoids using the interpreter entirely, provides a parameterized interface, or migrates to Object Relational Mapping Tools (ORMs).
  Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
  Note: SQL structures such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection**.**

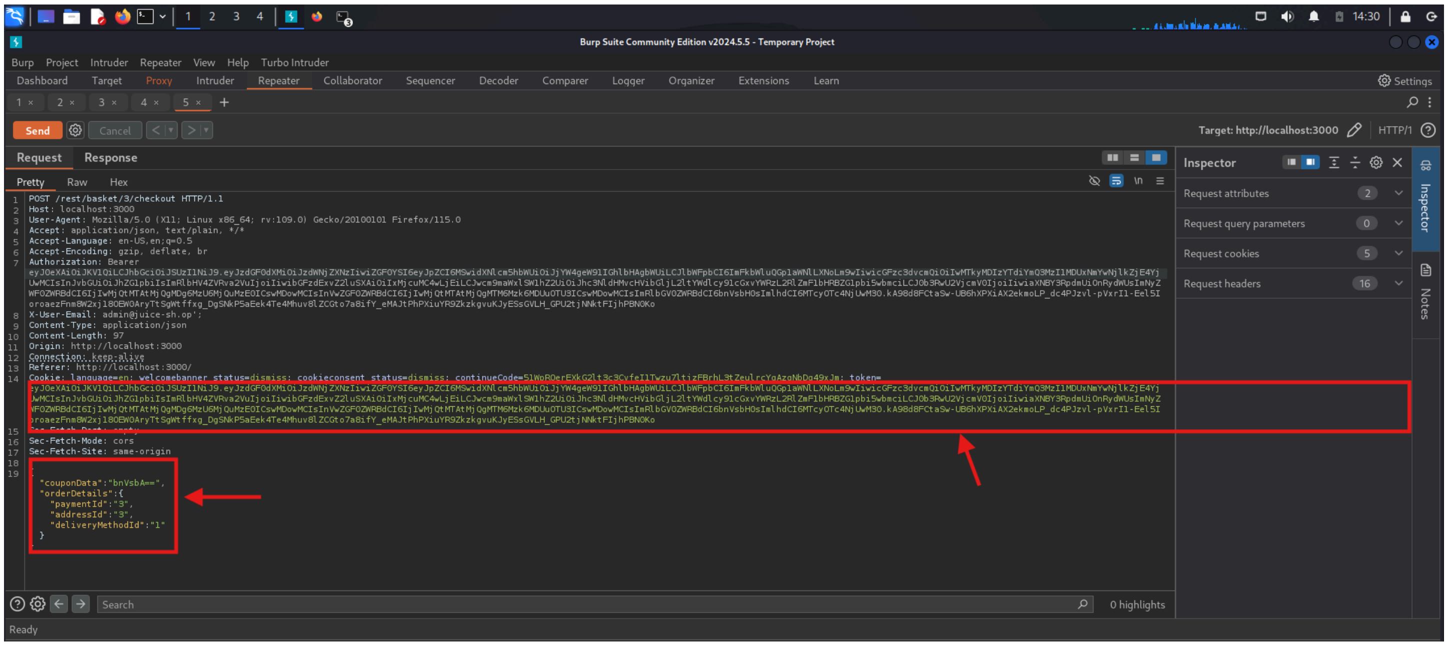
---

## Cryptographic Failure

**RISK LEVEL HIGH**

The vulnerability was identified while analyzing the requests during the checkout process. We found that the web app uses MD5 as the hashing algorithm to hash sensitive data such as passwords. MD5 is a deprecated hashing algorithm which means if anyone were to intercept the requests between server and client they can easily exfiltrate the data.

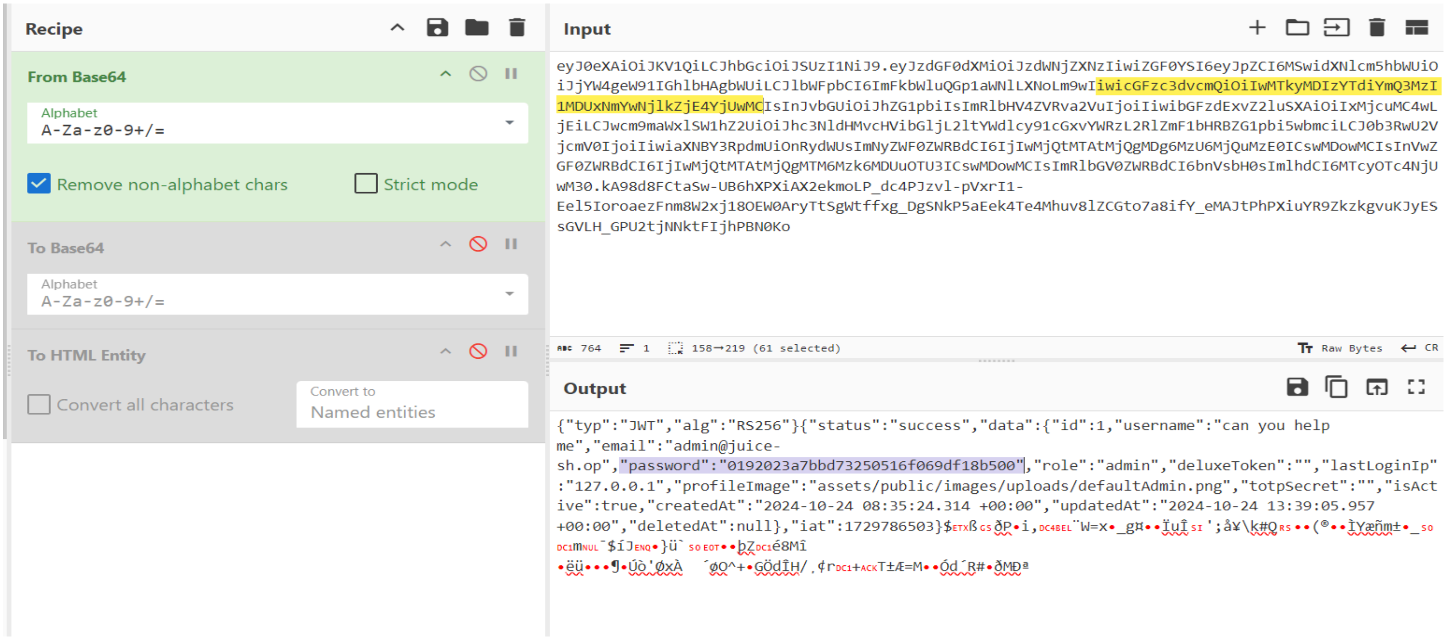
The web app uses MD5 to hash passwords inside JWT token we can find this token in the request. The JWT token itself contains in its' body sensitive data that should not be sent through the body.


Burp Suite Community Edition v2024.5.5 - Temporary Project
Burp Project Intruder Repeater View Help Turbo Intruder
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send Cancel
Target: http://localhost:3000 HTTP/1
Request Response
Pretty Raw Hex
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 16
Search
0 highlights
Ready

```
POST /rest/basket/3/checkout HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiJjYW4geW91IGhlbHAgbWUiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTAtMjQgMDg6MzU6MjQuMzE0ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMTAtMjQgMTM6Mzk6MDUuOTU3ICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyOTc4NjUwM30.kA98d8FCtaSw-UB6hXPXiAX2ekmoLP_dc4PJzvl-pVxrI1-Eel5IoroaezFnm8W2xj18OEW0AryTtSgWtffxg_DgSNkP5aEek4Te4Mhuv8lZCGto7a8ifY_eMAJtPhPXiuYR9ZkzkgvuKJyESsGVLH_GPU2tjNNktFIjhPBN0Ko
X-User-Email: admin@juice-sh.op';
Content-Type: application/json
Content-Length: 97
Origin: http://localhost:3000
Connection: keep-alive
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=51WpR0erEXkG2lt3c3CvfeI1Twzu7ltizFBrhL3tZeulrcYgAzgNbDg49xJm; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiJjYW4geW91IGhlbHAgbWUiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTAtMjQgMDg6MzU6MjQuMzE0ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMTAtMjQgMTM6Mzk6MDUuOTU3ICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyOTc4NjUwM30.kA98d8FCtaSw-UB6hXPXiAX2ekmoLP_dc4PJzvl-pVxrI1-Eel5IoroaezFnm8W2xj18OEW0AryTtSgWtffxg_DgSNkP5aEek4Te4Mhuv8lZCGto7a8ifY_eMAJtPhPXiuYR9ZkzkgvuKJyESsGVLH_GPU2tjNNktFIjhPBN0Ko
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "couponData":"bnVsbA==",
  "orderDetails":{
    "paymentId":"3",
    "addressId":"3",
    "deliveryMethodId":"1"
  }
}
```

We can notice that JWTs' body is base64 encoded we decode it we see the following



```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiJjYW4geW91IGhlbHAgbWUiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTAtMjQgMDg6MzU6MjQuMzE0ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMTAtMjQgMTM6Mzk6MDUuOTU3ICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyOTc4NjUwM30.kA98d8FCtaSw-UB6hXPXiAX2ekmoLP_dc4PJzvl-pVxrI1-
Eel5IoroaezFnm8W2xj18OEW0AryTtSgWtffxg_DgSNkP5aEek4Te4Mhuv8lZCGto7a8ifY_eMAJtPhPXiuYR9ZkzkgvuKJyESsGVLH_GPU2tjNNktFIjhPBN0Ko
```

```
{"typ":"JWT","alg":"RS256"}{"status":"success","data":{"id":1,"username":"can you help me","email":"admin@juice-sh.op","password":"0192023a7bbd73250516f069df18b500","role":"admin","deluxeToken":"","lastLoginIp":"127.0.0.1","profileImage":"assets/public/images/uploads/defaultAdmin.png","totpSecret":"","isActive":true,"createdAt":"2024-10-24 08:35:24.314 +00:00","updatedAt":"2024-10-24 13:39:05.957 +00:00","deletedAt":null},"iat":1729786503}$ßðP•i,¨W=x•_g¤••ÏuÎ';å¥\k#Q••(®••ÌYæñm±•_m¯$íJ•}ü`••þZé8Mî
•ëü•••¶•Úò'ØxÀ ´ØO^+•GÖdÎH/¸¢r+T±Æ=M••Ód´R#•ðMÐª
```

From the previous screen we can easily get the password by reversing the MD5 hash in password parameter

0192023a7bbd73250516f069df18b500 -> admin123

We can also see an interesting parameter “role” for regular users we can abuse this parameter to perform privilege Escalation.

## Probability of exploiting

This attack has a high probability of being exploited as it’s easy to perform.

## Impacted party

Every user on the web application.

## Remediation

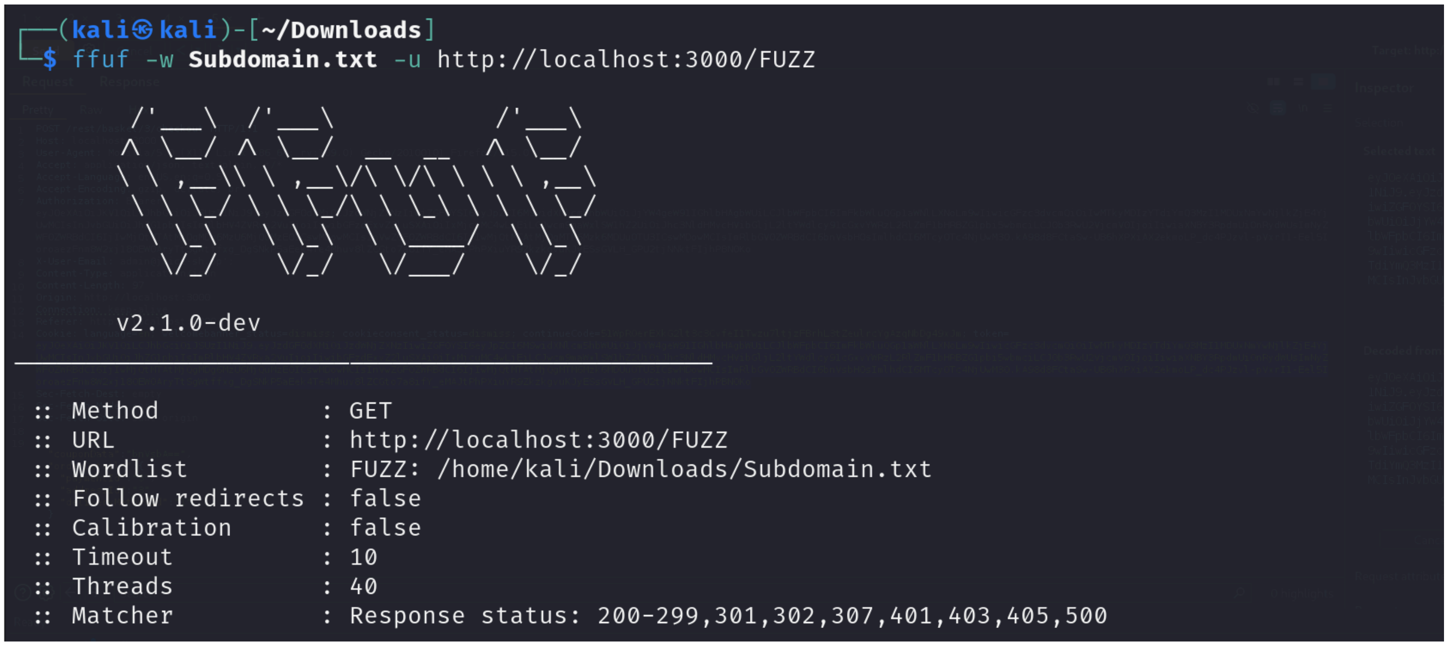
Do the following, at a minimum:

- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for response that contain sensitive data.
- Apply required security controls as per the data classification.
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.
- Initialization vectors must be chosen appropriate for the mode of operation. For many modes, this means using a CSPRNG (cryptographically secure pseudo random number generator). For modes that require a nonce, then the initialization vector (IV) does not need a CSPRNG. In all cases, the IV should never be used twice for a fixed key.
- Always use authenticated encryption instead of just encryption.
- Keys should be generated cryptographically randomly and stored in memory as byte arrays. If a password is used, then it must be converted to a key via an appropriate password base key derivation function.
- Ensure that cryptographic randomness is used where appropriate, and that it has not been seeded in a predictable way or with low entropy. Most modern APIs do not require the developer to seed the CSPRNG to get security.
- Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5 .
- Verify independently the effectiveness of configuration and settings.

## Security Logging and Monitoring Failure

**RISK LEVEL HIGH**

The vulnerability was identified while brute forcing against the login page. The web app logging and monitoring system fails as it does not specify a number of failed attempts, and it does not lock out the account no matter the number of failed attempts. This makes the web app easily susceptible for brute force attacks  across all it's pages not just the login page. This allows an attacker to perform attacks against login page, captcha, 2FA and perform forced browsing.



```
┌──(kali㉿kali)-[~/Downloads]
└─$ ffuf -w Subdomain.txt -u http://localhost:3000/FUZZ

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://localhost:3000/FUZZ
 :: Wordlist         : FUZZ: /home/kali/Downloads/Subdomain.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
```

### Probability of exploiting

This attack has a high probability of being exploited as it's easy to perform.

### Impacted party

Every user on the web application.

### Remediation

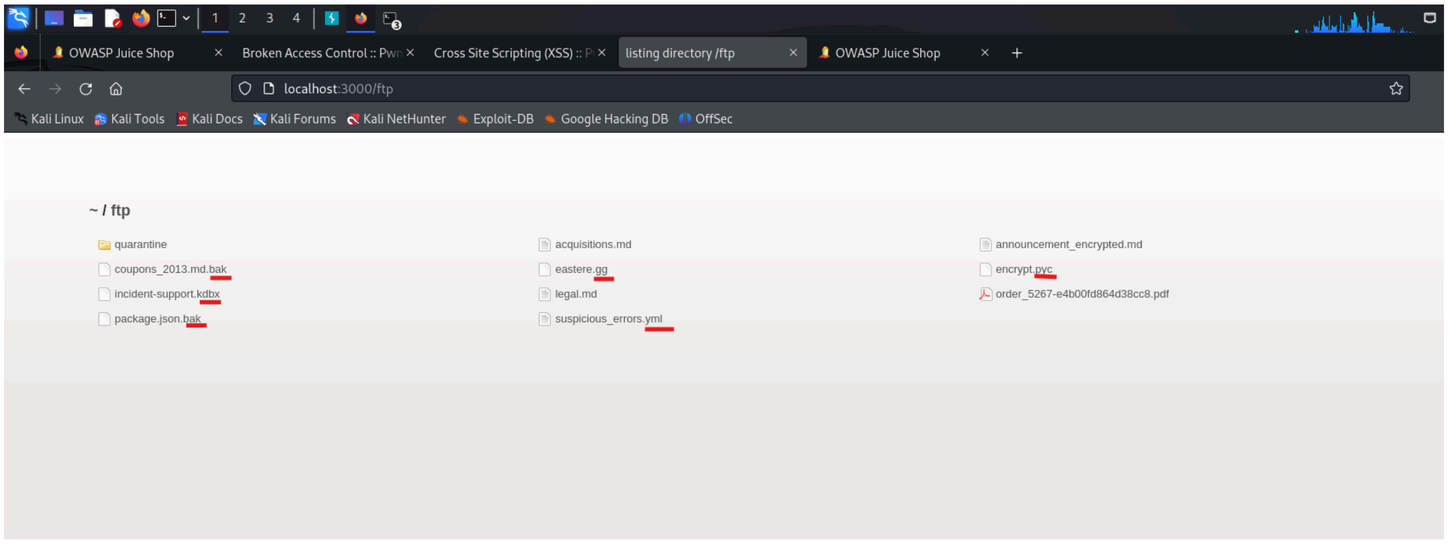
Developers should implement some or all the following controls:

- Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that log management solutions can easily consume.
- Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- DevSecOps teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.
- Establish or adopt an incident response and recovery plan, such as National Institute of Standards and Technology (NIST) 800-61r2 or later.

## Null Byte Injection

**RISK LEVEL HIGH**

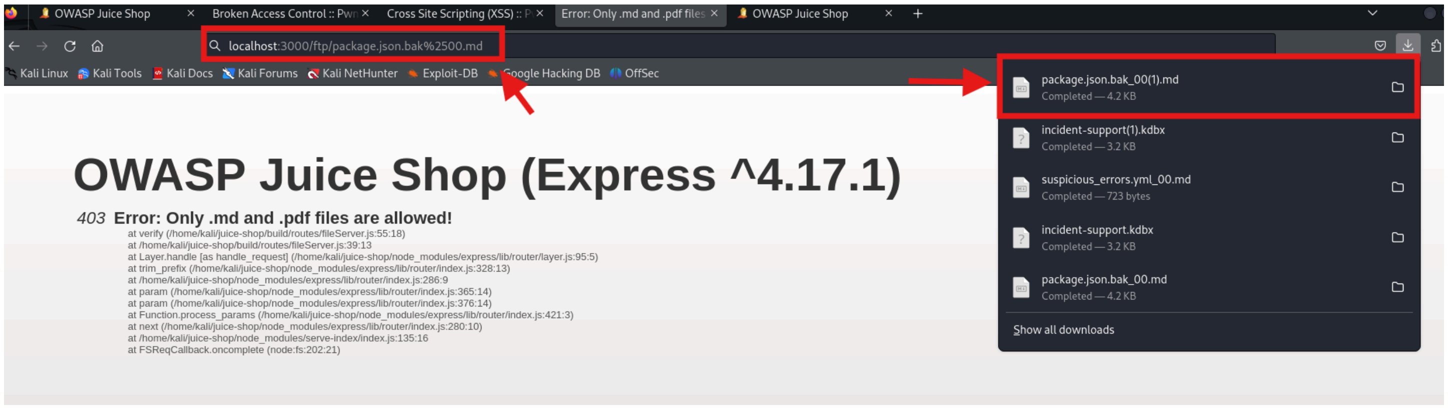
This vulnerability was identified while searching in the FTP page. In this page we find some interesting files but we can't download it as download is restricted to pdf and md extensions we can bypass this restriction by using null byte injection.



localhost:3000/ftp

~ / ftp

quarantine
coupons_2013.md.bak
incident-support.kdbx
package.json.bak
acquisitions.md
eastere.gg
legal.md
suspicious_errors.yml
announcement_encrypted.md
encrypt.pyc
order_5267-e4b00fd864d38cc8.pdf



localhost:3000/ftp/package.json.bak%2500.md

```
OWASP Juice Shop (Express ^4.17.1)

403 Error: Only .md and .pdf files are allowed!
    at verify (/home/kali/juice-shop/build/routes/fileServer.js:55:18)
    at /home/kali/juice-shop/build/routes/fileServer.js:39:13
    at Layer.handle [as handle_request] (/home/kali/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/home/kali/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /home/kali/juice-shop/node_modules/express/lib/router/index.js:286:9
    at param (/home/kali/juice-shop/node_modules/express/lib/router/index.js:365:14)
    at param (/home/kali/juice-shop/node_modules/express/lib/router/index.js:376:14)
    at Function.process_params (/home/kali/juice-shop/node_modules/express/lib/router/index.js:421:3)
    at next (/home/kali/juice-shop/node_modules/express/lib/router/index.js:280:10)
    at /home/kali/juice-shop/node_modules/serve-index/index.js:135:16
    at FSReqCallback.oncomplete (node:fs:202:21)
```

package.json.bak_00(1).md
Completed — 4.2 KB
incident-support(1).kdbx
Completed — 3.2 KB
suspicious_errors.yml_00.md
Completed — 723 bytes
incident-support.kdbx
Completed — 3.2 KB
package.json.bak_00.md
Completed — 4.2 KB
Show all downloads

### Probability of exploiting

This attack has a high probability of being exploited as it's easy to perform.

### Impacted party

No users affected

### Remediation

To protect against Null Byte Injection, consider implementing the following measures:

- Sanitize User Inputs: Ensure all user inputs are properly sanitized to remove any malicious characters, including null bytes.
- Validate Input Data: Implement robust input validation techniques to check for unexpected or harmful data before processing.
- Use Parameterized Queries: Employ parameterized queries to prevent injection attacks by separating user input from code execution.

- Regular Security Audits: Conduct regular security audits and code reviews to identify and fix vulnerabilities in your application.
- Employ Web Application Firewalls (WAFs): Utilize WAFs to detect and block malicious traffic, including attempts to exploit Null Byte Injection.

## Week Password Policy (Identification and Authentication Failure)

**RISK LEVEL HIGH**

This vulnerability was identified while investigating the login page we can find that the web application does not force any kind of strong password policy besides the password being 5 characters or longer. This way users can have passwords like  12345, pass123, user1 this type of week and predictable password can cause a huge security risk especially if it was in an admin account like we saw earlier.

### Probability of exploiting

This attack has a high probability of being exploited as it’s easy to perform.

### Impacted party

Users that have week passwords

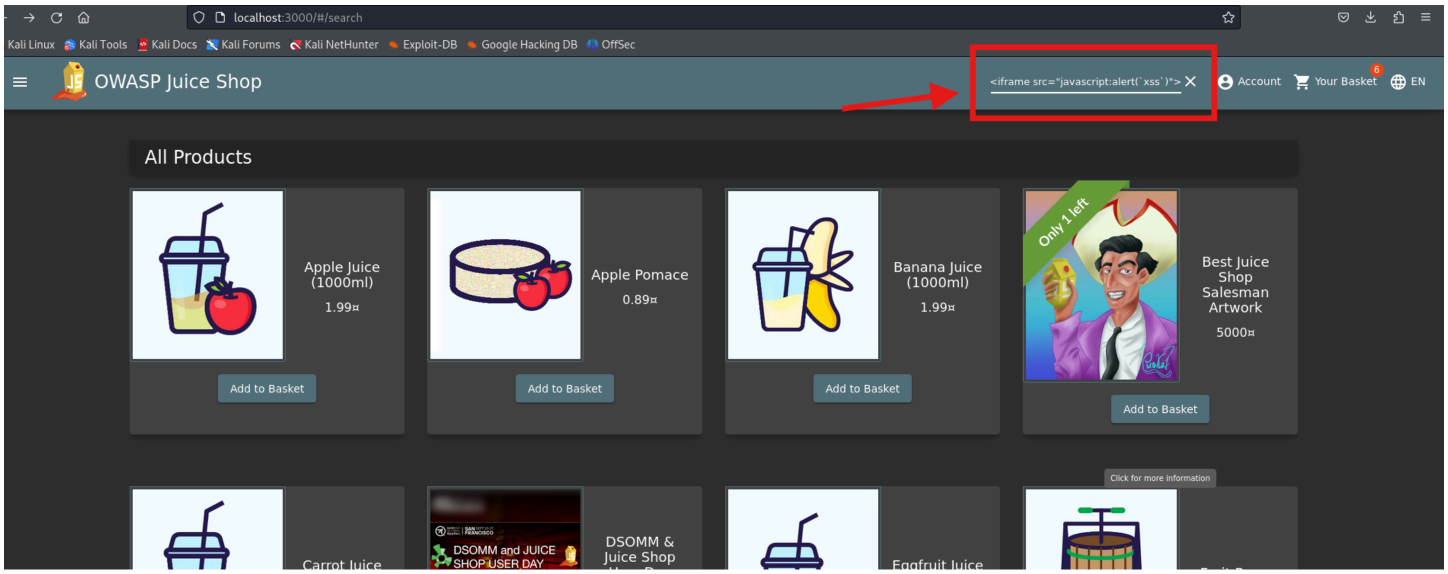
### Remediation

- Where possible, implement multi-factor authentication to prevent automated credential stuffing, brute force, and stolen credential reuse attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak password checks, such as testing new or changed passwords against the top 10,000 worst passwords list.
- Align password length, complexity, and rotation policies with National Institute of Standards and Technology (NIST) 800-63b's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence-based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts, but be careful not to create a denial of service scenario. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session identifier should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.

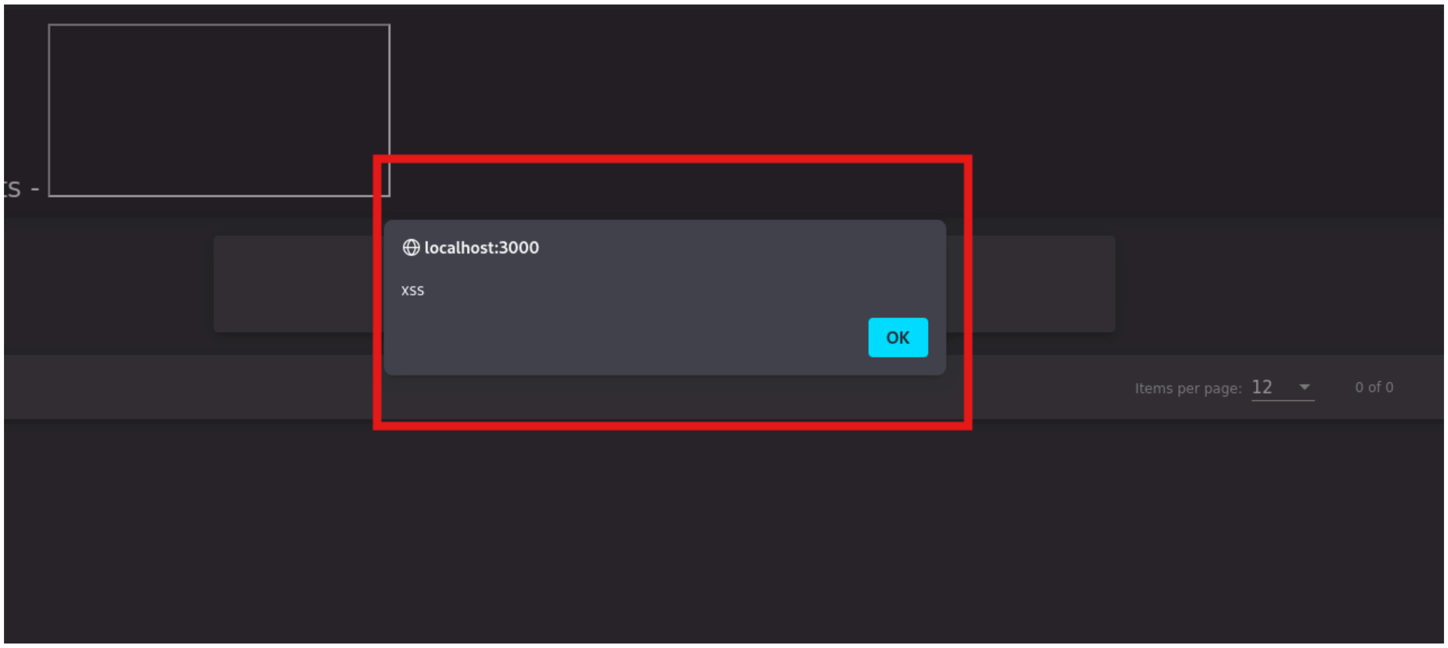
## DOM XSS

**RISK LEVEL HIGH**

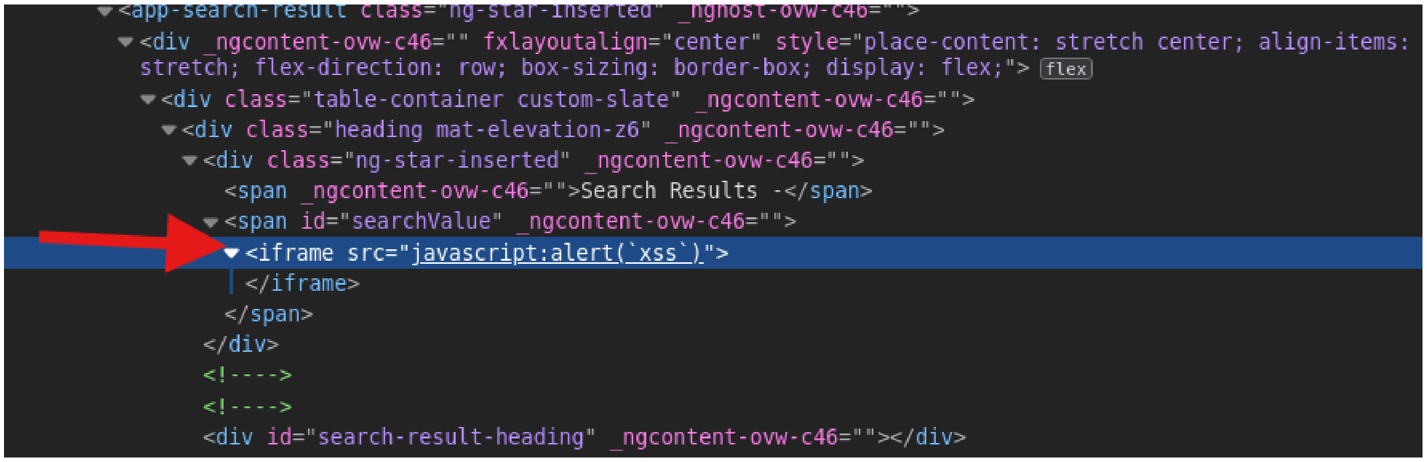
This vulnerability was identified while testing the search bar in the main page. We can easily find that it's vulnerable to XSS payloads by inserting one in it.



```
<iframe src="javascript:alert(`xss`)">
```



```
xss
```



```
▼<app-search-result class="ng-star-inserted" _nghost-ovw-c46="">
  ▼<div _ngcontent-ovw-c46="" fxlayoutalign="center" style="place-content: stretch center; align-items:
    stretch; flex-direction: row; box-sizing: border-box; display: flex;"> flex
    ▼<div class="table-container custom-slate" _ngcontent-ovw-c46="">
      ▼<div class="heading mat-elevation-z6" _ngcontent-ovw-c46="">
        ▼<div class="ng-star-inserted" _ngcontent-ovw-c46="">
           <span _ngcontent-ovw-c46="">Search Results -</span>
          ▼<span id="searchValue" _ngcontent-ovw-c46="">
            ▼<iframe src="javascript:alert(`xss`)">
             </iframe>
           </span>
         </div>
         <!---->
         <!---->
         <div id="search-result-heading" _ngcontent-ovw-c46=""></div>
```

We can see when we inspect the page that our payload added an ifram element to the DOM (the DOM itself was modified) so this is a DOM XSS.

## Probability of exploiting

This attack has a high probability of being exploited as it’s easy to perform.

## Impacted party

Users that interact with links that have a malicious payload.

## Remediation

sanitize all untrusted data, even if it is only used in client-side scripts. If you have to use user input on your page, always use it in the text context, never as HTML tags or any other potential code.

Avoid methods such as document.innerHTML and instead use safer functions, for example, document.innerText and document.textContent. If you can, entirely avoid using user input, especially if it affects DOM elements such as the document.url, the document.location, or the document.referrer.

Also, keep in mind that DOM XSS and other types of XSS are not mutually exclusive. Your application can be vulnerable to both reflected/stored XSS and DOM XSS. The good news is that if user input is handled properly at the foundation level (e.g. your framework), you should be able to mitigate all XSS vulnerabilities.

## Methodology

### Quick Overview

The penetration test was conducted following the OWASP Testing Guide's standard methodology, which ensures a comprehensive assessment of web application security. The testing process is divided into three key phases: **Information Gathering**, **Vulnerability Identification**, and **Exploitation**. Each phase is designed to identify vulnerabilities and evaluate the effectiveness of security controls within the OWASP Juice Shop environment. The overall goal of the assessment was to discover, analyze, and report any vulnerabilities or weaknesses that could potentially be exploited by malicious attackers.

### Assessment Tool Set Selection

To facilitate a thorough assessment of OWASP Juice Shop, a combination of automated and manual tools were utilized. These tools were selected based on their ability to detect various classes of vulnerabilities and ease of integration into the testing workflow.

- **Burp Suite** (Professional): Used for intercepting requests, performing active and passive scans, and automating testing of input validation vulnerabilities.
- **OWASP ZAP**: Employed for automated vulnerability scanning and testing of common web application vulnerabilities.
- **Nessus**: Used for identifying vulnerabilities, misconfigurations, and weaknesses on the infrastructure side.
- **Nikto**: A web server scanner used for basic enumeration and discovery of known vulnerabilities.
- **SQLMap**: For identifying and exploiting SQL injection vulnerabilities.
- **Firefox Developer Tools**: Used for manual exploration, input testing, and verifying the behavior of web functionalities.
- **Kali Linux**: The operating system used to consolidate the testing tools and environment.
- **Custom Scripts**: Tailored scripts were used where necessary for automated interaction with various web application components and to attempt exploitation of discovered vulnerabilities.

### Assessment Methodology Detail

The assessment was conducted using a mix of automated and manual techniques in line with the **OWASP Testing Guide**. Below is the detailed breakdown of the testing methodology:

Information Gathering

- **Reconnaissance and Fingerprinting**: The initial phase involved collecting information about the OWASP Juice Shop's server configuration, technologies in use, and potential points of entry. Automated tools like **Nmap** were used for port scanning, and **Nikto** was used to identify potential weaknesses in the server configuration.

- **Enumeration of Endpoints**: All publicly accessible endpoints, parameters, and features of the Juice Shop web application were enumerated to create a testing scope. Tools like **Burp Suite** and **OWASP ZAP** were employed to map out the application.

Vulnerability Identification

- **Input Validation Testing**: Tests were performed to identify weaknesses in input validation mechanisms. This included:
  - Testing for **SQL injection** using **SQLMap** and manual input fuzzing.
  - **Cross-Site Scripting (XSS)** testing, both reflected and stored, by inserting crafted payloads in all input fields.
  - **Cross-Site Request Forgery (CSRF)** testing to check for unprotected requests.

- **Session Management Testing**: The application's authentication and session management mechanisms were examined, including cookie handling, session expiration, and the presence of secure flags. **Burp Suite** and **Firefox Developer Tools** were used to manipulate cookies and session tokens.

- **Authentication and Authorization**: Tests were conducted to validate the effectiveness of authentication mechanisms and to check if users with lower privileges could access restricted resources. **Burp Suite** was used to tamper with user roles, session IDs, and HTTP methods to bypass authentication controls.

- **File Upload Vulnerabilities**: OWASP Juice Shop's file upload functionality (if present) was tested to ensure that malicious files could not be uploaded and executed.

Exploitation

- **Privilege Escalation**: Attempts were made to elevate privileges by exploiting vulnerabilities found in authorization checks or by leveraging poorly protected user sessions.

- **Injection Attacks**: Identified input vulnerabilities were exploited using tools like **SQLMap** for SQL injection or manual payload insertion for **XSS**.

- **Data Exfiltration**: In cases where sensitive data exposure was found (e.g., via **IDOR**, SQLi), data was exfiltrated to validate the impact of the vulnerability.

- **Exploitation of Business Logic Flaws**: Manual testing was conducted to identify business logic vulnerabilities, ensuring that security controls enforced by the business processes were functioning as intended.

Reporting and Verification

- **Results Documentation**: Each vulnerability identified was documented with its impact, risk rating, and proof-of-concept. Screenshots and HTTP request/response logs were included for each successful exploit.

- **Remediation Recommendations**: For each vulnerability, mitigation steps and security best practices were recommended, in alignment with OWASP's guidelines.

---

This concluded the vulnerability assessment methodology portion of this report.