Owasp Juice ShopVAPT REPORT

short line

SANJAY S KUMAR

# 

| **SI NO** | **Vulnerability** |
| --- | --- |
| **1** | **XSS** |
| **2** | **Command Injection** |
| **3** | **SSRF** |
| **4** | **IDOR** |
| **5** | **HTTP parameter Pollution** |

# **1. Xss**

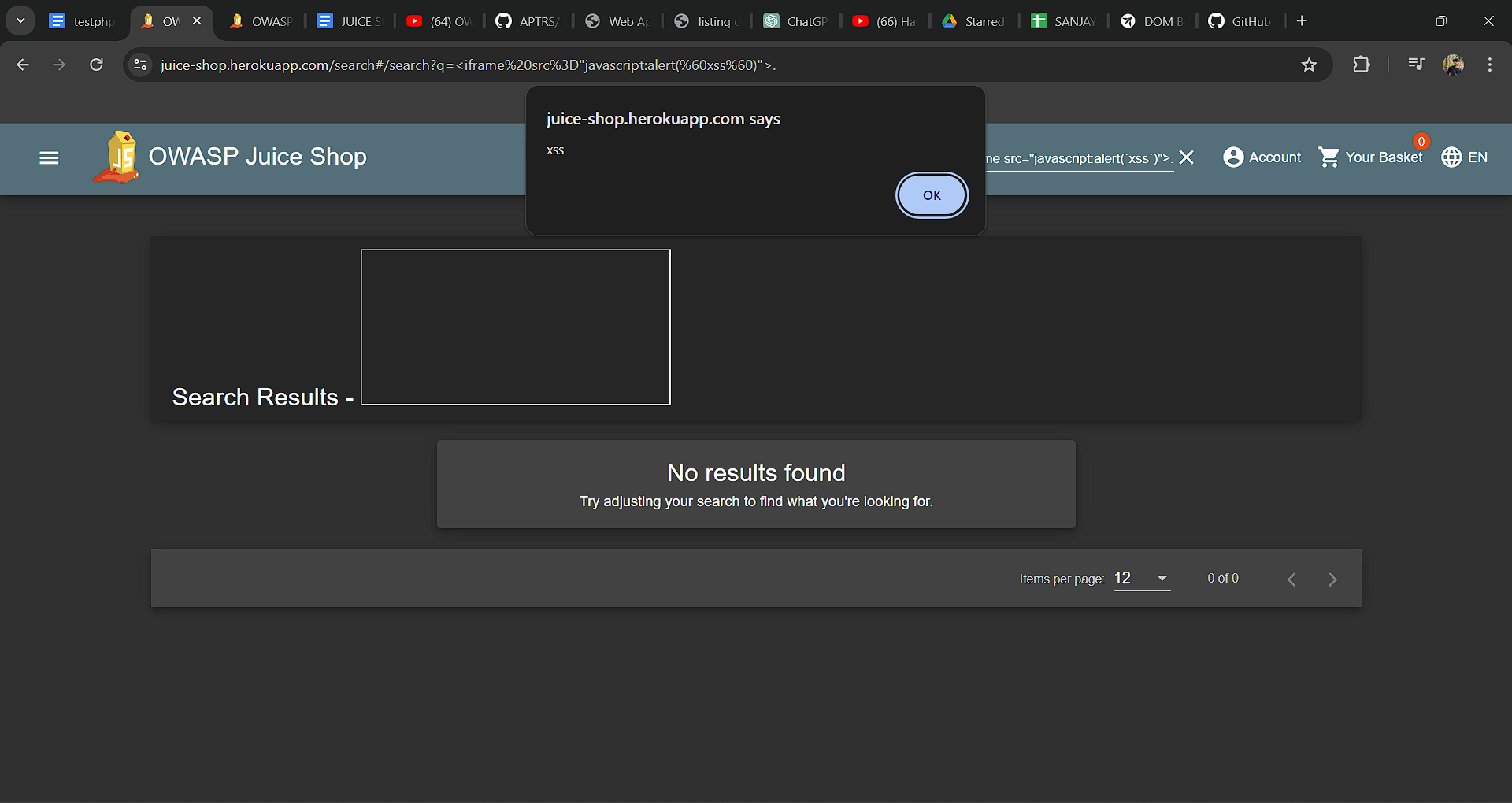
# **1.1 Description**

The search box within the web application is vulnerable to DOM XSS (cross-site scripting), posing a significant security risk that could potentially compromise user data and the integrity of the entire system.

## **1.2 Vulnerable instance**

https://juice-shop.herokuapp.com/#/

## **1.3 Proof of concept.**

As can be seen, the user is able to inject and execute the javascript code into the page. Using this payload “<iframe src="javascript:alert(`xss`)">”we can see that the code is executed successfully.

## **1.4 Mitigation**

**Input Validation and Sanitization:** Implement strict input validation and sanitization mechanisms to ensure that user-supplied input is free from malicious content. This includes validating input format, length, and type, as well as filtering out potentially dangerous characters or commands.

**Least Privilege Principle:** Follow the principle of least privilege by granting only the minimum necessary permissions to database users and restricting access to sensitive database operations and resources.

**Web Application Firewall :** Deploy a web application firewall to monitor and filter incoming HTTP traffic, detecting and blocking potential injection attacks in real-time

# **2.Command Injection**

## **2.1 Description**

The login page of this web application is susceptible to SQL injection, posing a security risk. Exploiting this vulnerability, an attacker could effortlessly gain access to the web application without requiring a username or password.

## **2.2 Vulnerable instance**

https://juice-shop.herokuapp.com/#/login

## **2.3 Proof of concept.**

The vulnerability lies in the sign-up process of this web application, lacking validation. Consequently, we can exploit it by injecting payloads such as "Abc' or 1=1;--".



## **2.4 Mitigation**

**Input Validation:** Implement strict input validation on the login page to ensure that user input is sanitized and validated before being used in SQL queries.Use parameterized queries or prepared statements to separate SQL code from user input, preventing attackers from injecting malicious SQL code.

**Web Application Firewall:** Deploy a WAF to monitor and filter incoming traffic to the web application. Configure the WAF to detect and block SQL injection attacks based on predefined rules .

**Security Testing and Code Reviews:** Conduct regular security testing, including penetration testing and code reviews, to identify and remediate vulnerabilities in the web application codebase. Automated tools and manual reviews can help uncover potential SQL injection vulnerabilities.

## 

## 

# **3.SSRF**

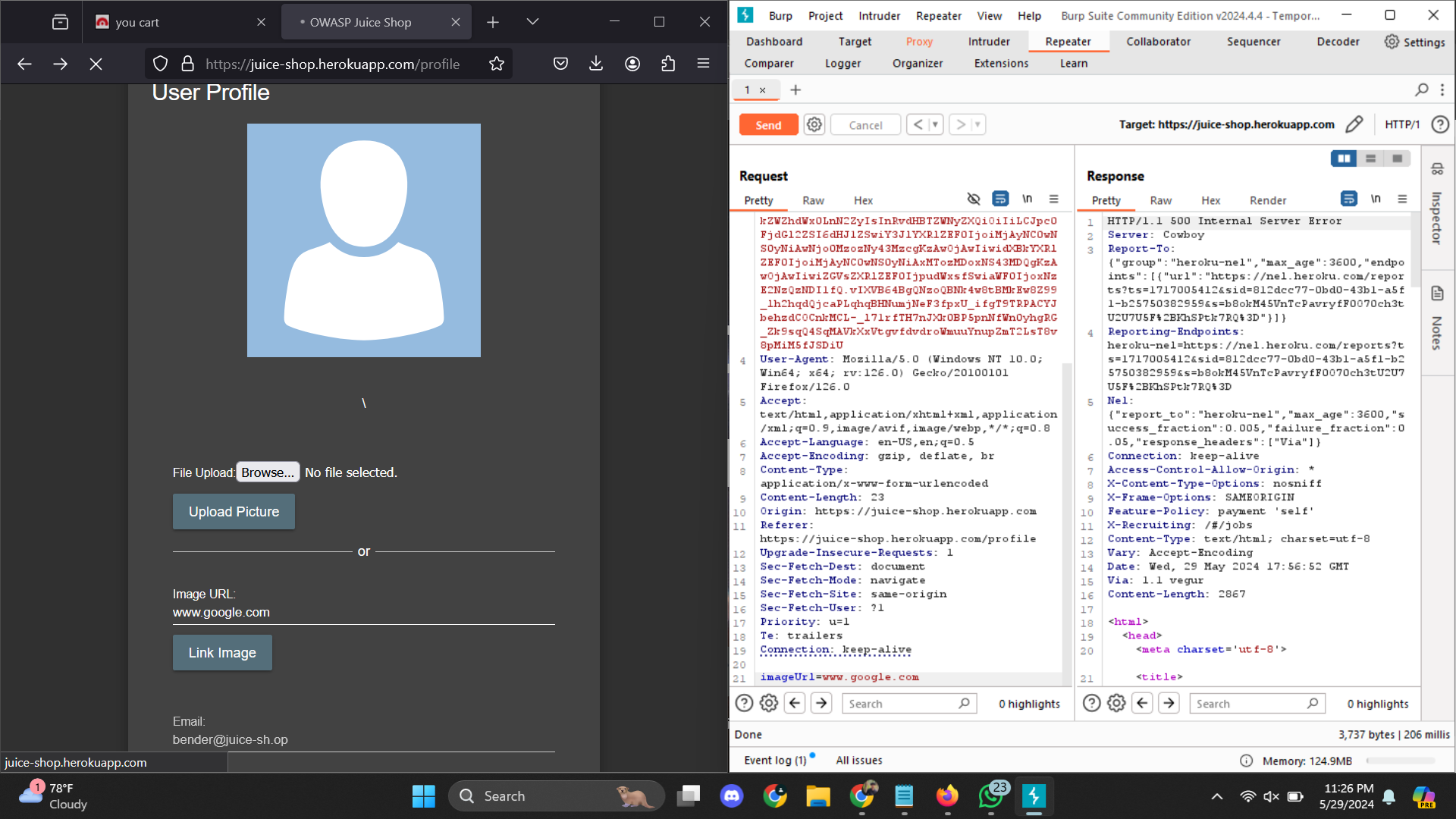
## **3.1 Description**

In this application there is an option for adding a url for adding a profile picture .In that option we can add a url that is executed on the server.

## **3.2 Vulnerable instance**

https://juice-shop.herokuapp.com/profile

## **3.3 Proof of concept.**

There is an option for adding a url for adding a profile .We can interrupt the request ,then we can see that the url is executed on the server .

## **3.4 Mitigation**

**Input Validation and Sanitization :** Whitelist trusted URLs.Validate and sanitize user inputs to ensure they follow expected patterns.

**Monitoring and Logging :** Log all outbound requests. Monitor for unusual outbound traffic patterns.

# **4.IDOR**

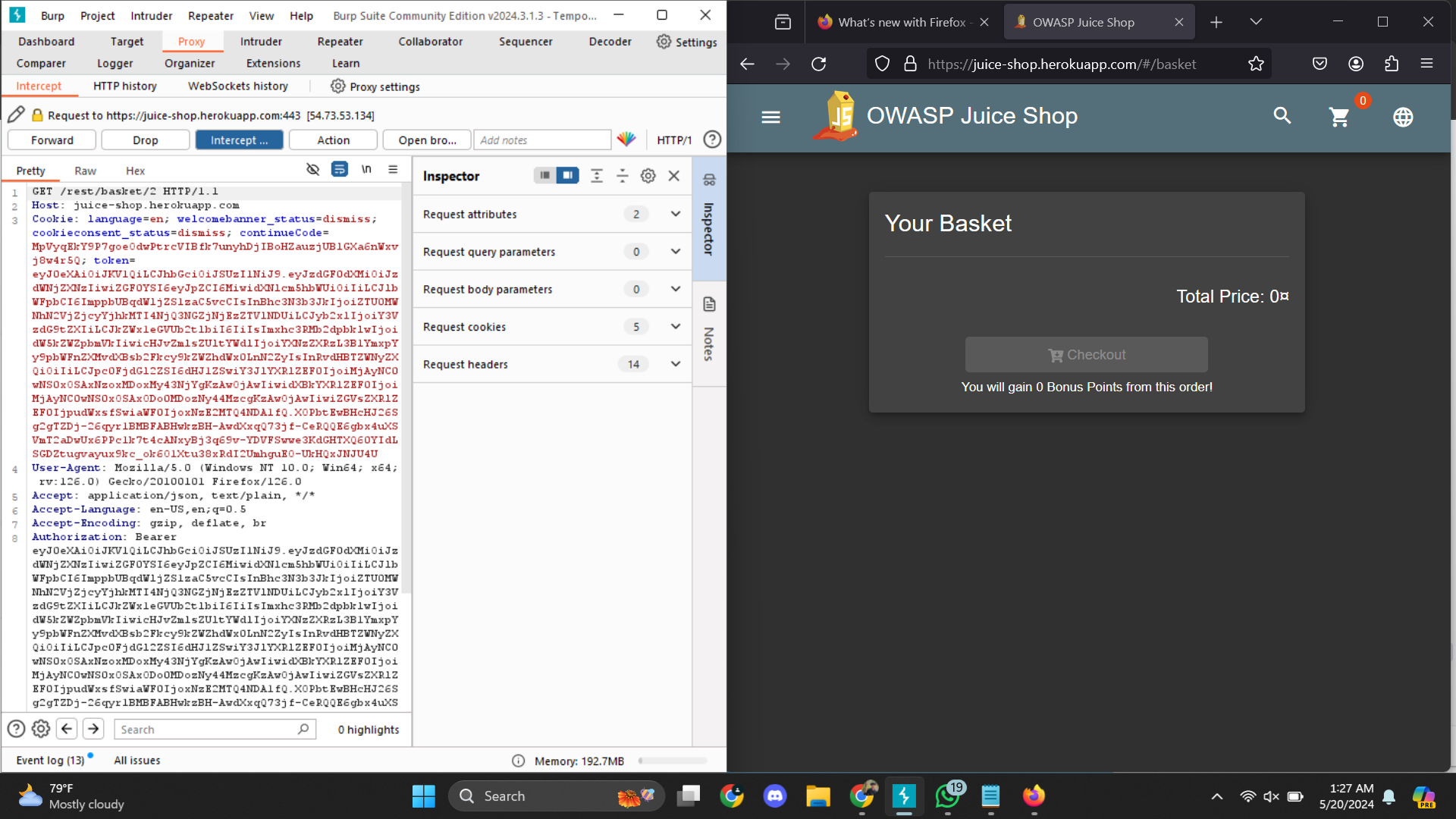
# **4.1 Description**

In this web application, the user id is not encrypted but rather stored in plain text format. Consequently, if an attacker intercepts the communication between the client and the server using tools like Burp Suite, they can easily manipulate the user ID and thereby gain unauthorized access to view or modify other users' data. This lack of encryption leaves the user ID susceptible to interception and manipulation, posing a significant security risk as it enables attackers to potentially access sensitive information belonging to other users.

## **4.2 Vulnerable instance**

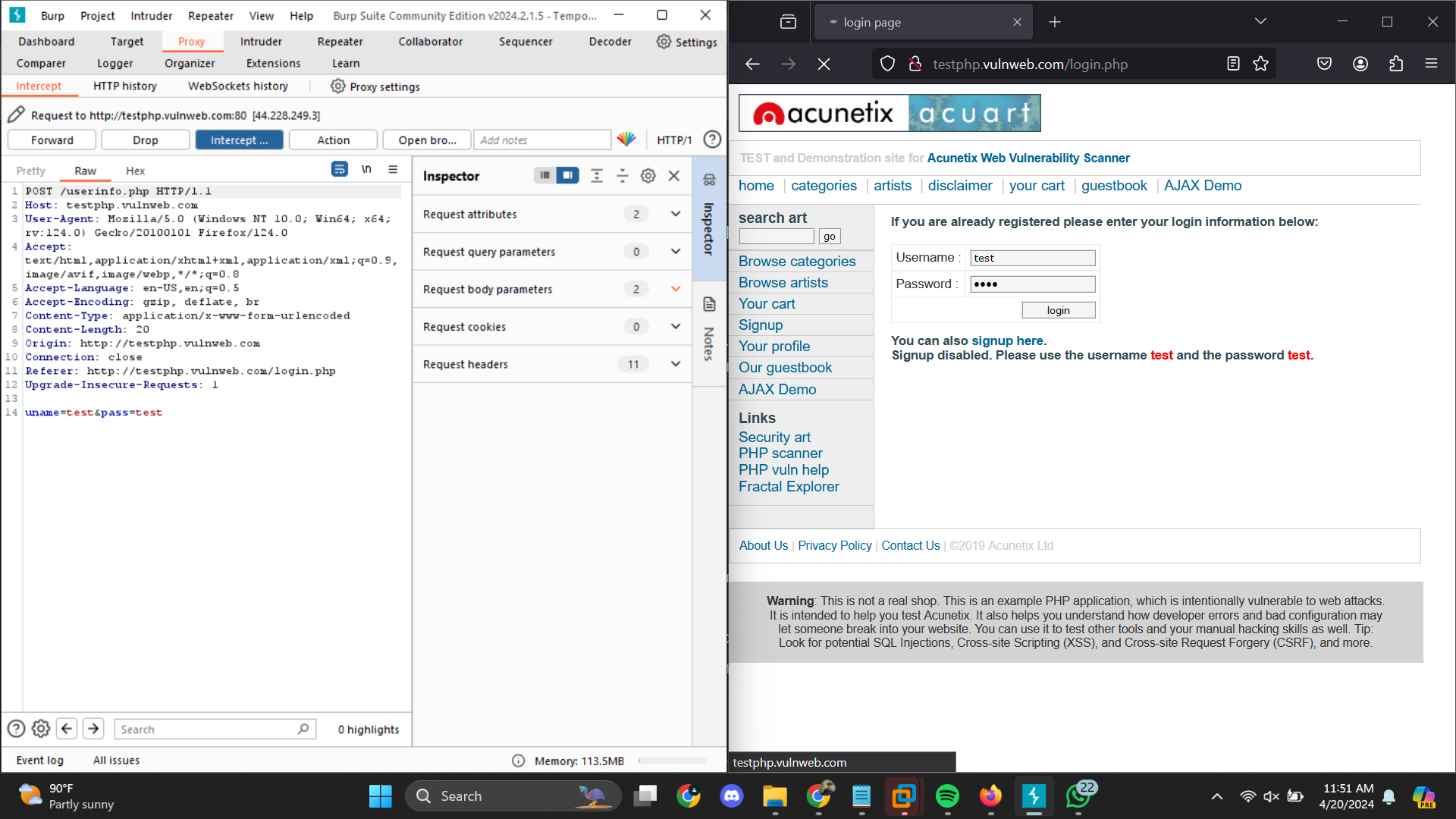
https://juice-shop.herokuapp.com/#/basket

## **4.3 Proof of concept.**

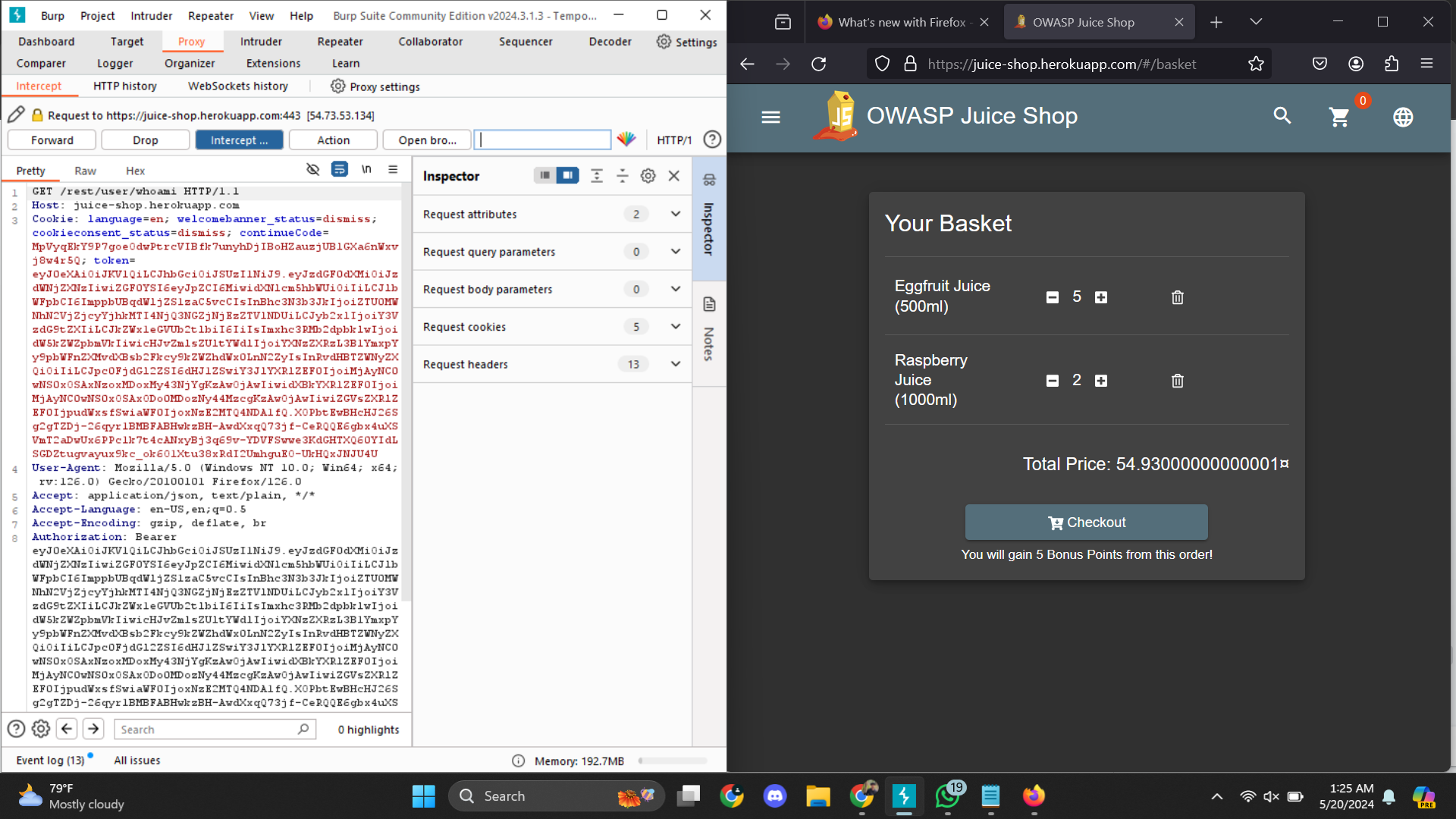
**Step 1:** In the web app, there is a cart option. When we access that option ,using burp suite we can see the id is passing in plain text .

In this case the id is “2”

**Step 1:** we can change the id to another number at that time we can see others data.



We alter the request then we get this result.



## **4.4 Mitigation.**

**Encrypt User IDs:** Implement encryption mechanisms to protect user IDs stored in databases or transmitted over the network. Use strong encryption algorithms and ensure secure key management practices.

# **5.HTTP parameter Pollution**

## **5.1 Description**

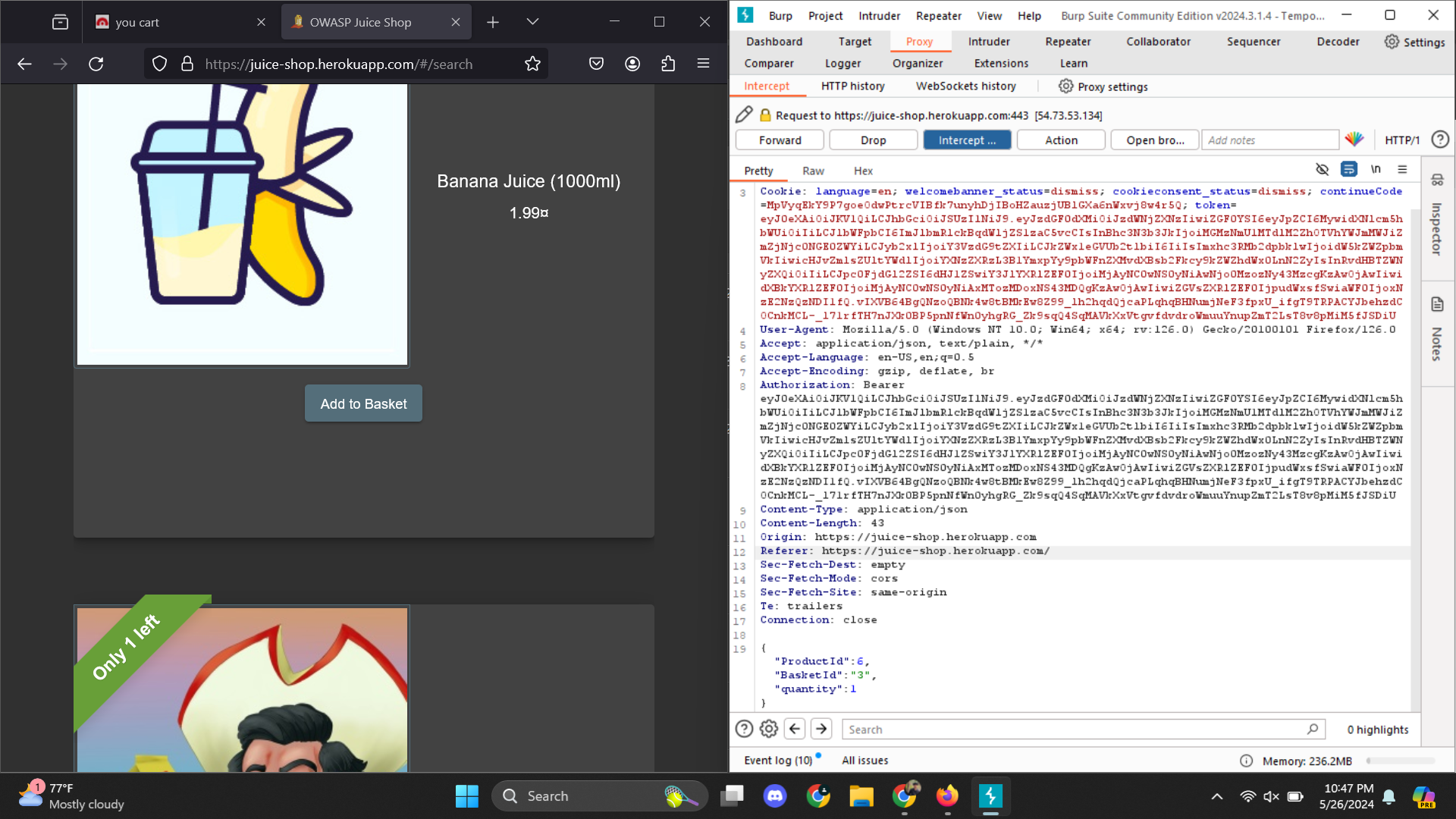
In the web application there is an option to add products to cart .In this option there is a vulnerability. When we add items in the cart , in the request the number of quantity and id are passed in plain text .So we can edit the quantity to negative value and we can see that the price is negative .

## **5.2 Vulnerable instance**

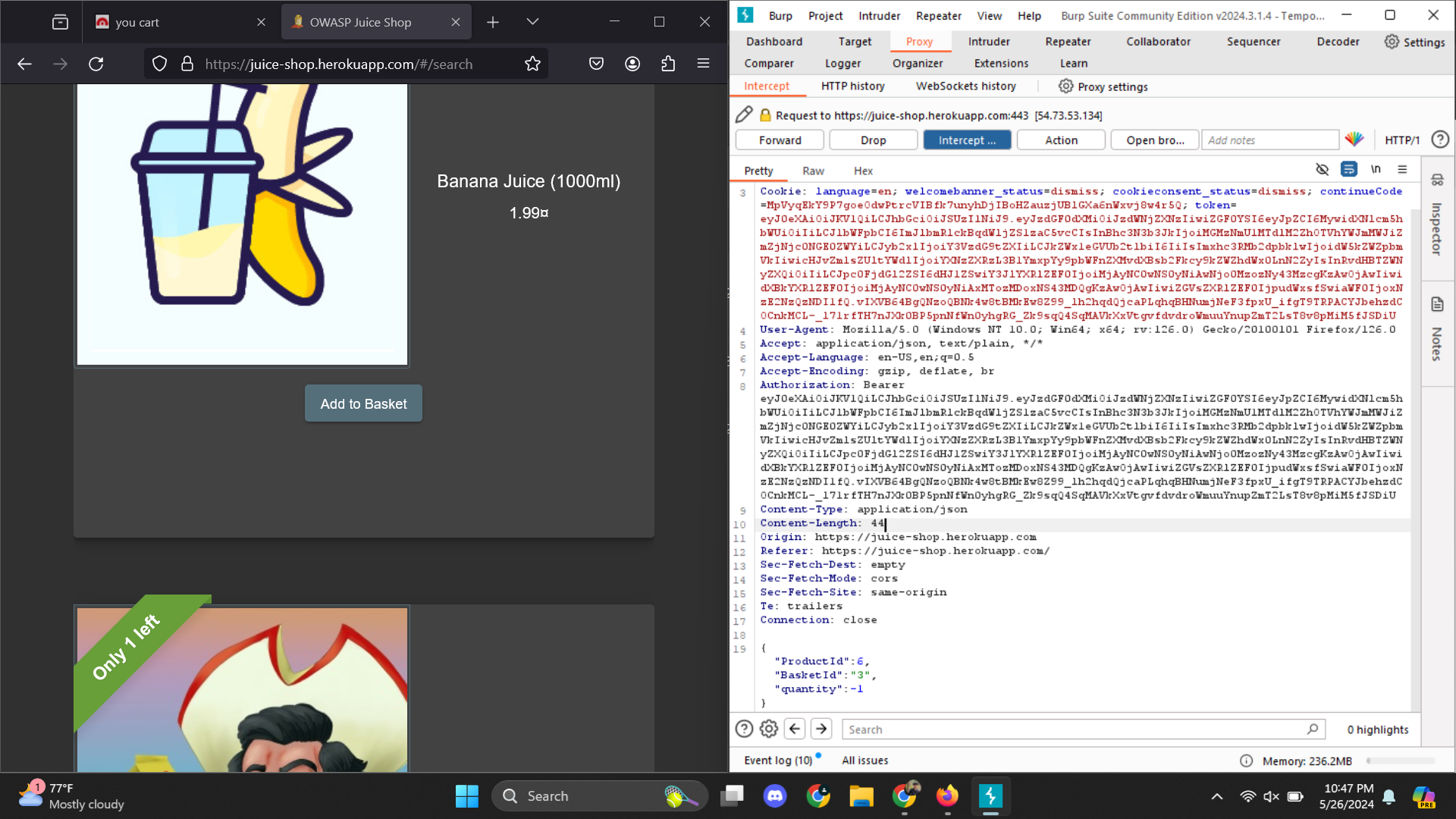
https://juice-shop.herokuapp.com/#/basket

## **5.3 Proof of concept.**

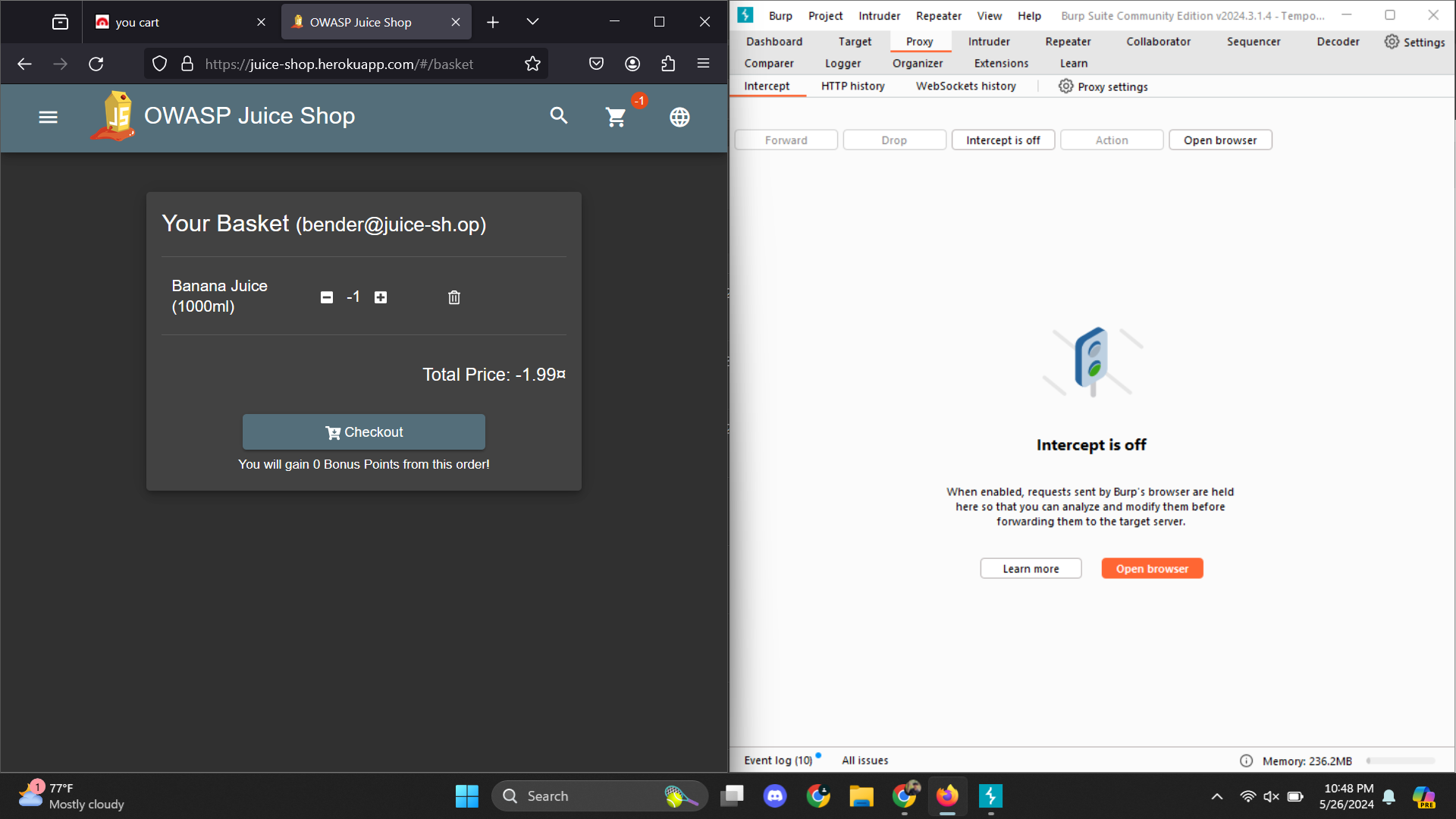
We can see that the request, the number of quantity and id are passed in plain text .



We can edit the quantity to negative value



We can see that the price of the item is negative.



## **5.4 Mitigation**

**Tokenization:** Generate unique tokens for each session/cart encapsulating price and product ID.Decode tokens on the server to retrieve original values.

**Input Validation and Sanitization:** Strictly validate and sanitize all incoming data to adhere to expected formats and values.