SWAGGER PET SHOPVAPT REPORT

short line

SANJAY S KUMAR

# 

**INDEX**

**I : Summery**

**II : Tools used**

**III : vulnerabilities lists.**

**Summery**

This document presents a comprehensive security assessment of the swagger pet shop api. The primary purpose of this assessment is to thoroughly identify and analyze the existing vulnerabilities within the api and evaluate the associated security risks. Through this detailed evaluation, the document aims to provide a clear understanding of the current security posture of the swagger pet shop api.

The assessment process involved a meticulous examination of the api to uncover various vulnerabilities. Each identified vulnerability is documented in detail within this report, highlighting the potential threats they pose.The document outlines specific mitigation strategies to address and remediate these vulnerabilities effectively. By implementing these mitigation measures, the security of the api can be significantly enhanced, reducing the risk of exploitation by malicious actors.

**Scope :** <https://petstore.swagger.io/#>

**Tools used**

The tool used for almost every assessment is Burp suite community edition and chrome browser.Burp Suite is a popular and powerful tool used for web application security testing. It is widely utilized by security professionals, penetration testers, and ethical hackers to identify and exploit vulnerabilities in web applications.

**vulnerabilities lists**

| **SI NO** | **Vulnerability** | **Severity** |
| --- | --- | --- |
| **1** | **broken authentication** | **8.9** |
| **2** | **Improper validation .** | **8.5** |
| **3** | **weak password policy** | **8.3** |
| **4** | **Duplicate user creation** | **9.3** |
| **5** | **url not encrypted** | **7.2** |
| **6** | **broken authorization** | **8.9** |
| **7** | **Lack of rate limiting** | **9.2** |
| **8** | **Broken Authorization in delete option** | **8.7** |
| **9** | **File upload vulnerability.** | **8.9** |
| **10** | **HTTP parameter Pollution** | **9.1** |
|  |  |  |

# **1.broken authentication**

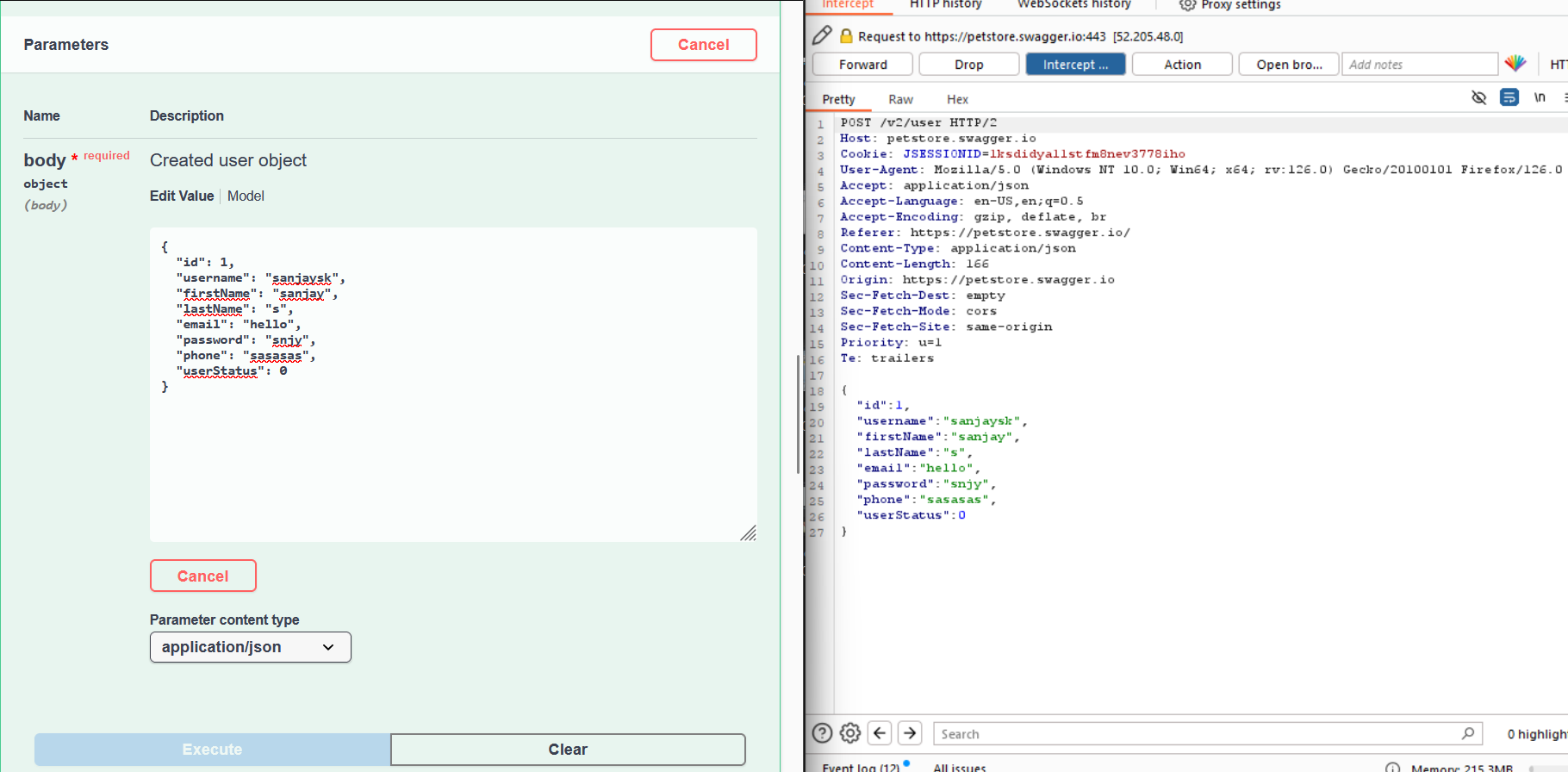
## **1.1 Description**

This API contains a potential security weakness. Users are able to modify the 'id' field. Normally, IDs function as unique and unchanging references within a system. However, this API grants users the ability to alter these identifiers. This vulnerability could be exploited by malicious actors to manipulate data or gain unauthorized access.

## **1.2 Vulnerable instance**

<https://petstore.swagger.io/#/user/createUser>

## **1.3 Proof of concept.**

**Step1 :** go to user creation ,then add details of the user .We can see that the id is on the user side so the user can easily change the user id. In this a change the id to ‘1’.

## **1.4 Mitigation**

**Make IDs non-editable:** The most secure approach is to prevent users from modifying the 'id' field altogether.

**Implement strong authorization checks:** If editing IDs is a necessary function, enforce strict authorization rules.

**Validate ID changes:** If users must edit IDs, implement validation checks. Ensure new IDs are unique and don't conflict with existing ones.

# **2.Improper validation .**

## **2.1 Description**

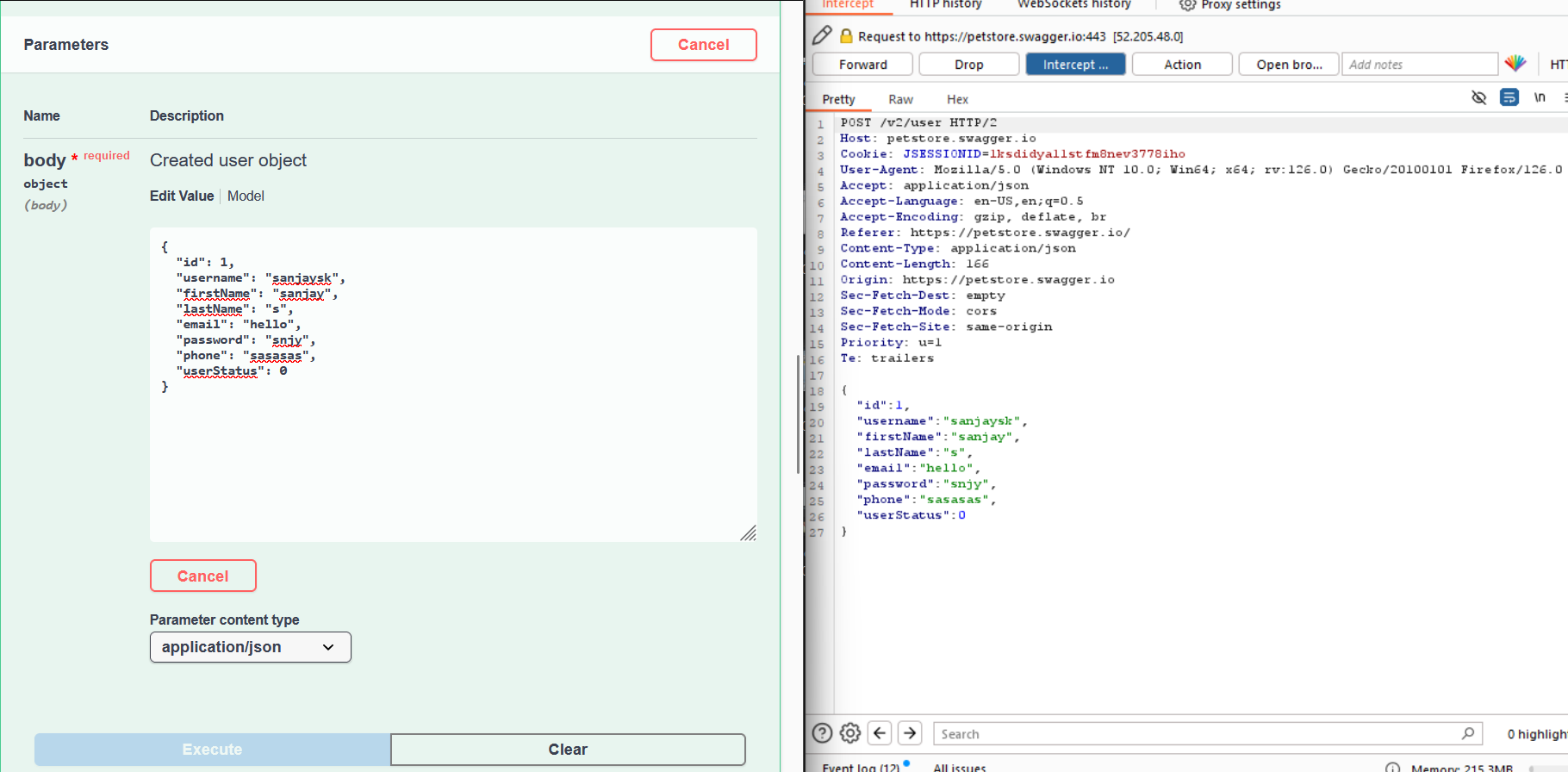
This API suffers from a critical security issue: it doesn't validate user input. Specifically, the phone number field appears vulnerable to accepting characters other than valid digits. Malicious actors could exploit this weakness by injecting special characters or exceeding the expected length limit.

## **2.2 Vulnerable instance**

<https://petstore.swagger.io/#/user/createUser>

## **2.3 Proof of concept.**

Go to the user creation page and add the user details .we can see that there is no validation or verification in this api.



## **2.4 Mitigation**

**Data Type Checks:** Define the phone number field as a numerical data type. This automatically restricts entries to only numbers.

**Length Limits:** Set a maximum length for the phone number field to prevent exceeding valid phone number formats.

**Provide clear and specific error messages** when users enter invalid data. This helps them understand the issue and correct their input.

# **3.week password policy**

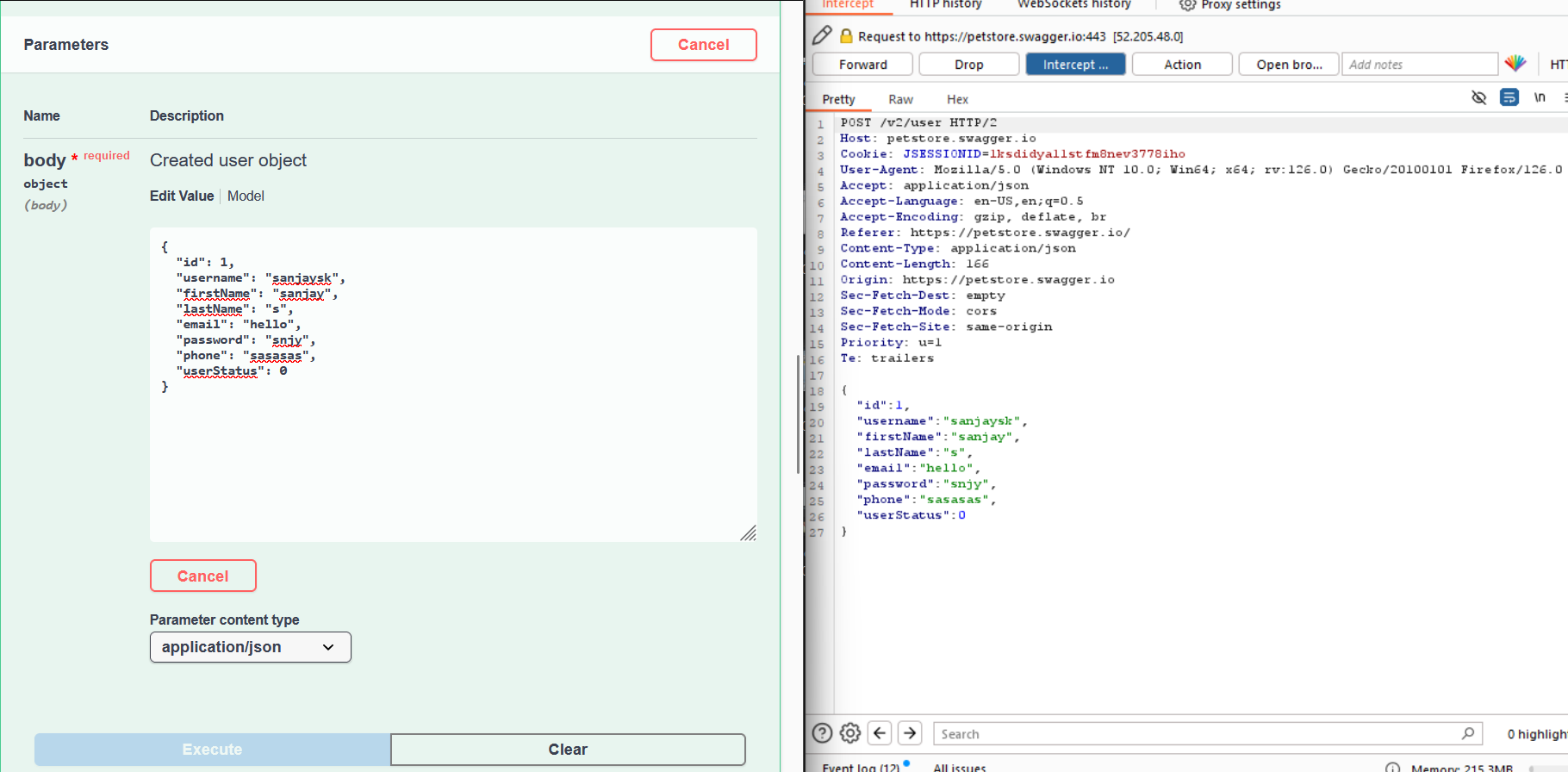
## **3.1 Description**

A critical security concern exists with the api registration form. The form currently lacks a password policy, meaning there are no enforced requirements for users to create strong passwords. This is problematic because weak passwords are easily guessed or cracked by hackers, putting user accounts and potentially sensitive data at risk.

## **3.2 Vulnerable instance**

https://petstore.swagger.io/#/user/createUser

## **3.3 Proof of concept.**

First go to the user creation page ,in this page there is no password policy .We can add a simple password.

## **3.4 Mitigation**

**Minimum Length:** Set a minimum password length, ideally 12 characters or more. Longer passwords are exponentially harder to crack through brute-force attacks.

**Implement Multi-Factor Authentication (MFA):**MFA adds an extra layer of security beyond just a password. It requires users to provide a second verification factor.

# **4. Duplicate user creation**

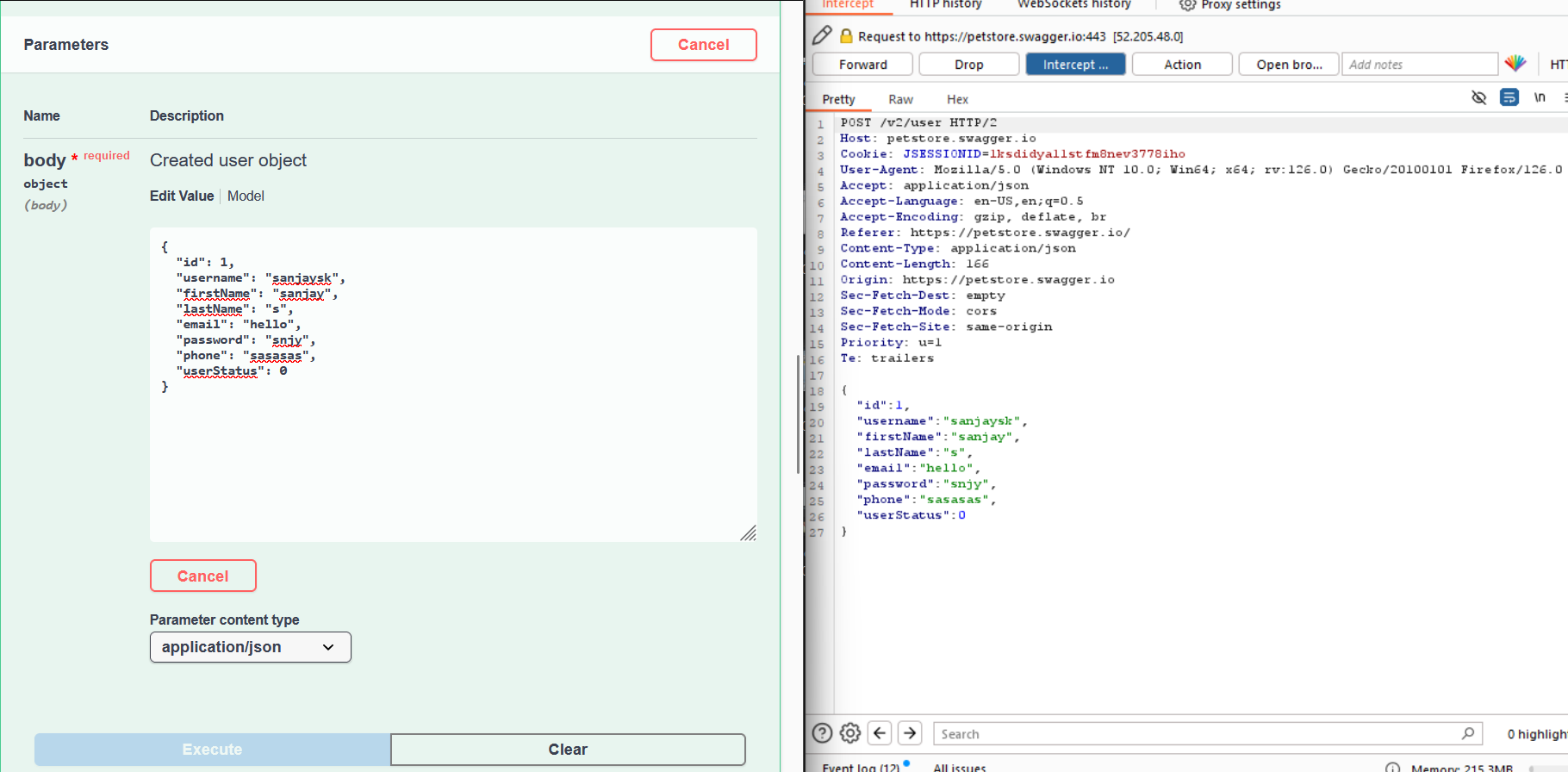
## **4.1 Description**

This API contains a security weakness related to username management. The system allows users to register with duplicate usernames. Ideally, usernames should be unique identifiers to distinguish between different accounts. However, in this case, multiple users could potentially share the same username.

**4.2 Vulnerable instance**

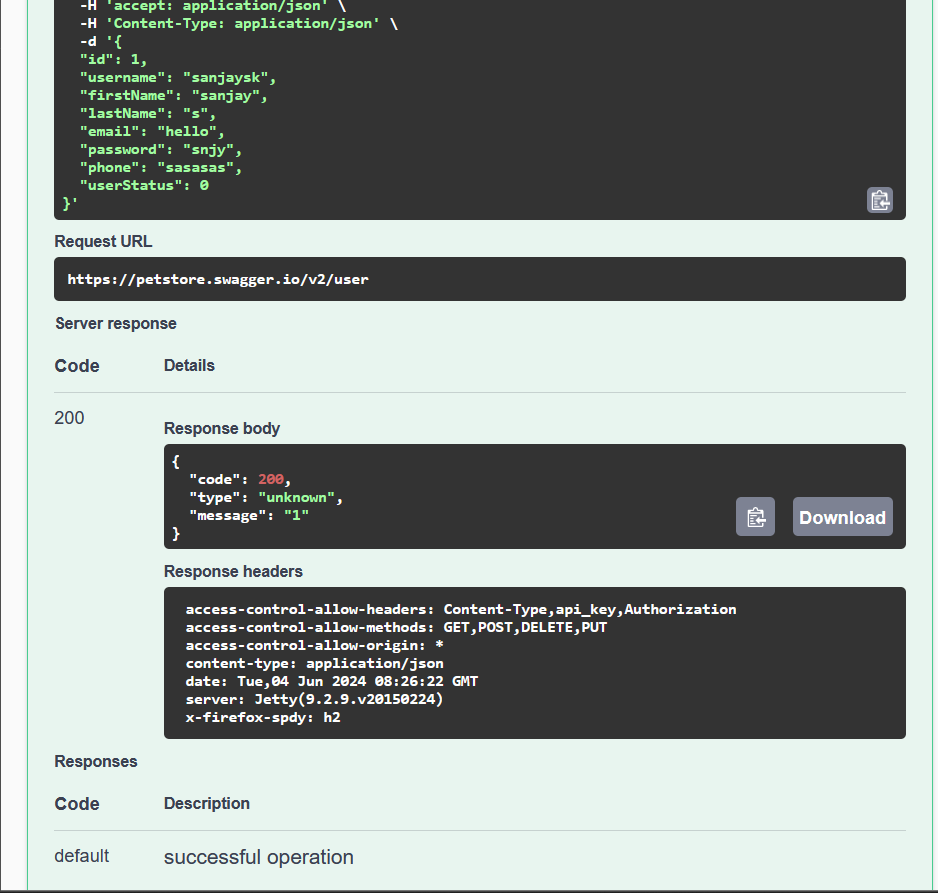
<https://petstore.swagger.io/#/user/createUser>

## **4.3 Proof of concept.**

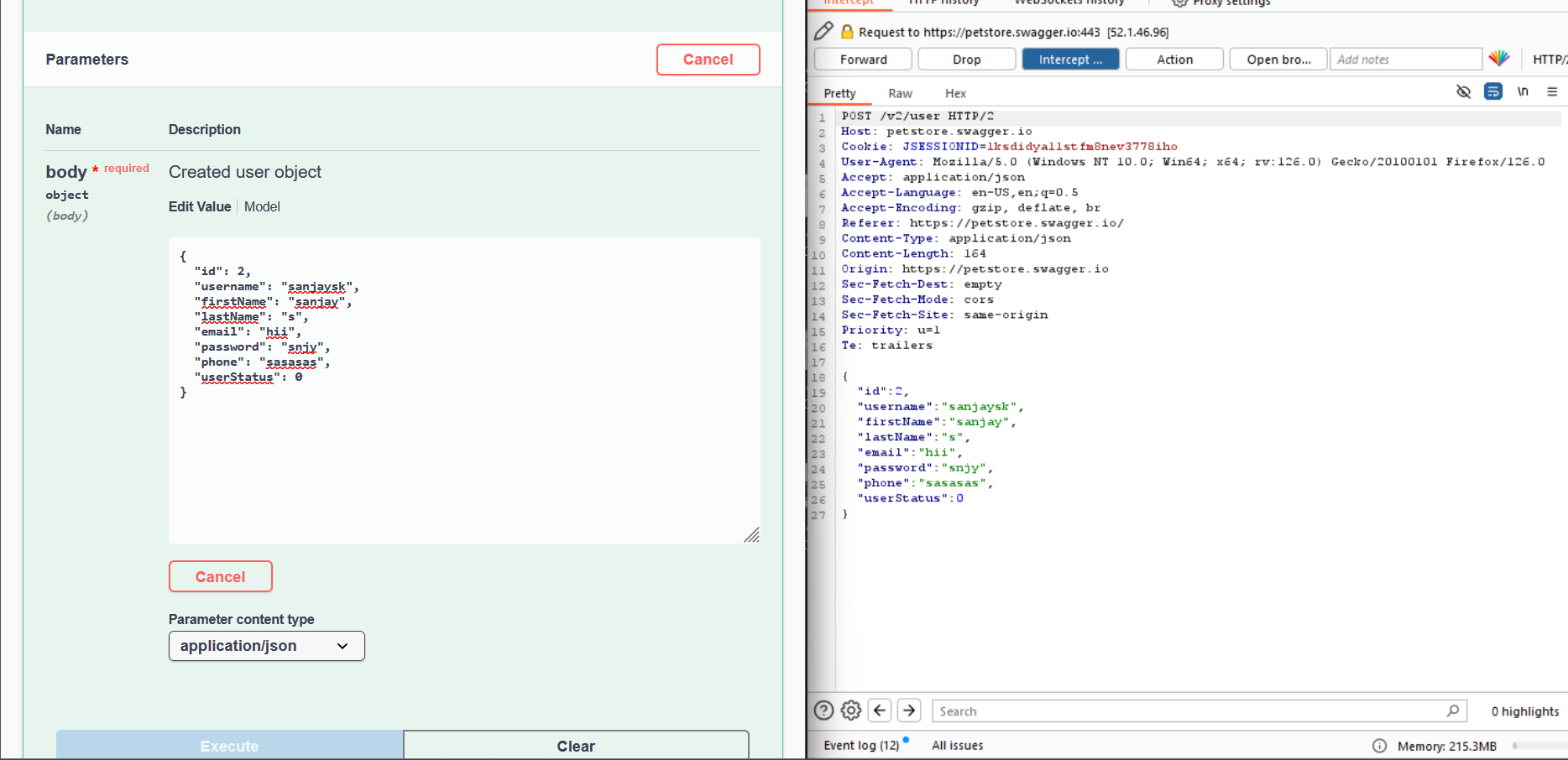
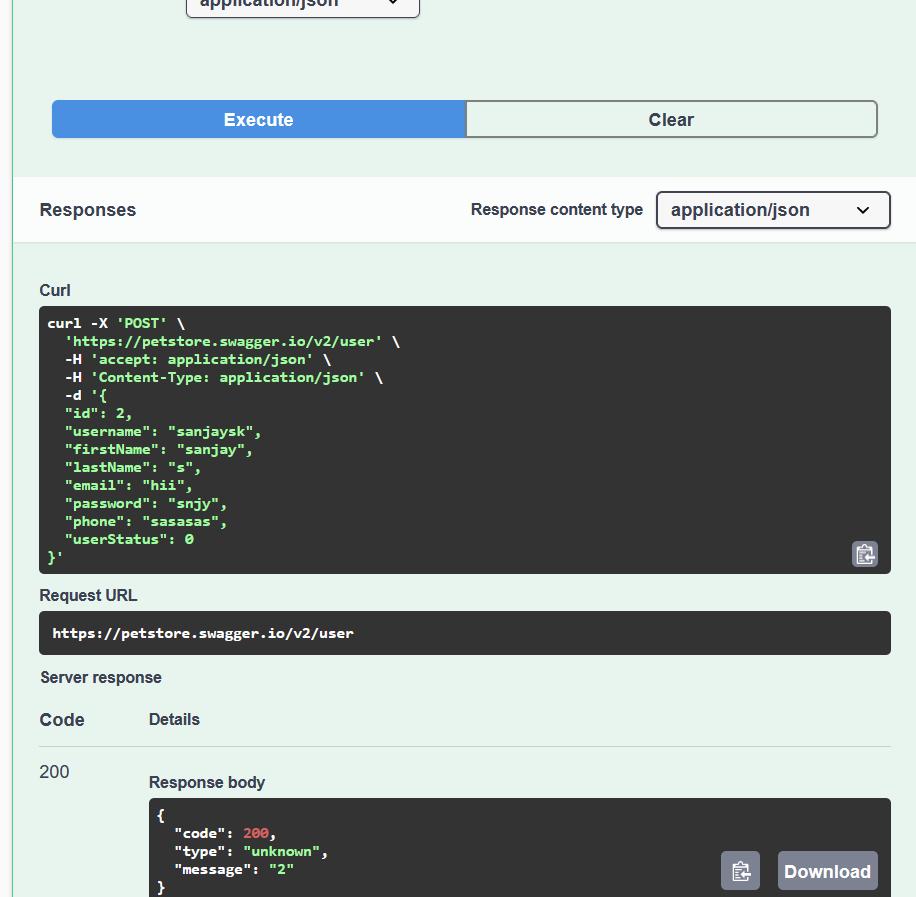
step 1: go to the user creation page . then create a user .I used the username “sanjaysk”.

We can output , then the user is created successfully.

Output:



**Step 2:** create another user with same user user name “sanjaysk”



We can see the second user is also created using the same username.

## **4.4 Mitigation**

**Database Constraints:** The most common solution is to implement a unique constraint on the username field within your database. This ensures the database physically cannot store duplicate usernames.

**Email-Based Login:** Consider offering alternative login methods besides usernames. Email addresses are generally considered more unique than usernames and can be used for authentication.

# **5.url not encrypted**

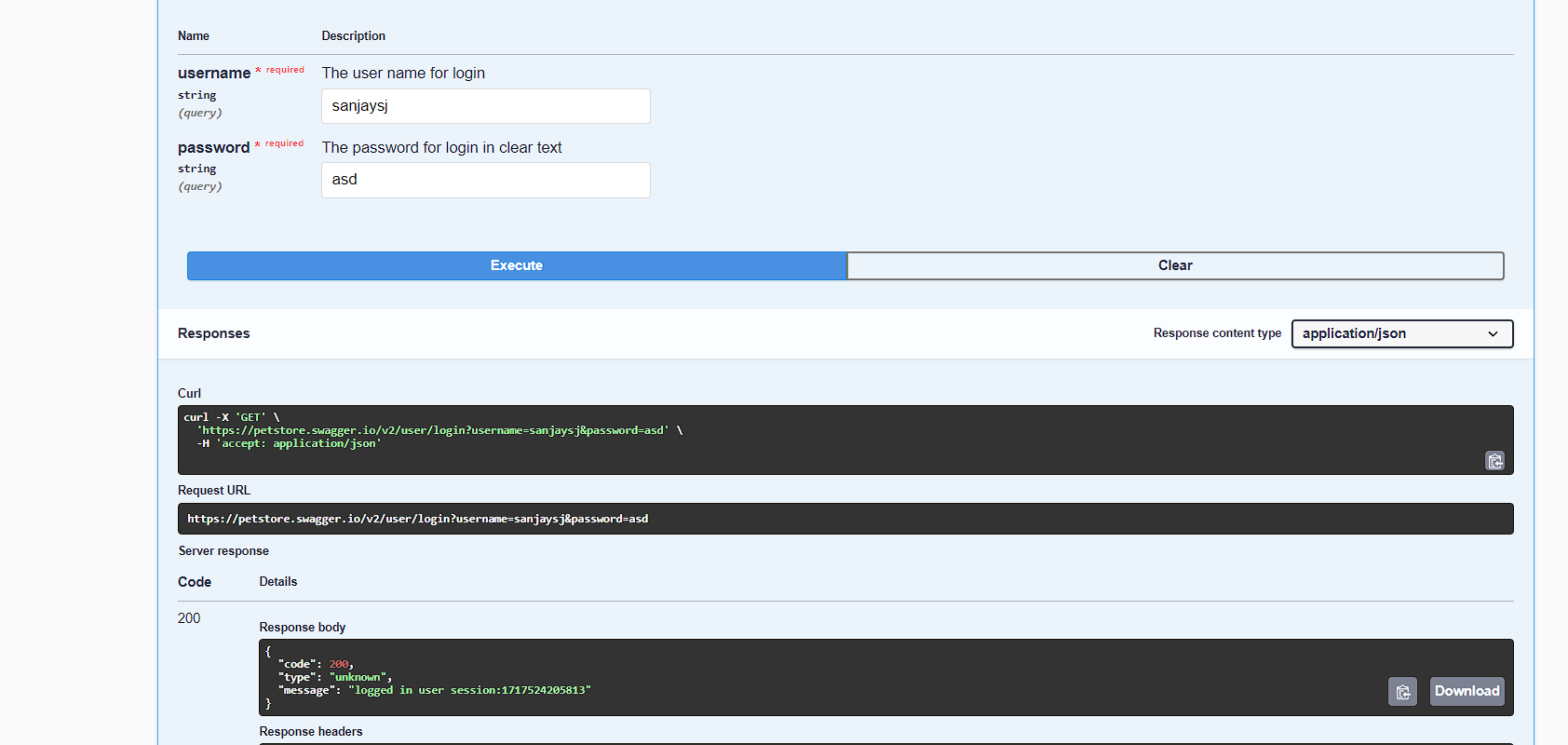
## **5.1 Description**

This API has a critical security flaw: usernames and passwords are transmitted in plain text during the login process. Ideally, login credentials should be encrypted to ensure confidentiality. However, in this case, anyone intercepting the communication between the user and the API could potentially steal the login information

## **5.2 Vulnerable instance**

<https://petstore.swagger.io/#/user/loginUser>

## **5.3 Proof of concept.**

Step 1: go to the login page then enter the username and password . we can see the username and password in the url .

## **5.4 Mitigation**

**Implement Secure Transport Layer (TLS):**This is the most crucial step. Ensure all communication between the user and your API utilizes TLS (formerly known as SSL).

**Hash Passwords:**Even with TLS, it's best practice to never store passwords in plain text on the server. Instead, implement a secure password hashing algorithm. This algorithm transforms the password into a unique string that cannot be easily reversed back into the original password.

**Two-Factor Authentication (2FA):** Consider adding an extra layer of security with 2FA.

# **6.broken authorization**

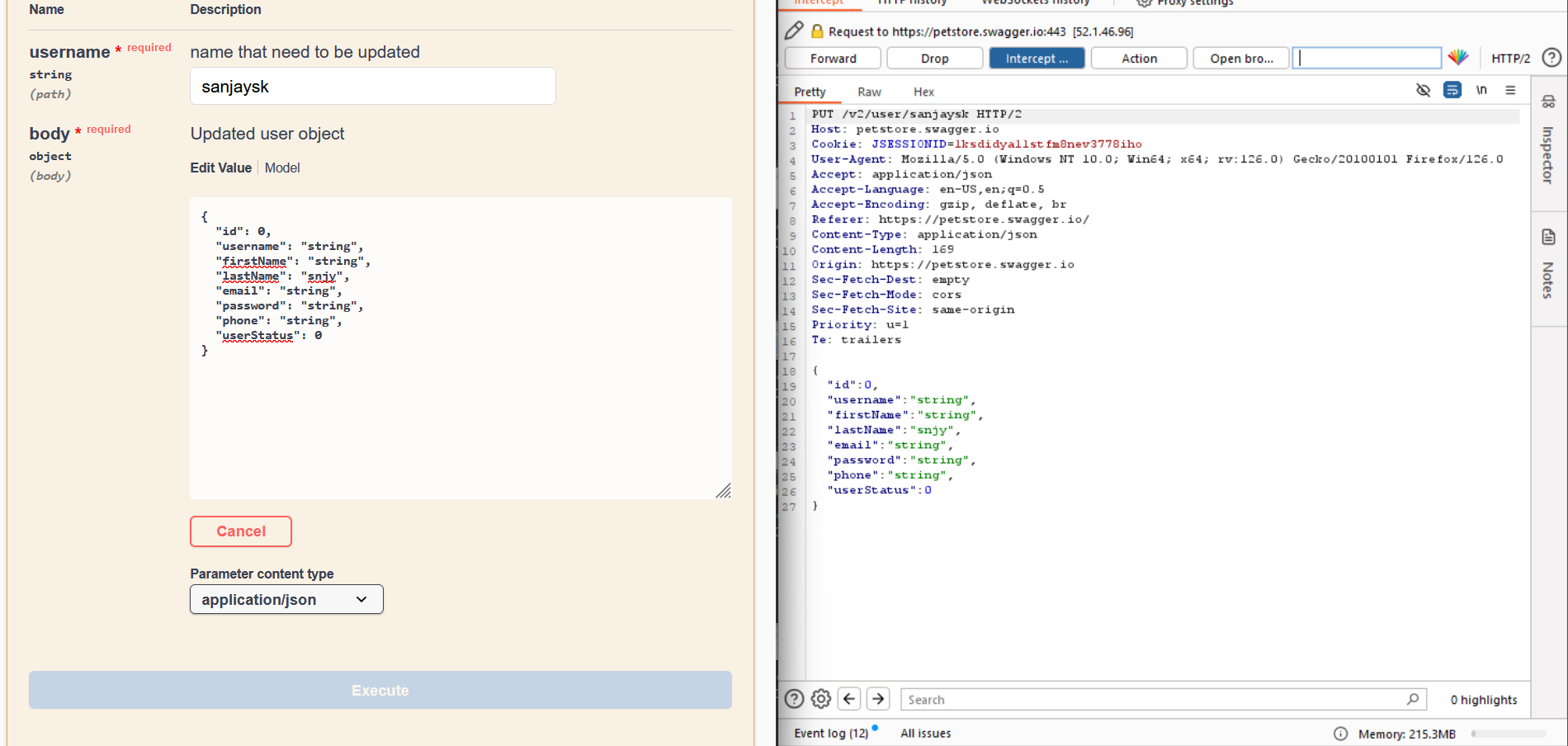
## **6.1 Description**

This API contains a critical security weakness. It offers an option to retrieve user details, but this functionality lacks proper authorization checks. Ideally, access to user information should be restricted based on user roles and permissions. However, in this case, anyone can potentially exploit this option to find and view user details. This could lead to a privacy breach, exposing sensitive user information to unauthorized individuals

## **6.2 Vulnerable instance**

<https://petstore.swagger.io/#/user/getUserByName>

## **6.3 Proof of concept.**

First go to the user finding page ,then search about the user .When intercept the request then we can easily change the username to another . 

## **6.4 Mitigation**

**Role-Based Access Control (RBAC):** This is a common approach where users are assigned roles with specific permissions.

**API Logging and Monitoring:**Implement robust logging and monitoring for API requests, especially those involving access to user details. This allows you to track who accessed what data and identify any suspicious activity.

# **7.Lack of rate limiting**

## **7.1 Description**

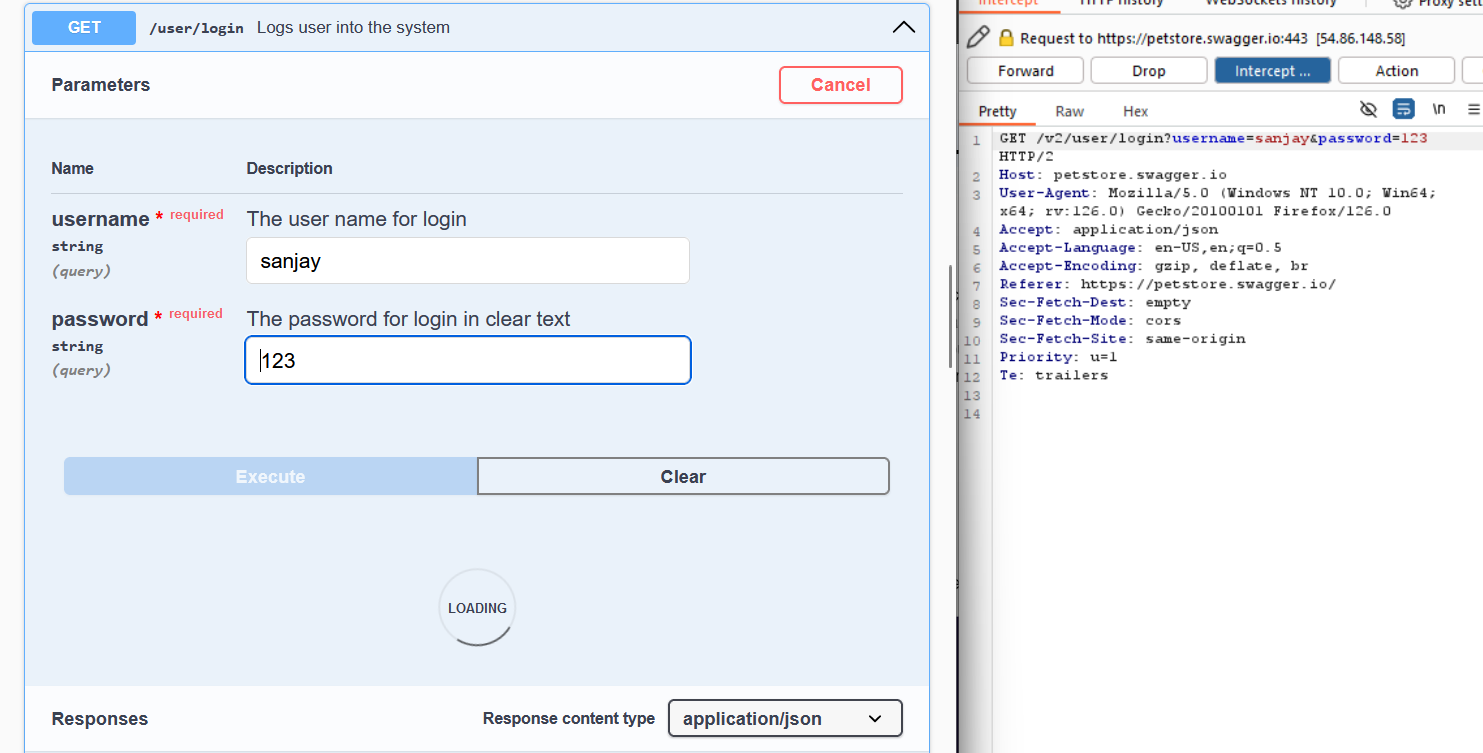
This API contains a critical security weakness.This API's hourly rate limit is too generous. It allows for DoS attacks and hinders performance. Consider lowering it to minutes or seconds for better security and efficiency.

## **7.2 Vulnerable instance**

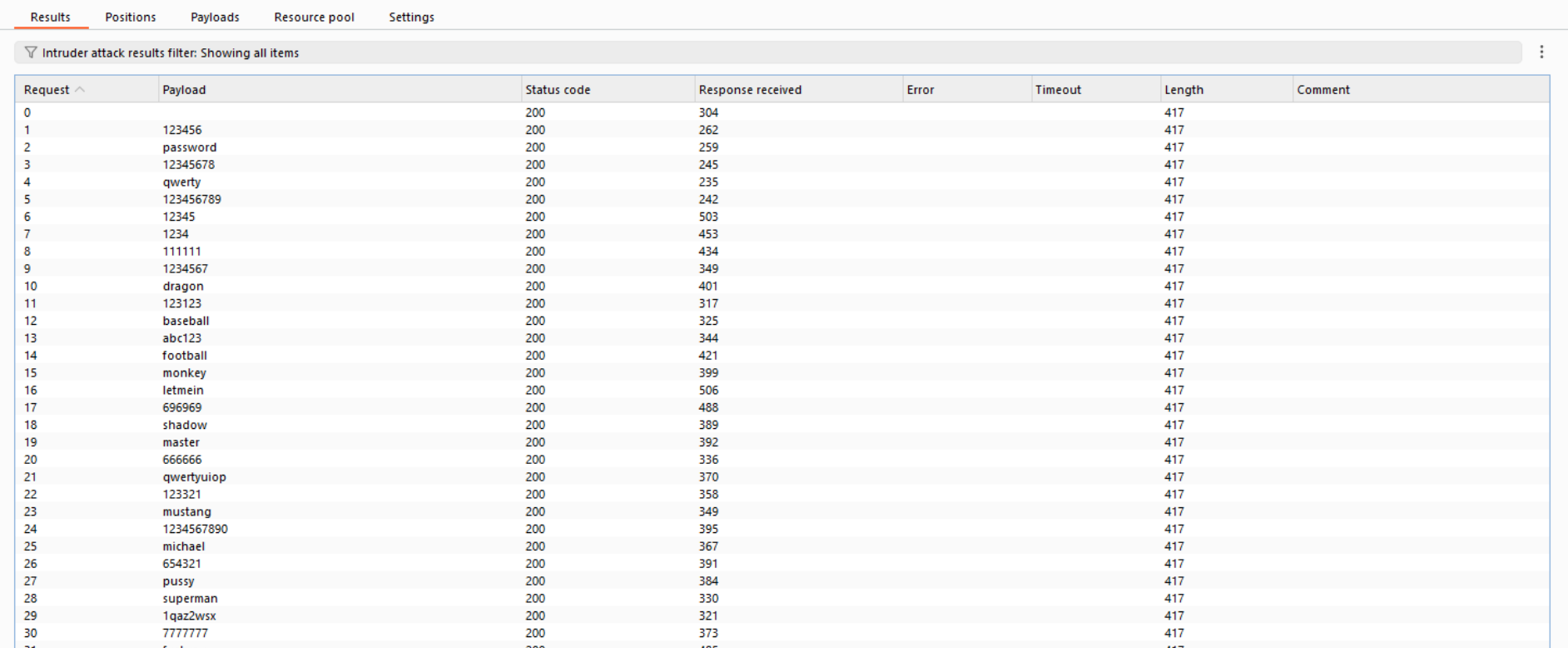
<https://petstore.swagger.io/#/user/loginUser>

## **7.3 Proof of concept.**

**Step 1 :** go to the login, add password and username and capture the request .



**Step 2:** transfer the request to the intruder and start an attack. After starting the attack we can see that the request made and the response getting 200 OK for all the request that is sent through the intruder. Every request is getting a 200 OK so we can confirm it’s vulnerable to brute force.



## **7.4 Mitigation**

**Reduce Rate Limit Window:** The most straightforward approach is to shorten the time window for the rate limit.

**Token Bucket Algorithm:** Consider using a token bucket algorithm for rate limiting. This approach allocates users a set number of tokens that replenish over time. Each API request consumes a token.

# **8. Broken Authorization in delete option**

## **8.1 Description**

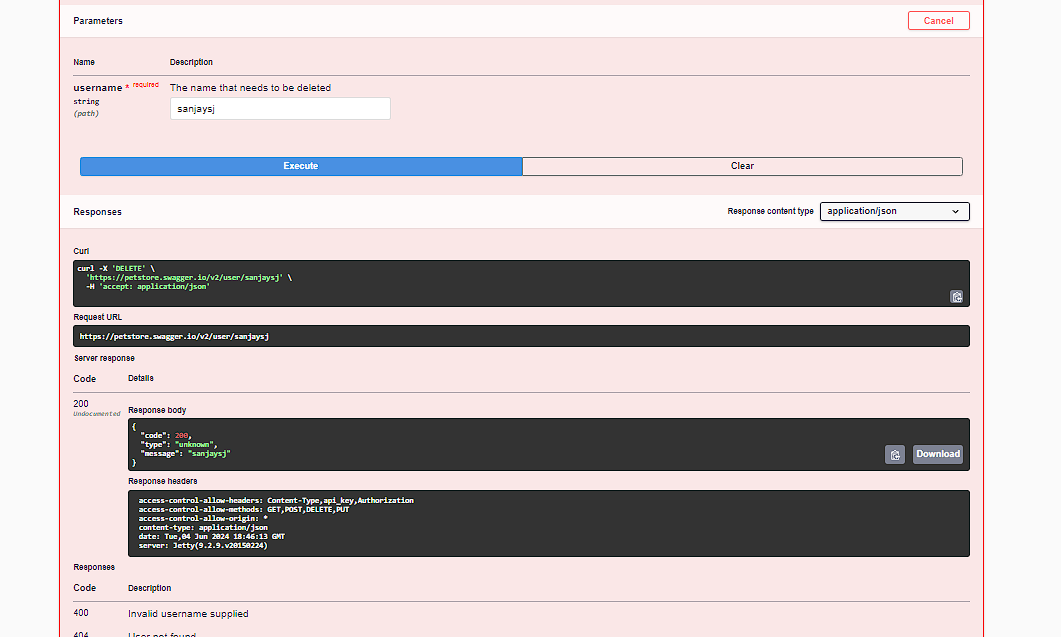
This API contains a critical security flaw: it allows users to delete user data without any form of authorization. Ideally, deleting user data should be a highly restricted action requiring strong authentication. However, in this case, anyone with access to the API endpoint could potentially delete any user's data. This vulnerability could be exploited by malicious actors to cause significant damage, permanently deleting user information.

## **8.2 Vulnerable instance**

<https://petstore.swagger.io/#/user/createUser>

## **8.3 Proof of concept.**

In this api there is an option to delete the user . at that option there is no validation and not asking for password . an attacker can easily delete any other user using username .



## **8.4 Mitigation**

**Strong Password Checks:** Enforce strong password requirements during user registration and login. This discourages weak passwords that could be easily guessed or stolen.

**Multi-Factor Authentication (MFA):** Consider implementing multi-factor authentication (MFA) for critical actions like user data deletion.

**Role-Based Access Control (RBAC):** Implement RBAC to restrict user data deletion permissions.

# **9.File upload vulnerability.**

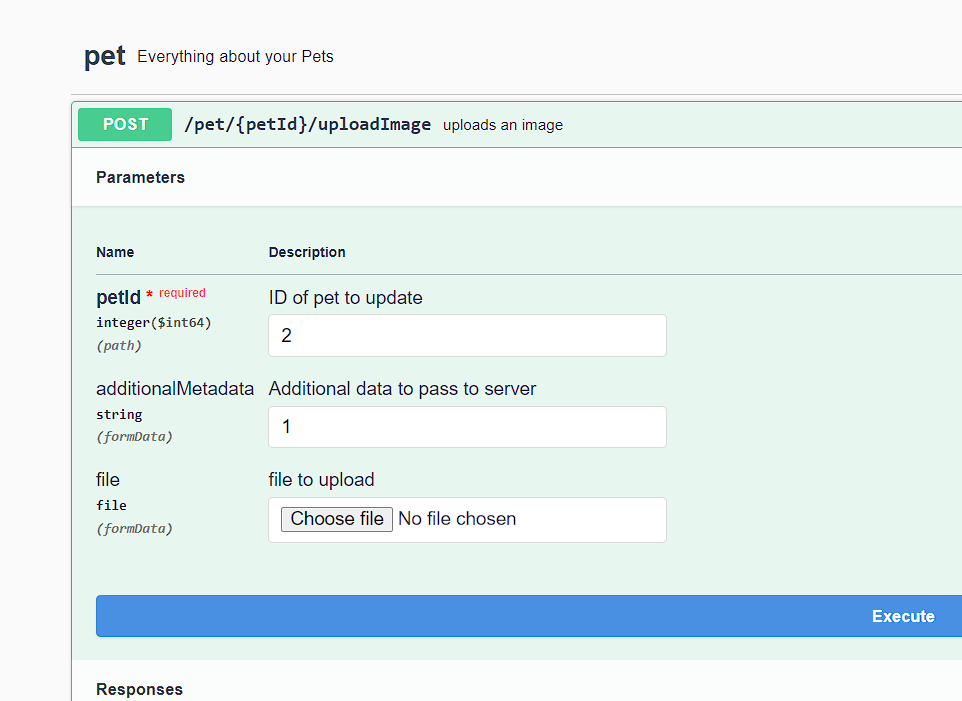
## **9.1 Description**

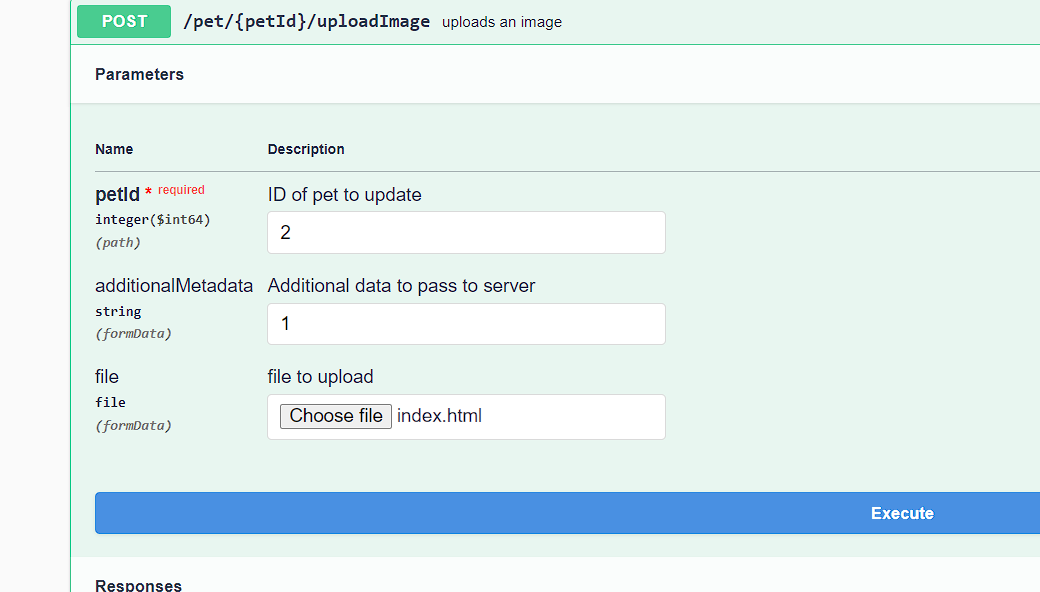
There's a security concern with the pet photo upload feature in this API. The API doesn't validate the files being uploaded, which means users can upload any type of file instead of just images of their pets. This poses a potential security risk. Malicious users could upload files containing viruses or other harmful code that could compromise the system or even spread to other users.

## **9.2 Vulnerable instance**

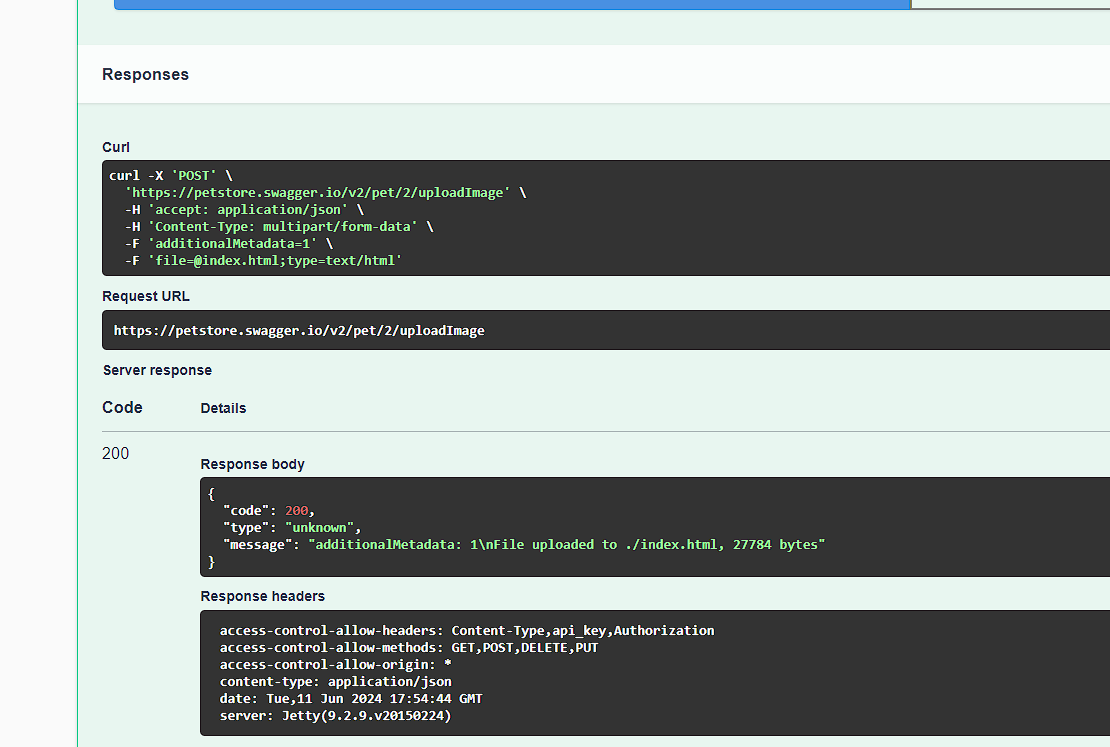
<https://petstore.swagger.io/#/pet/uploadFile>

## **9.3 Proof of concept.**

**Step 1:** Go to the file uploading page , we can see there is an option to add photo.

Step 2 : then add another file like html etc…

Then check the response



In the response , we can see the file is executed on the server.

## **9.4 Mitigation**

**File Extension Checking:** While less secure than MIME type checking, validate the file extension (e.g., .jpg, .png). This can prevent some unexpected file types but may be bypassed by renaming malicious files with image extensions.

**Image Format Verification:** Use libraries to verify the file contents beyond the extension or MIME type. This ensures the uploaded data actually represents a valid image format.

**Limit File Size:** Set a maximum file size limit to prevent users from uploading excessively large files that could overload the server or storage.

# **10.HTTP parameter Pollution**

## **10.1 Description**

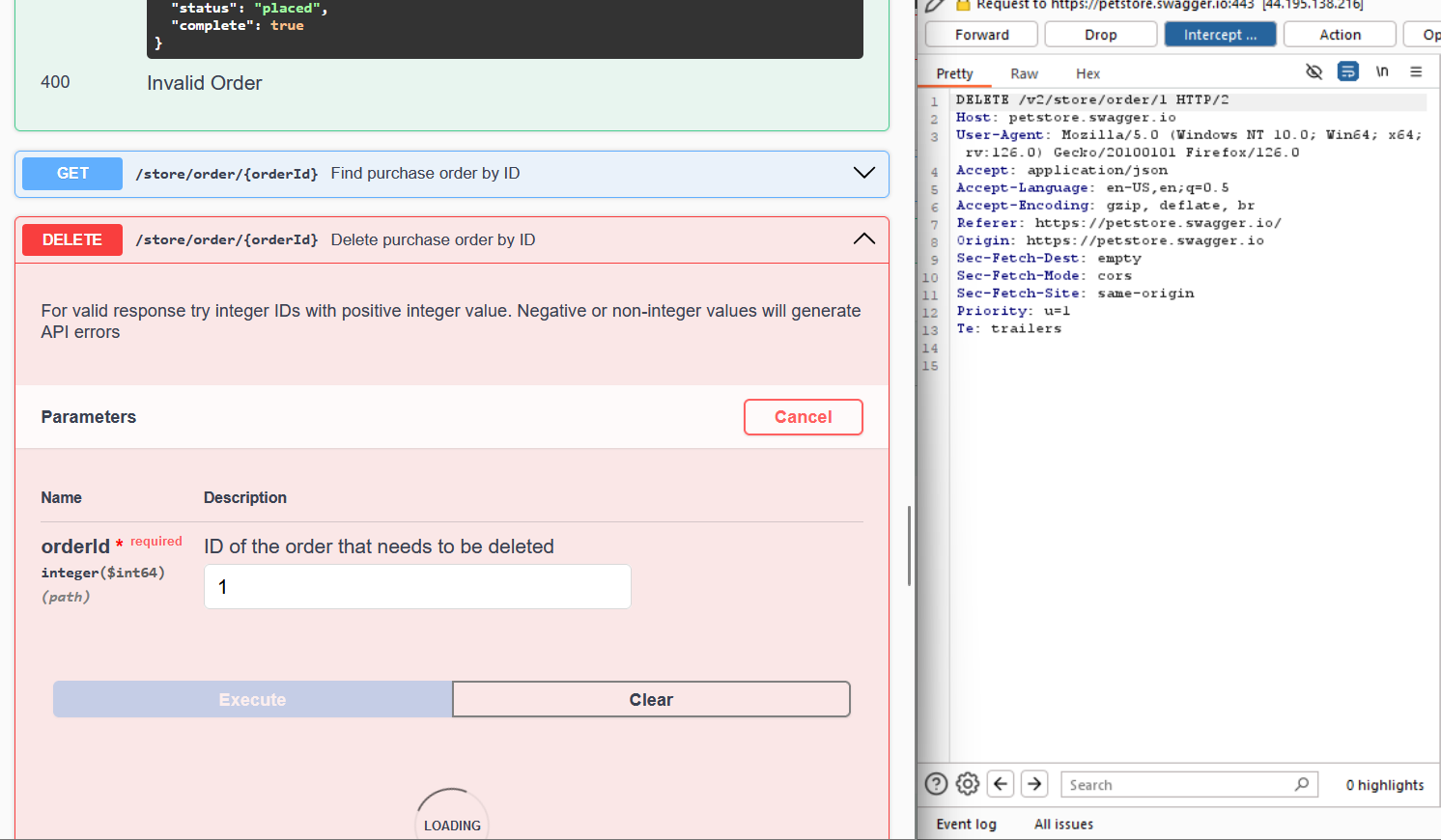
The delete order functionality within this API contains a security vulnerability. When sending a request to delete an order, the order's identification number (ID) is included in plain text within the request itself. This lack of validation allows anyone to tamper with the ID in the request.

## **10.2 Vulnerable instance**

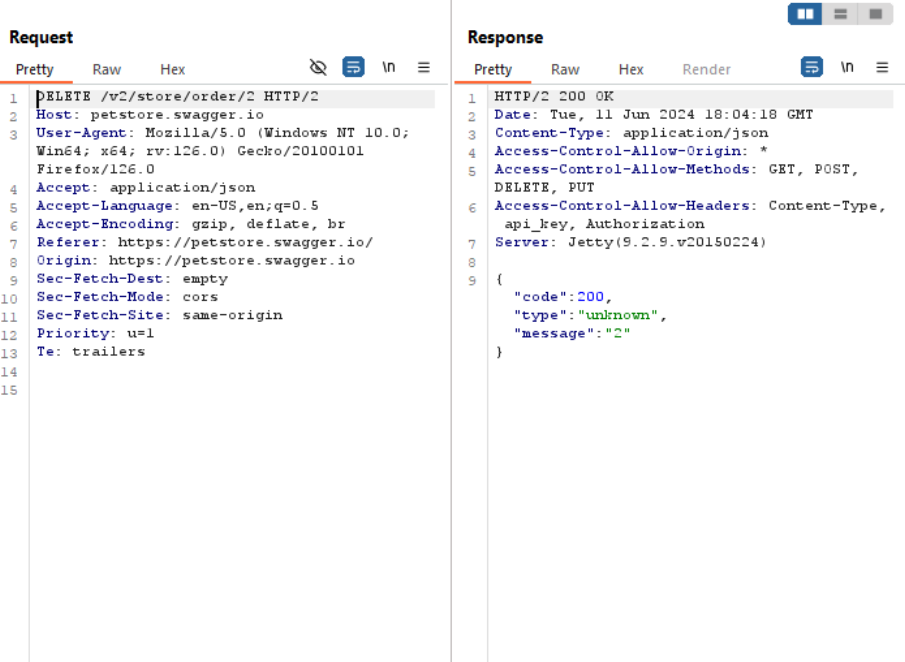
<https://petstore.swagger.io/#/store/deleteOrder>

## **10.3 Proof of concept.**

Go to the delete option then capture the request . we can see the id is in plain text.



Then edit the id to another .I change the id to ”2”.



We can see, we get 200 ok response .

## **10.4 Mitigation**

**Validate Order ID on Server:** Even if you implement secure transmission, always validate the order ID on the server-side before processing the deletion request. This ensures the received ID is a valid order within the system and prevents unauthorized attempts to delete other users' orders.

**Verify User Permissions:** Implement authorization checks to ensure the user making the delete request has the necessary permissions to delete the specific order.

**Use HTTPS:** Ensure the API uses HTTPS (Hypertext Transfer Protocol Secure) for communication. HTTPS encrypts data transmission between the client and server, making it much harder to intercept and modify the order ID in plain text.