Demoblaze storeVAPT 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 demoblaze store web application. The primary purpose of this assessment is to thoroughly identify and analyze the existing vulnerabilities within the web application 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 demoblaze store web application.

The assessment process involved a meticulous examination of the web application 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 web application can be significantly enhanced, reducing the risk of exploitation by malicious actors.

**Scope :**<https://www.demoblaze.com/index.html>

**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** | **Lack of rate limit** | **9.2** |
| **2** | **Weak password policy** | **8.3** |
| **3** | **Broken authentication** | **8.9** |
| **4** | **Cryptographic failure** | **8.3** |
| **5** | **Session management** | **6.8** |
| **6** | **Security misconfigured** | **9.1** |
| **7** | **Default credentials** | **9.2** |
| **8** | **IDOR** | **9.2** |
| **9** | **Cross Origin Resource Sharing (CORS)** | **8.1** |

# **1.Lack of rate limiting**

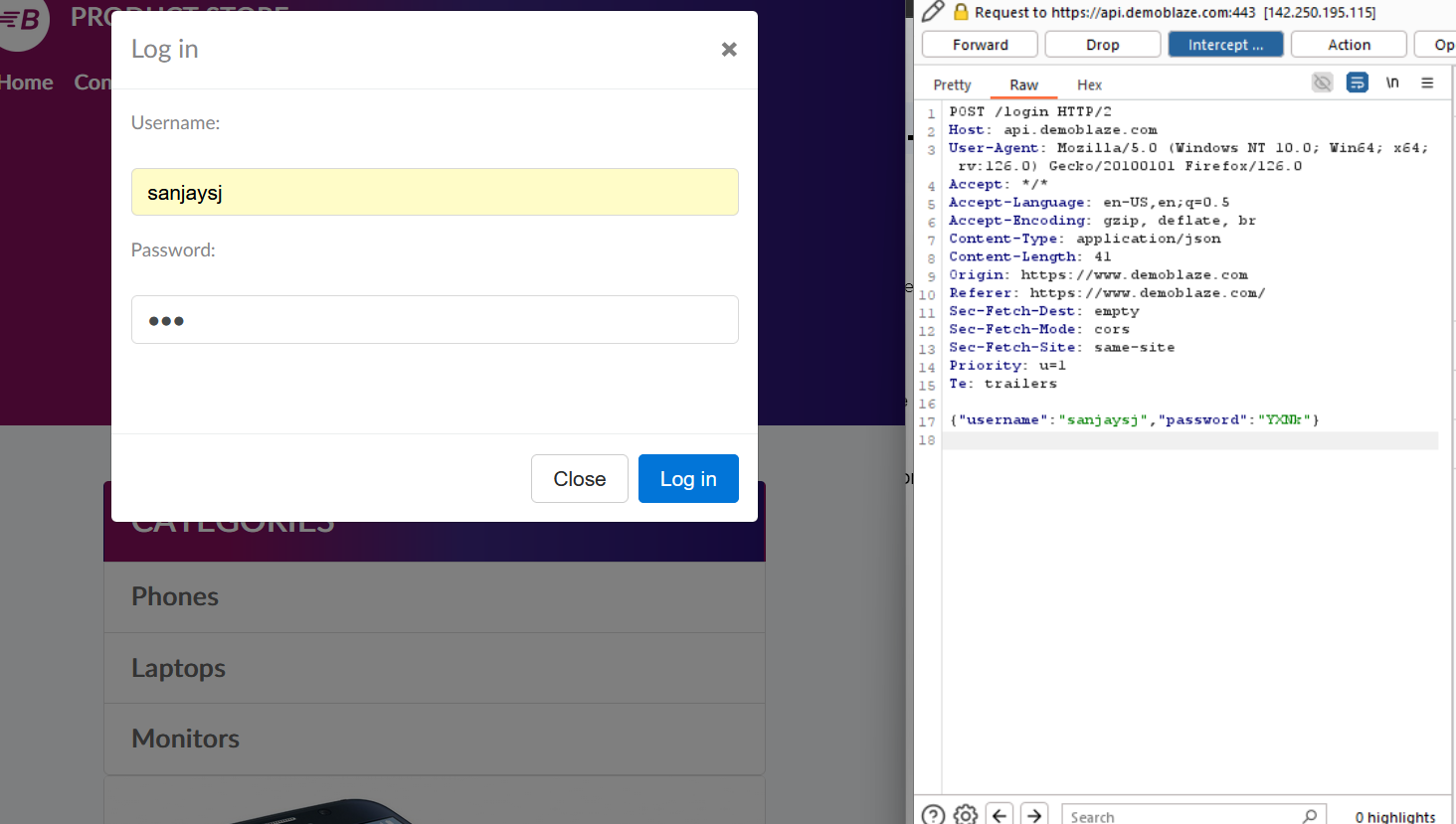
## **1.1 Description**

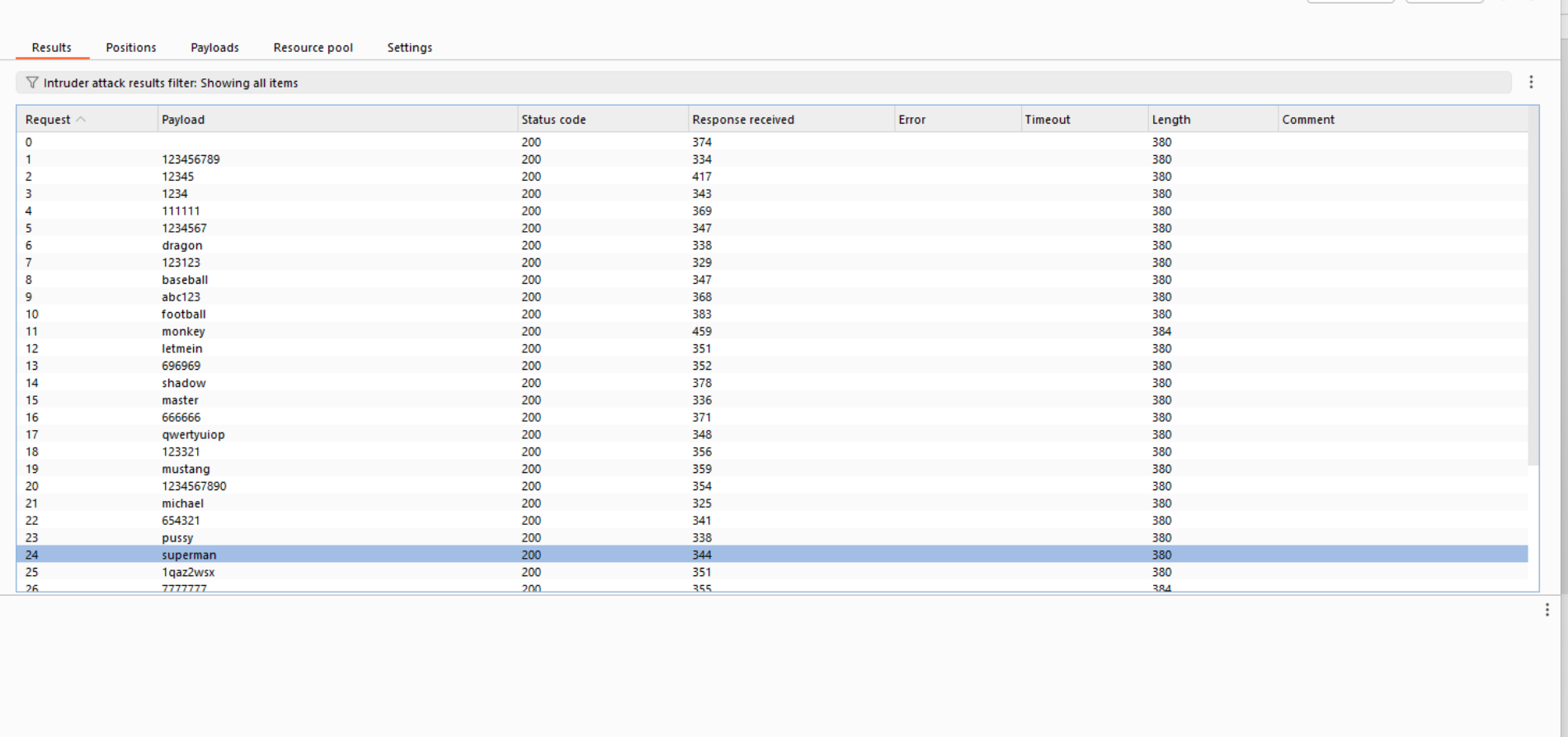
The website's login page appears to lack a security measure called "rate limiting." This feature restricts the number of login attempts a user can make within a specific timeframe. Without rate limiting, an attacker could exploit a vulnerability by repeatedly trying to guess a user's password through a "brute-force attack."

## **1.2 Vulnerable instance**

## **1.3 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.

## **1.4 Mitigation**

**Introduce CAPTCHA Challenges:** While rate limiting is essential, consider adding an extra layer of security with CAPTCHAs.

**Implement Rate Limiting:** This is the most common and straightforward approach.

# **2.week password policy**

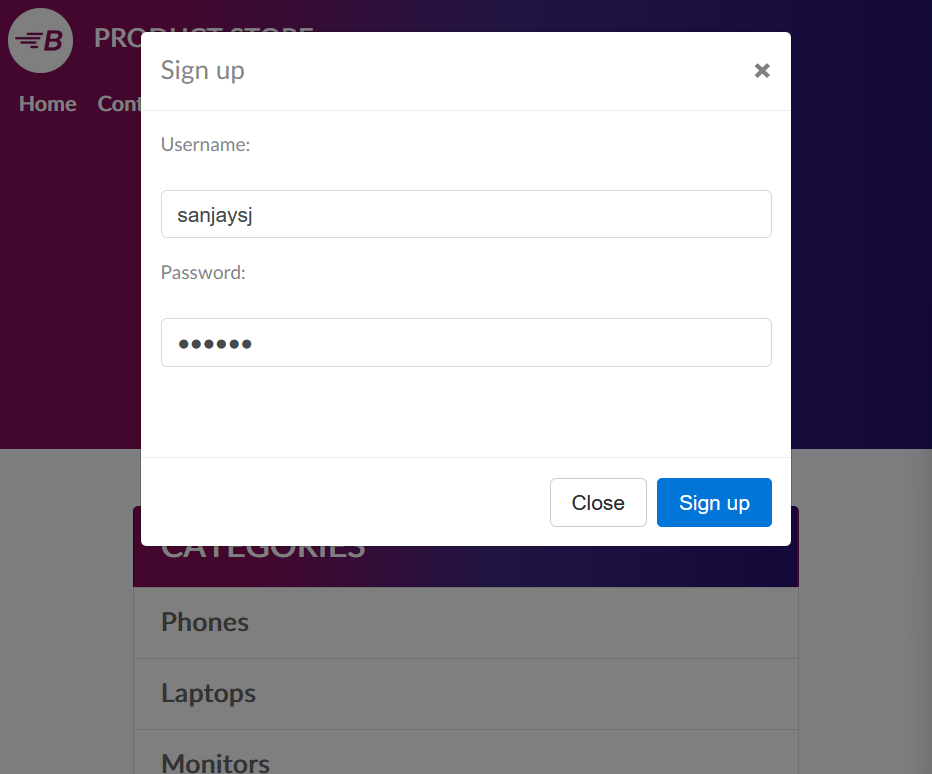
## **2.1 Description**

A critical security concern exists with the web application's signup page. 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. To improve security, the application should implement a password policy that mandates a minimum password length, along with complexity requirements that include a combination of uppercase and lowercase letters, numbers, and symbols.

## **2.2 Vulnerable instance**

<https://www.demoblaze.com/index.html#>

## **2.3 Proof of concept.**

First go to the sign up page then, add username and password. We can see there is no password policy in this input box. Users can enter weak passwords like “123” etc..

## **2.4 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.

# **3.Broken authentication.**

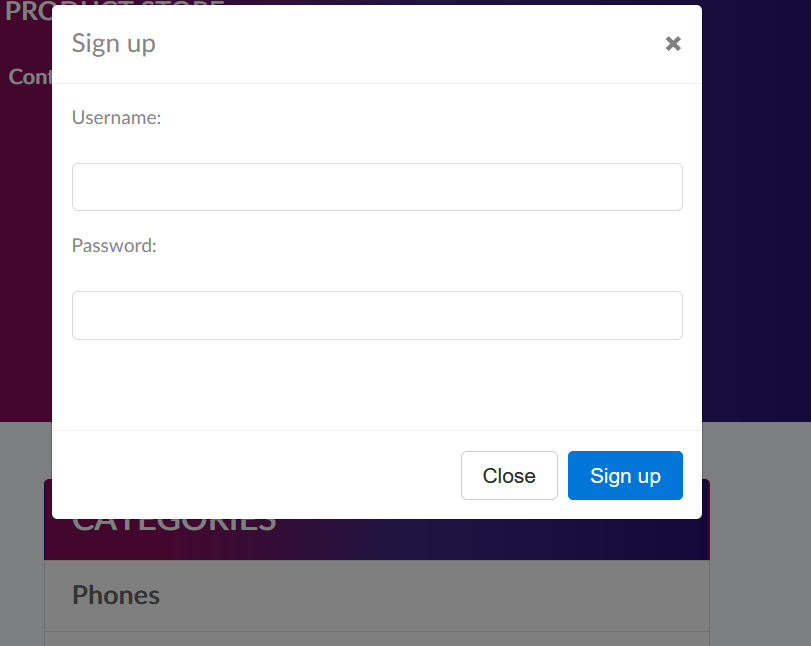
## **3.1 Description**

The web app's sign-up process lacks essential security measures. It only requires a username and password for creating an account. This absence of additional authentication methods weakens the overall security posture.

## **3.2 Vulnerable instance**

<https://www.demoblaze.com/index.html#>

## **3.3 Proof of concept.**

First go to the sign up page , we can see in this page there is only username and password for creating an account.

## **3.4 Mitigation**

**Multi-Factor Authentication (MFA):** Implement multi-factor authentication (MFA) during sign-up. This requires users to provide not only their username and password but also a secondary verification code.

**Email Verification:** Introduce email verification during sign-up. After a user creates an account with username and password, send a verification email with a unique link.

# **4.cryptographic failure**

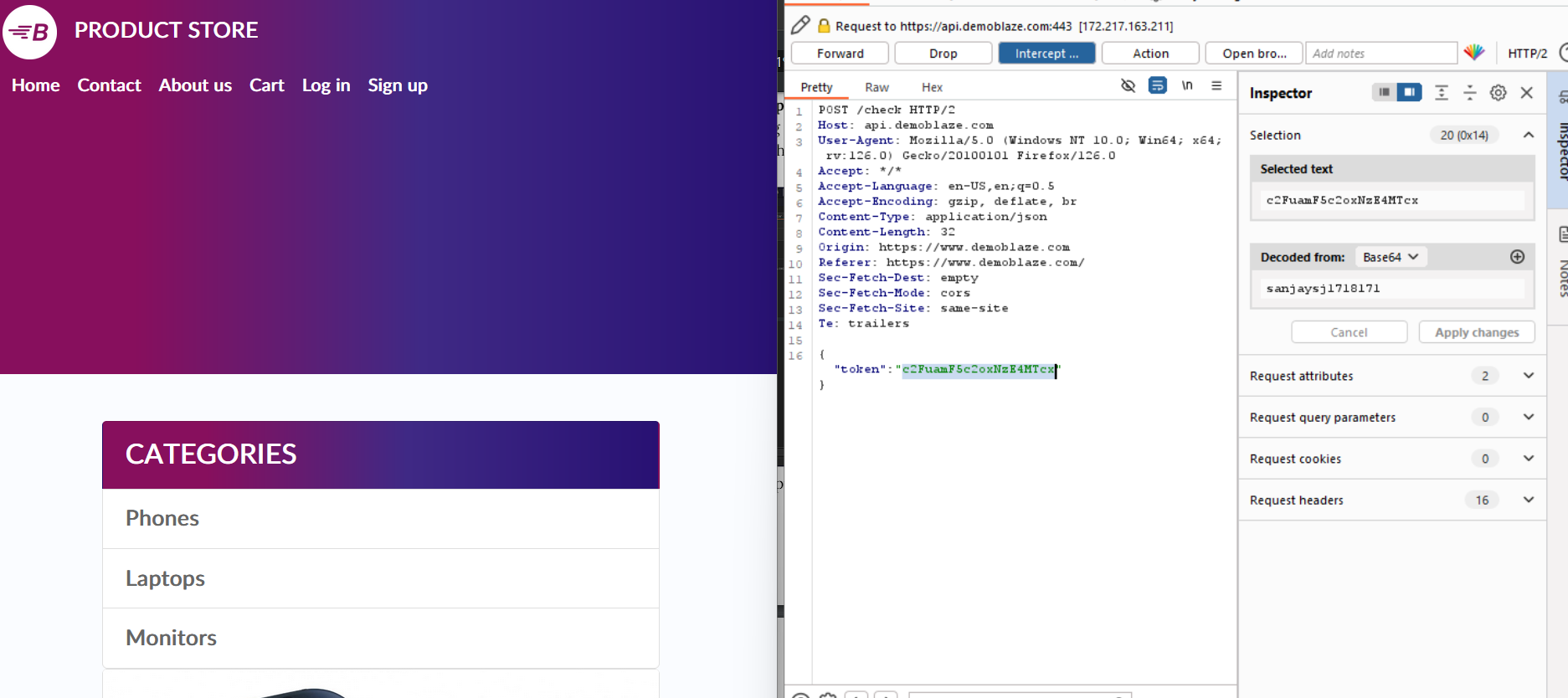
## **4.1 Description**

There's a security issue with how this web app handles logins. The session token created after login is simply encoded (likely in Base64), revealing the username and potentially a session timeframe when decoded. This is risky because anyone who intercepts the token can see the username and potentially hijack the session.

## **4.2 Vulnerable instance**

[https://www.demoblaze.com/index.html#](https://www.demoblaze.com/index.html#b)

## **4.3 Proof of concept.**

When we login , interrupt the request and we can see the token in the base64. We can easily decode the token into plain text.This is risky because anyone who intercepts the token can see the username and potentially hijack the session.

## **4.4 Mitigation**

**Secure Hashing:** Replace the current Base64 encoding (likely a misspelling of Base64) with a secure hashing algorithm like SHA-256. This transforms the data into a unique, unreadable string.

**Enhanced Session Management:** Implement robust session management practices.

# **5.session management**

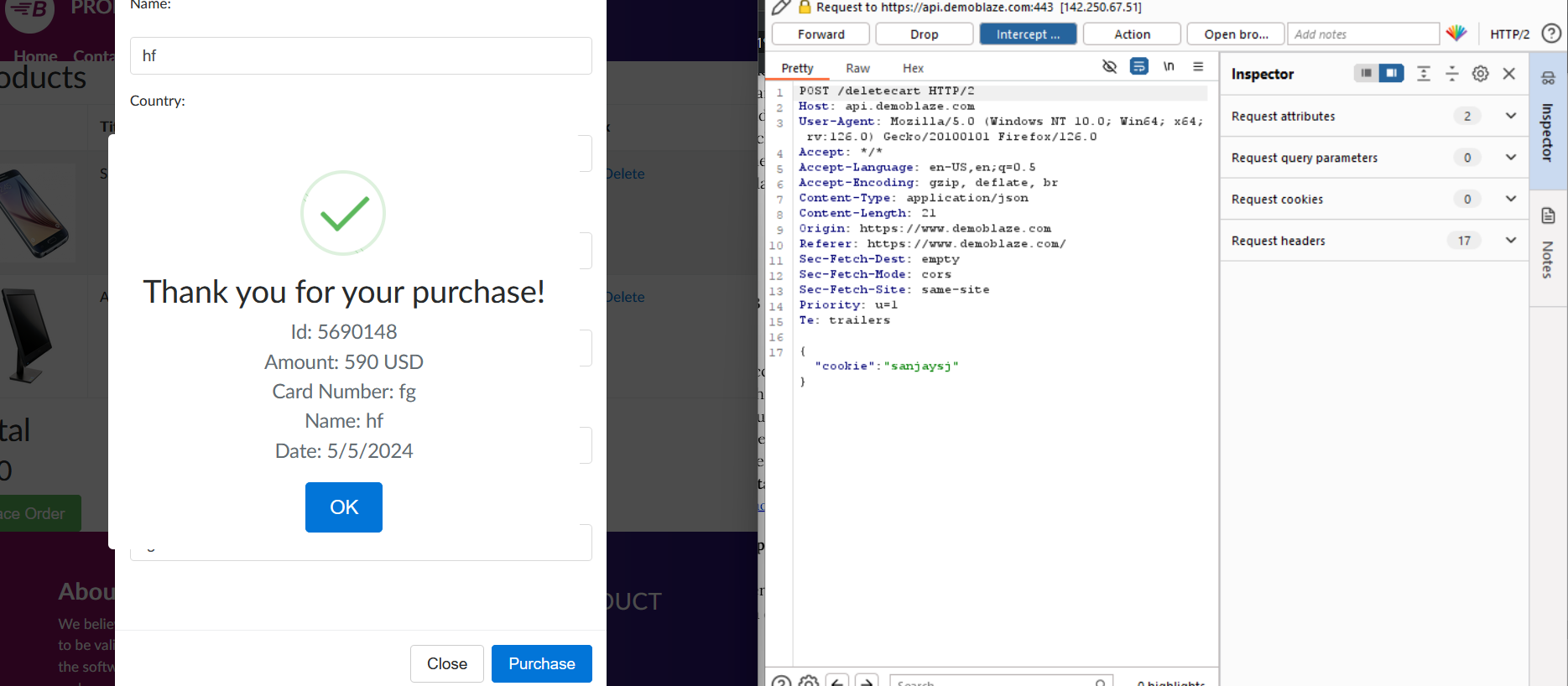
## **5.1 Description**

There's a potential security issue with how this web application handles user information during the purchase process. The concern arises because interrupting the request while buying a product might expose the username stored in a cookie. This vulnerability is critical because an attacker who intercepts the request and accesses the username cookie could potentially gain unauthorized access to the user's account.

## **52 Vulnerable instance**

<https://www.demoblaze.com/cart.html>

## **5.3 Proof of concept.**

Go to the cart page and buy any product and interrupt the request .We can the cookies is the user name.

## **5.4 Mitigation**

**Enforce HTTPS:** Make HTTPS mandatory for all communication between the user's browser and the web application.

**Secure Cookies:** Implement the "Secure" attribute for cookies containing sensitive information like usernames.

**Server-Side Session Management:** Explore using server-side session management instead of relying solely on cookies.

# **6.Security misconfigured**

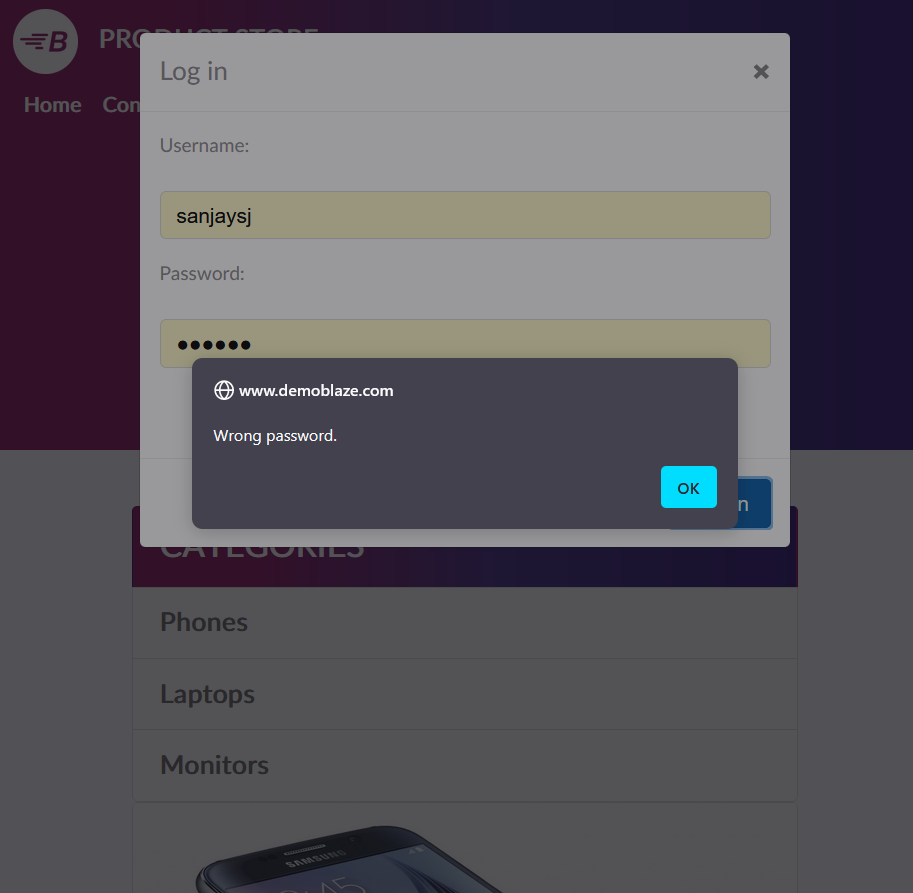
## **6.1 Description**

There's a potential security concern with the way the login process handles errors in this web application. When someone enters an incorrect password, the error message simply says "password is wrong." This lack of specificity can be exploited by attackers.

## **6.2 Vulnerable instance**

<https://www.demoblaze.com/index.html#>

## **6.3 Proof of concept.**

Go to the login page and enter the username and password. If the password is wrong ,they show a message saying the password is wrong.

## **6.4 Mitigation**

**Generic Error Message:** Replace the specific "password is wrong" message with a more generic one like "Invalid login credentials." This avoids revealing whether the username or password is incorrect, making it harder for attackers to confirm valid accounts.

**CAPTCHA Integration:** For high-risk login attempts (e.g., repeated failures from a single IP address), integrate a CAPTCHA challenge.

**Account Lockout:** Consider implementing account lockout after a certain number of consecutive failed login attempts

# **7.Default credentials**

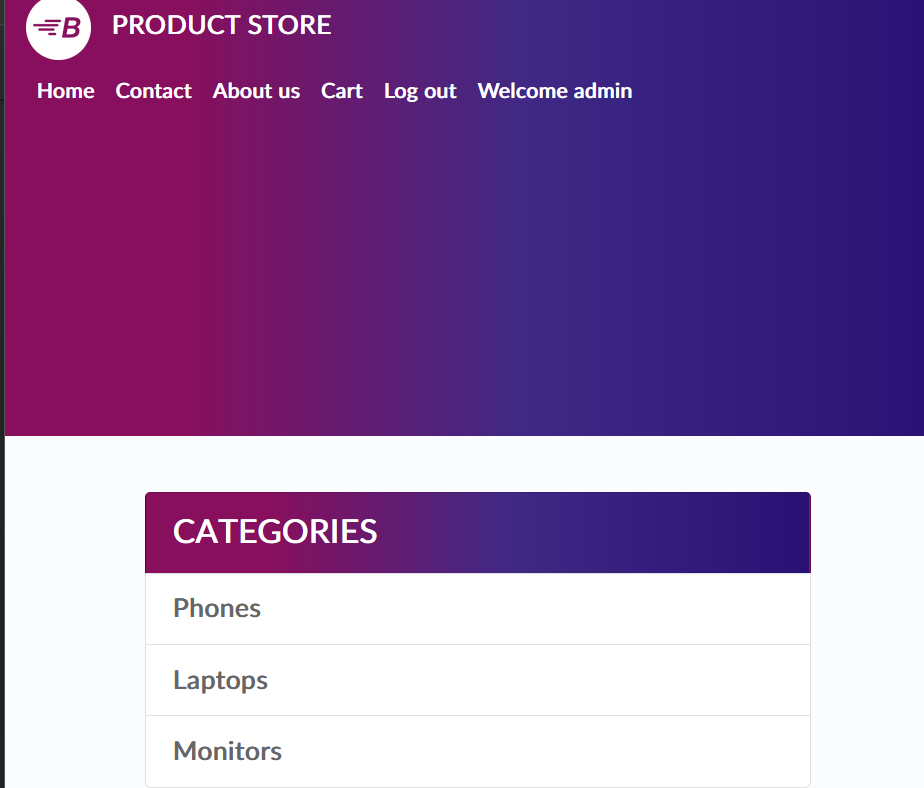
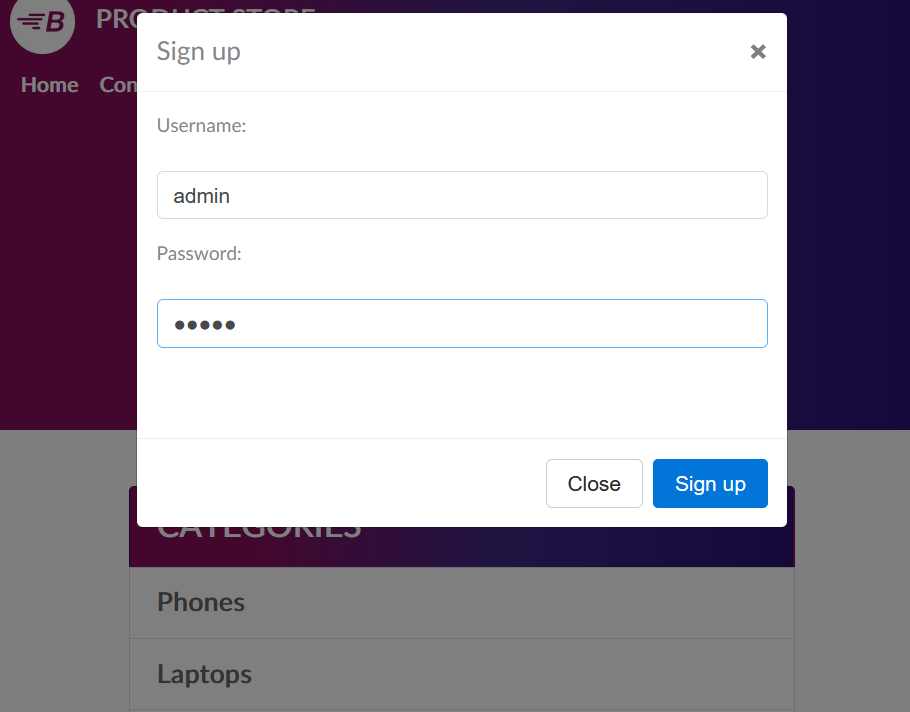
## **7.1 Description**

The login system in this web application has a critical security vulnerability. It appears to allow access using common default credentials, like "admin/admin." This is a serious misconfiguration.

## **7.2 Vulnerable instance**

<https://www.demoblaze.com/index.html#>

## **7.3 Proof of concept.**

In this web app , we can easily access the admin page using the username and password admin /admin.

## **7.4 Mitigation**

**Disable Default Credentials:** Immediately remove all default usernames and passwords from the system.

**Enforce Strong Passwords:** Implement password complexity requirements. Passwords should be at least 12 characters long and include a combination of uppercase and lowercase letters, numbers, and symbols.

# **8.IDOR**

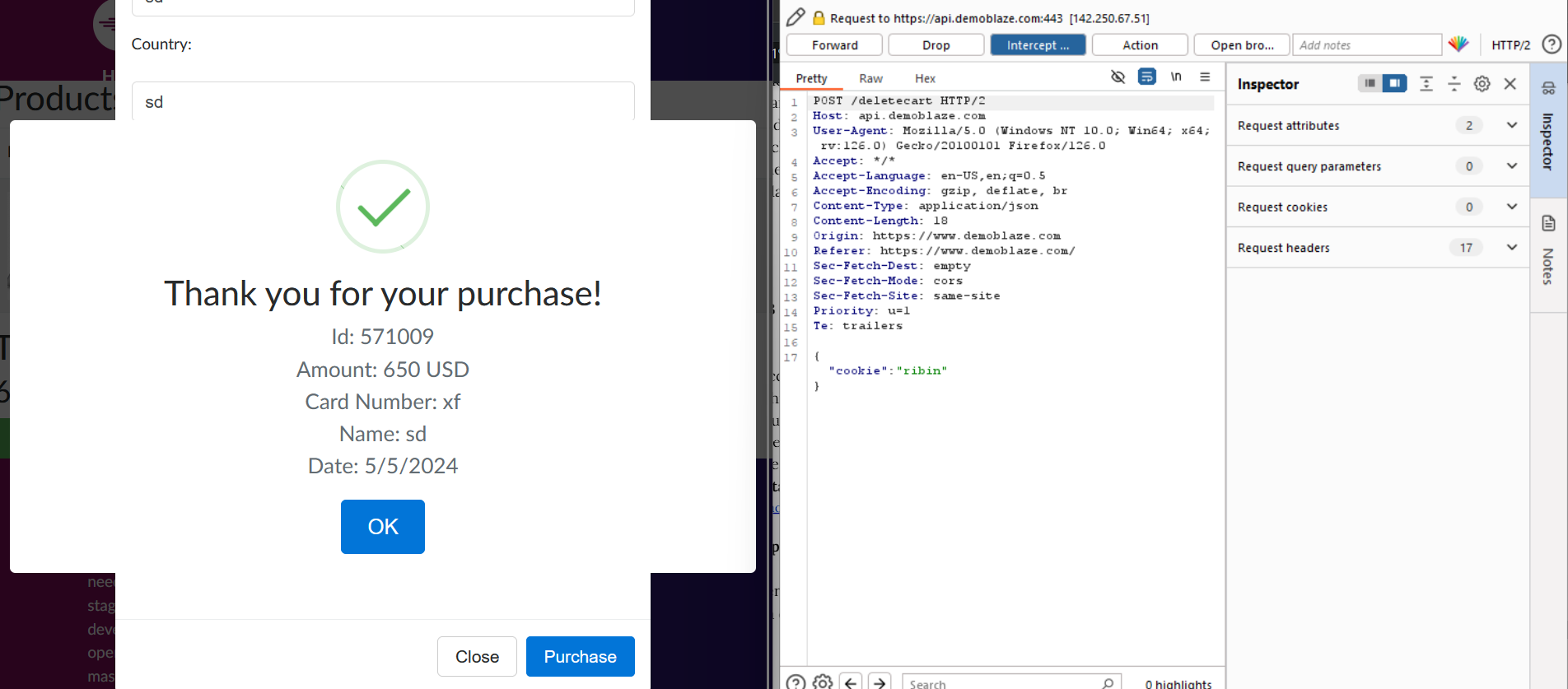
## **8.1 Description**

A critical security vulnerability lurks within this web application's shopping cart functionality. The issue stems from how user information, specifically usernames, is stored during the purchase process. Here's the problem: usernames are simply stored in cookies. This lack of security measures makes them vulnerable. An attacker who intercepts the data transfer (request) during a purchase could steal the username cookie.

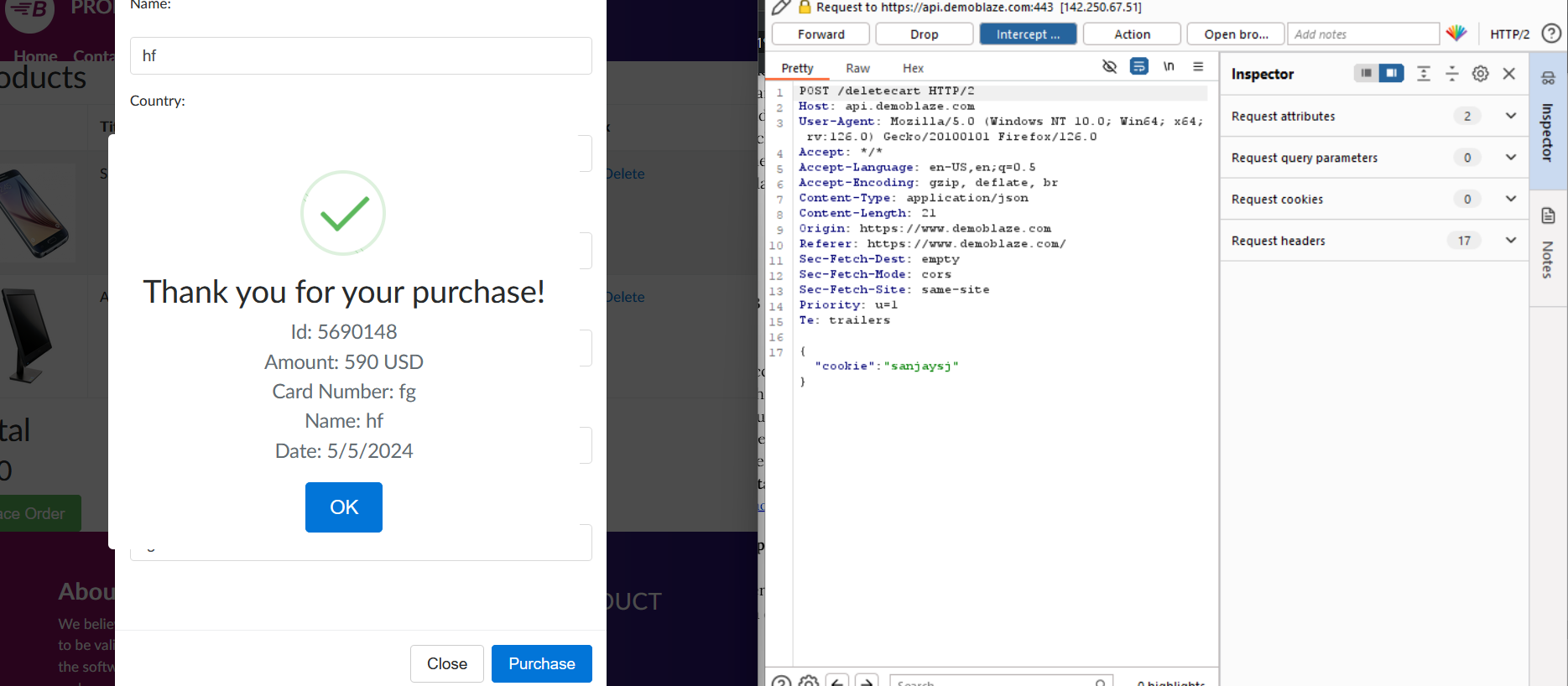
## **8.2 Vulnerable instance**

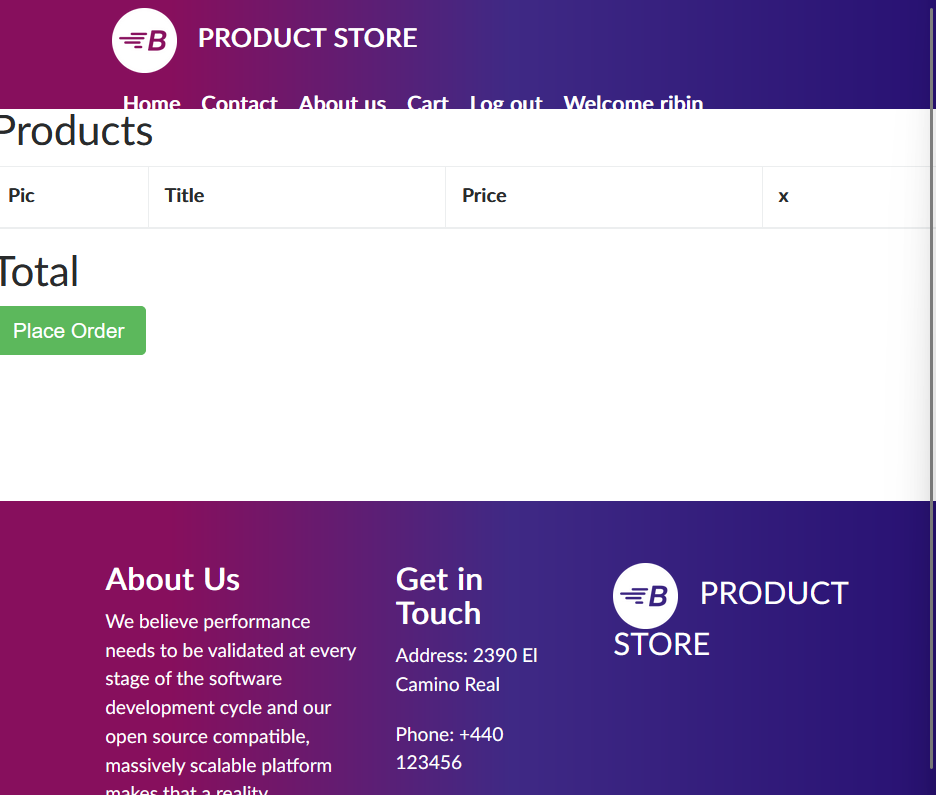
<https://www.demoblaze.com/cart.html>

## **8.3 Proof of concept.**

**Step 1:** first buy any product and interrupt the request .we can see that the cookie is the username .

Step2 : change the user name to another.



**Step3 :** After that go to the cart we see the cart is empty.

**8.4 Mitigation**

**Secure Cookies:** Implement the "Secure" attribute for cookies containing usernames.

**Server-Side Session Management (Advanced):** Explore using server-side session management instead of relying solely on cookies.

# **9.Cross Origin Resource Sharing (CORS)**

## **9.1 Description**

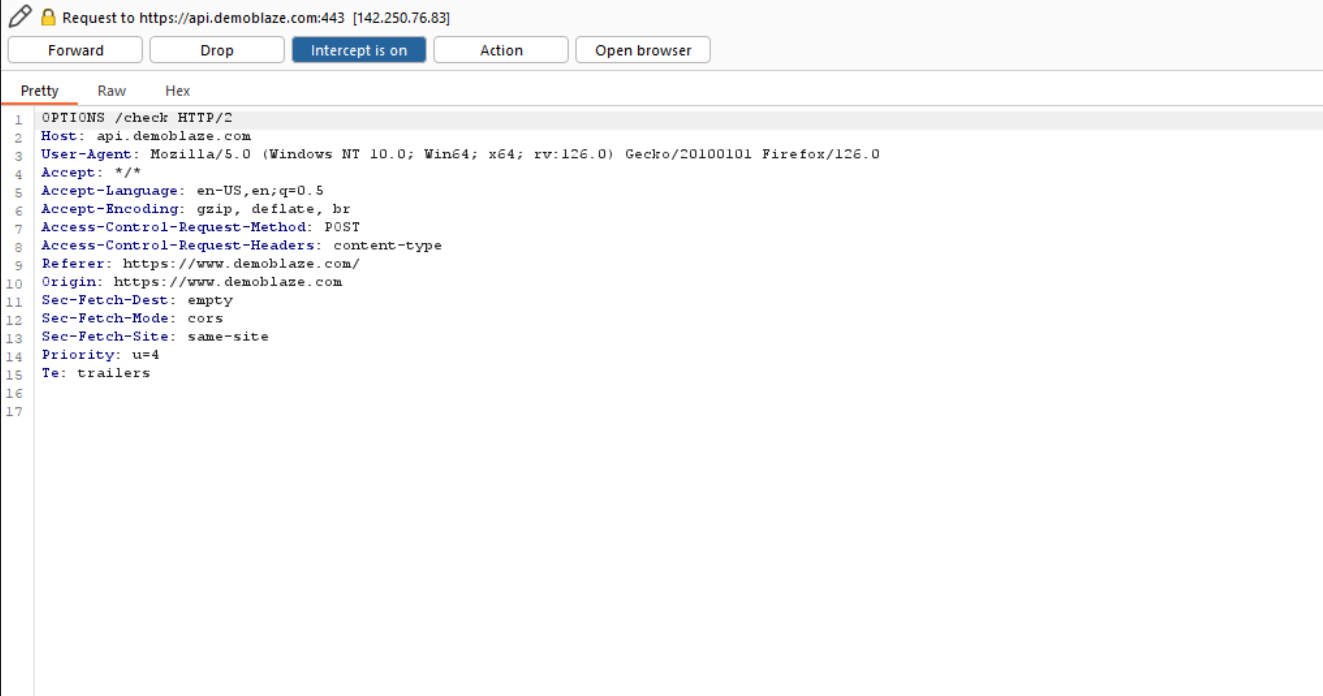
The security of this web application is compromised due to a flaw in how it handles requests from other websites. This issue, known as a Cross-Origin Resource Sharing (CORS) vulnerability, stems from the configuration file. The file mistakenly allows requests from any domain, including potentially malicious ones, to access login functionality.

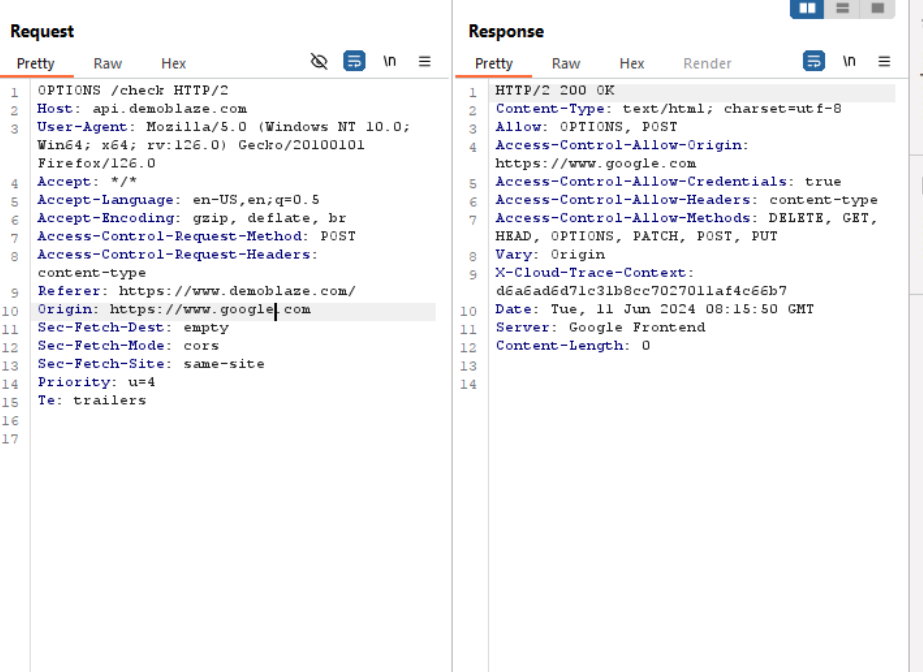
## **9.2 Vulnerable instance**

<https://www.demoblaze.com/>

## **9.3 Proof of concept.**

**Step 1:** go to the instance given above and fetch any data. While doing this, turn on the burp and intercept the request.



**Step 2** : send this request to the Repeater. Then change the origin to google.com. And analyze responses. 

The response says 200 ok .

## **9.4 Mitigation**

**Whitelist Trusted Origins:** This is the most secure approach. Here, the server configuration is updated to specify a list of allowed domains in the Access-Control-Allow-Origin header. Only requests originating from these whitelisted domains will be permitted to access the login functionality. Avoid using wildcards and instead, explicitly list each trusted domain.

**Restrict Allowed Methods:** While whitelisting origins is crucial, limiting the allowed HTTP methods for cross-origin requests provides additional security. Focus on allowing only "safe" methods like GET, HEAD, and possibly POST (for login) for cross-origin requests. Disallow "unsafe" methods like PUT, DELETE, and PATCH, which could be used for malicious purposes.