Altoro MutualVAPT 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 altoro mutual banking 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 altoro mutual banking 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 api can be significantly enhanced, reducing the risk of exploitation by malicious actors.

**Scope :** <http://altoro.testfire.net/>

**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** | **Default credentials** | **8.3** |
| **2** | **XSS in search box** | **8.4** |
| **3** | **Sql injection** | **9.3** |
| **4** | **Lack of rate limiting** | **9.1** |
| **5** | **Weak password policy** | **8.5** |
| **6** | **IDOR** | **8.3** |
| **7** | **Duplicate user login** | **9** |
| **8** | **Broken authorization** | **8.8** |
| **9** | **XSS in feedback** | **8.6** |
| **10** | **Improper validation** | **8.9** |
| **11** | **Information Disclosure.** | **4** |
| **12** | **sensitive data exposure.** | **4** |
| **13** | **Unencrypted url.** | **5.5** |
| **14** | **Missing of ssl/tls** | **9.1** |

# **1.Default credentials**

## **1.1 Description**

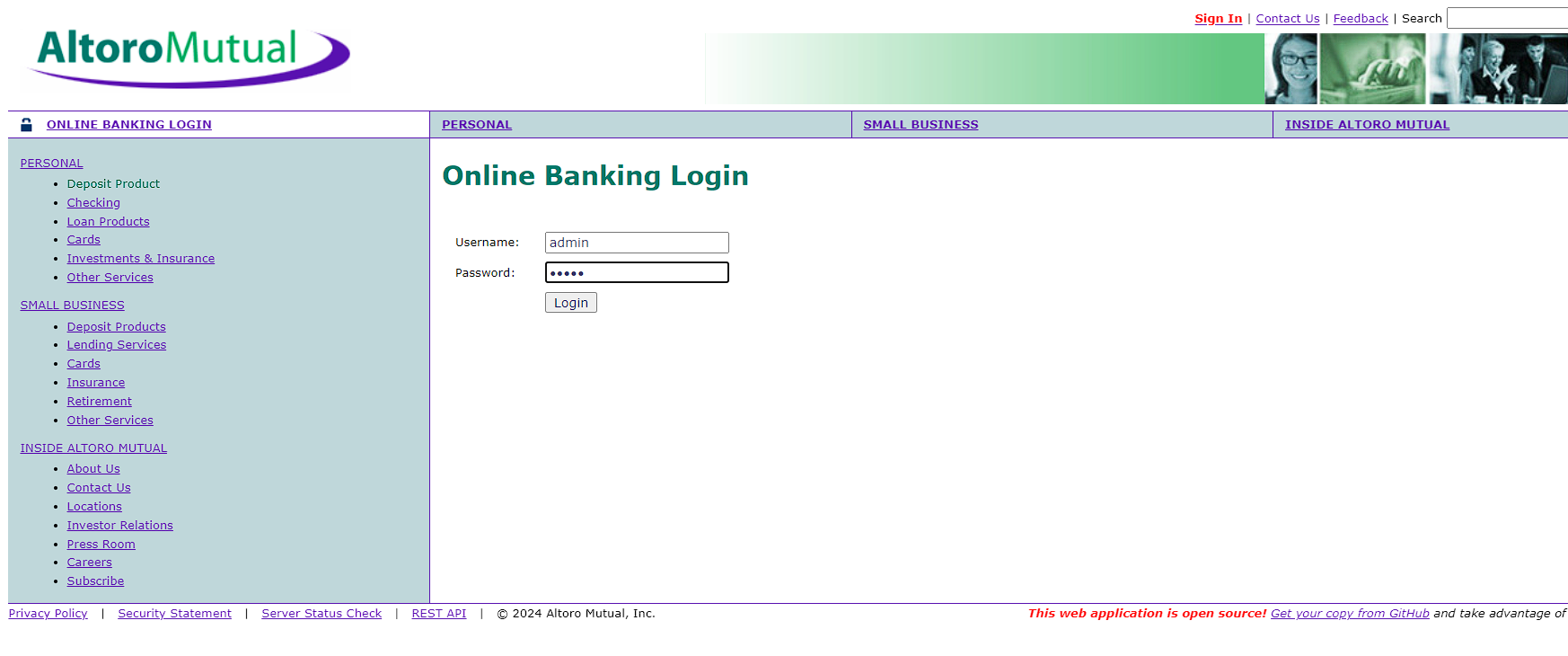
The login system in this web application has a critical security vulnerability. It appears to allow access using common default credentials, like "admin/admin." This is a serious misconfiguration.

## **1.2 Vulnerable instance**

<http://altoro.testfire.net/login.jsp>

## **1.3 Proof of concept.**

Step 1 : go to the login page .Then enter the username and password is “admin”.



We can see that ,we can access the admin page.

## 

## **1.4 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.

# **2.XSS**

## **2.1 Description**

The web application suffers from a critical security issue known as an XSS (Cross-Site Scripting) vulnerability within its search functionality. This means an attacker could potentially inject malicious scripts into the search box. These scripts would then be executed by the web application itself, granting the attacker unauthorized access to sensitive user data or even compromising the entire system.

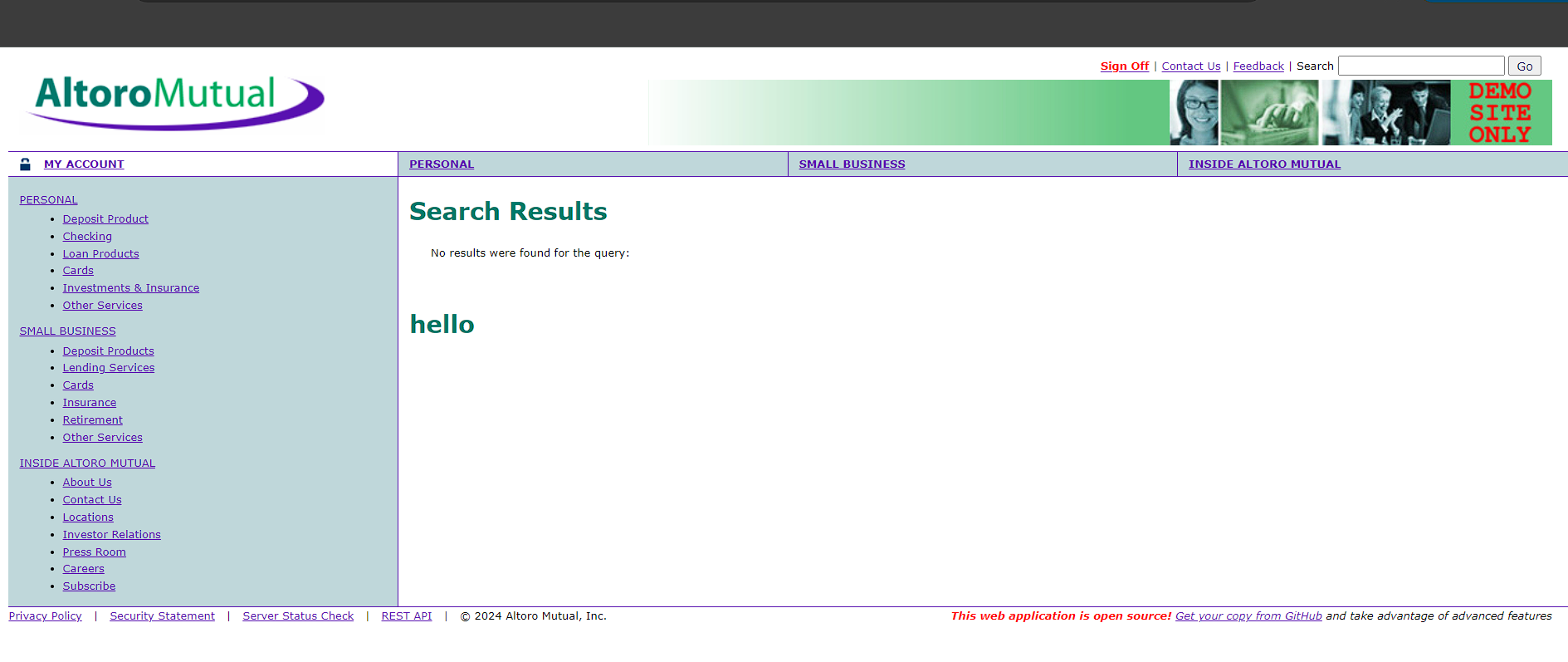
## **2.2 Vulnerable instance**

<http://altoro.testfire.net/bank/main.jsp>

## **2.3 Proof of concept.**

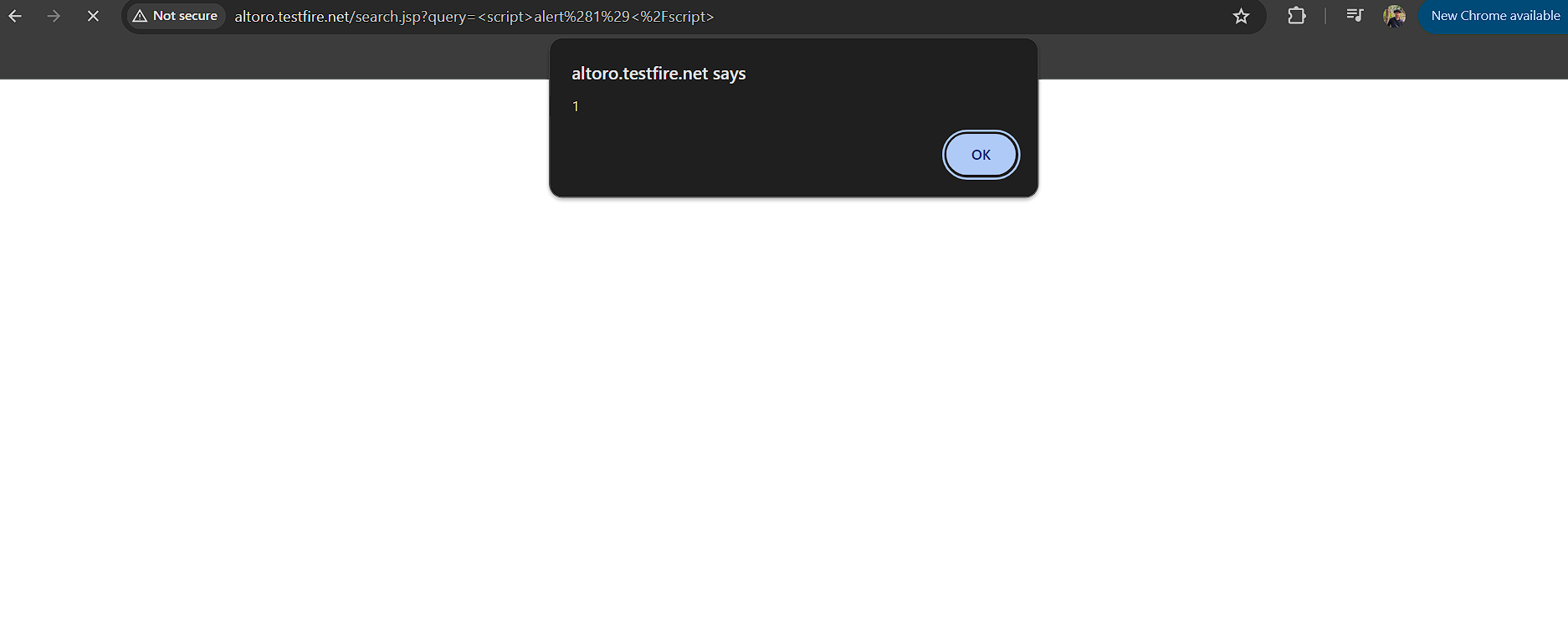
When we add any data that all are displayed so we can add malicious scripts.

**Step 1:** go to the registration page . add this script”<h1>hello</h1>” in the input box.



**Step 2 :** We can see that the script is executed on the web page .

**Step 3 :** go to the register page and add ‘<script>alert(1)</script>” this script to the input box .



## **2.4 Mitigation**

**Validate User Input:** Implement robust validation on the server-side to ensure user input conforms to expected formats .

**Sanitize User Input:** Before displaying or storing user input, sanitize it to remove any harmful code or scripts

# **3.Sql injection.**

## **3.1 Description**

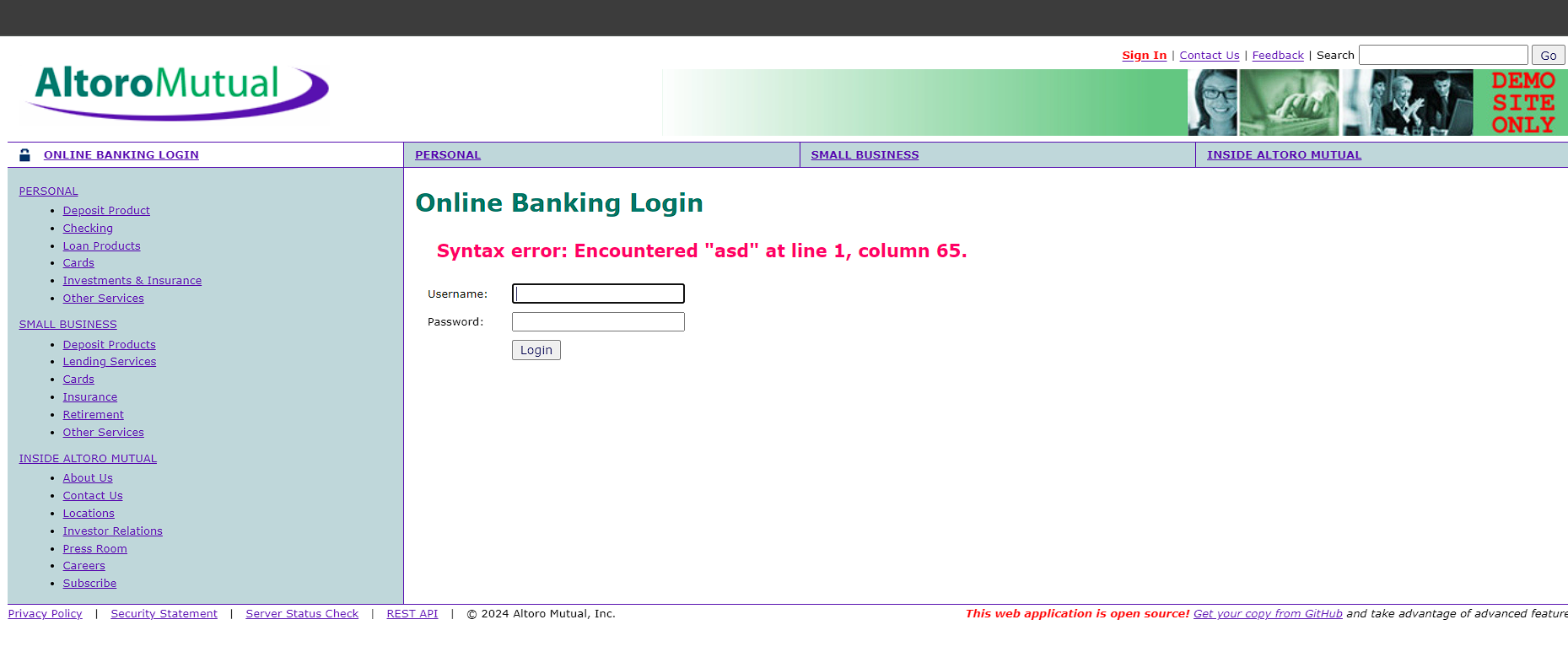
The login page of this web application is susceptible to SQL injection, posing a security risk. Exploiting this vulnerability, an attacker could effortlessly gain access to the web application without requiring a username or password.

## **3.2 Vulnerable instance**

<http://altoro.testfire.net/index.jsp>

## **3.3 Proof of concept.**

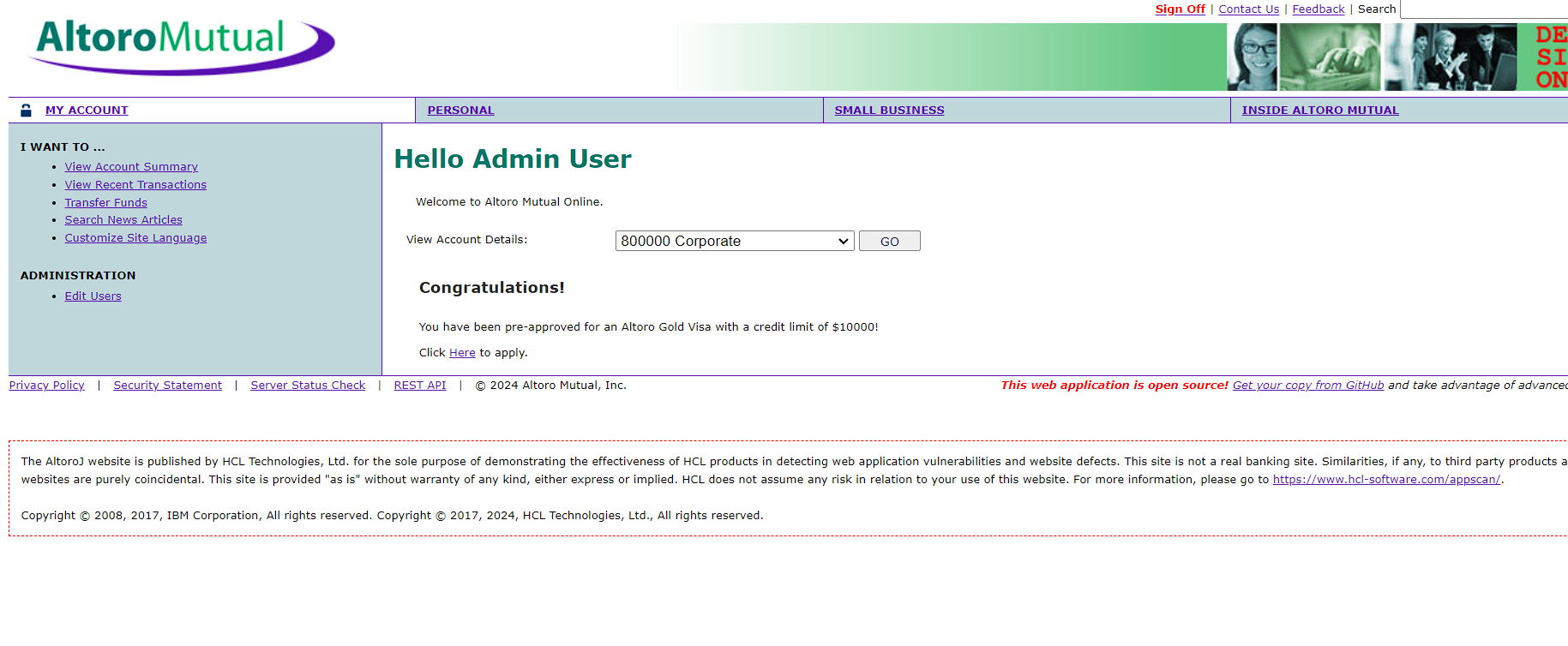
**Step 1 :** go to the login page .Then enter any username and password,then add a ‘.



We can see an error formed . That means that it is vulnerable to sqli.

**Step 2:** Then we can add the payload to the username option “ abc;' or 1=1– “





We can directly access the admin account.

## **3.4 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.

# **4.Lack of rate limiting**

## **4.1 Description**

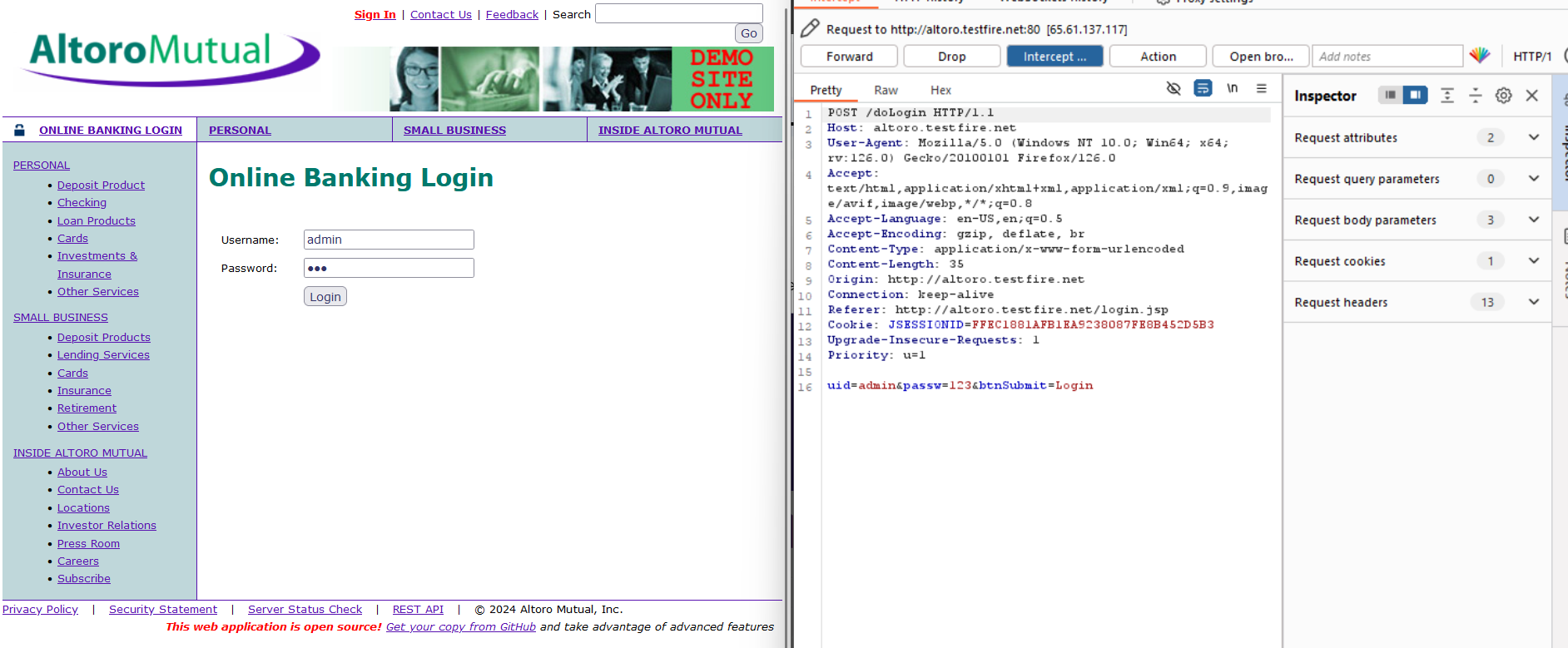
The website's login page appears to lack a security measure called "rate limiting." This feature restricts the number of login attempts a user can make within a specific timeframe. Without rate limiting, an attacker could exploit a vulnerability by repeatedly trying to guess a user's password through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

## **4.2 Vulnerable instance**

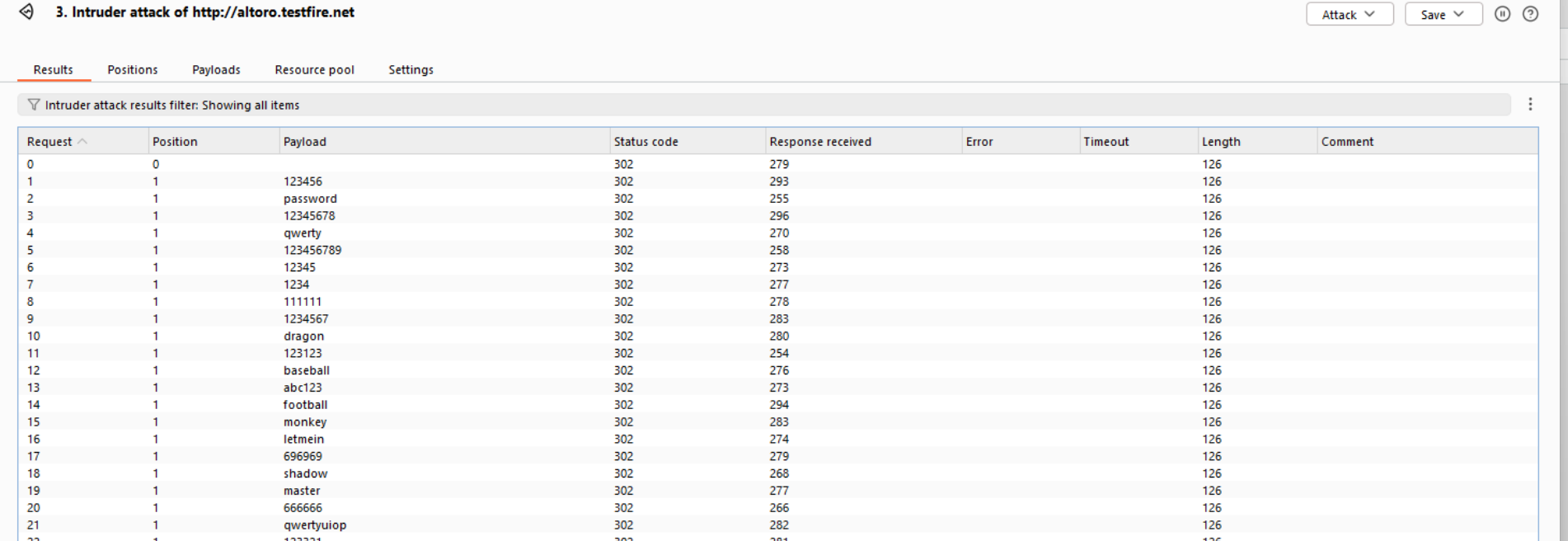
<http://altoro.testfire.net/index.jsp>

## **4.3 Proof of concept.**

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



**Step 2:**send the request to the intruder and add the username. Give a common.txt file as payload and initiate the attack.



**Step 3 :** as you can see that the request is going on and it's been redirected . This can be vulnerable to brute forcing.

## **4.4 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.

# **5. Weak password policy**

## **5.1 Description**

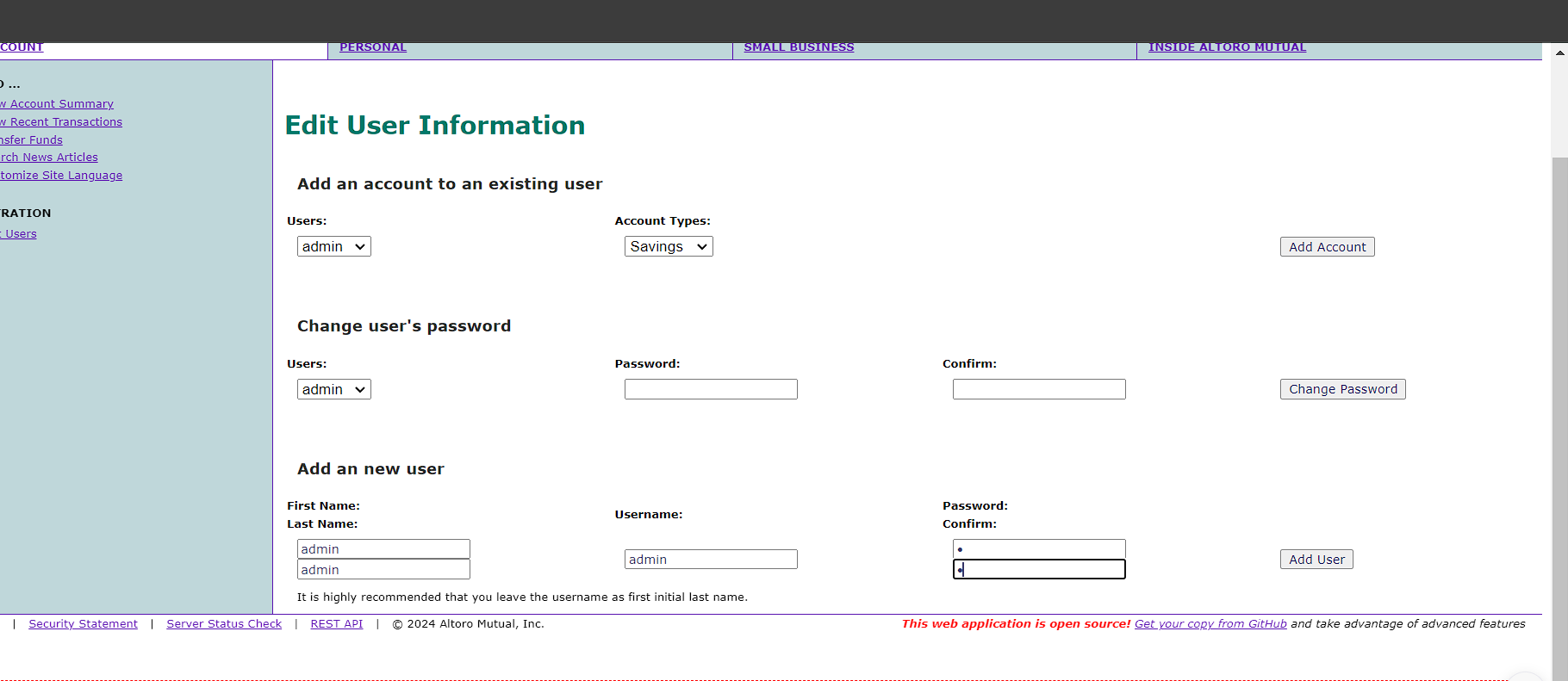
A critical security concern exists with the web application's registration form. The form currently lacks a password policy, meaning there are no enforced requirements for users to create strong passwords. This is problematic because weak passwords are easily guessed or cracked by hackers, putting user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

## **5.2 Vulnerable instance**

<http://altoro.testfire.net/admin/admin.jsp>

## **5.3 Proof of concept.**

In the web app there is an option to add a user and change password. In the 2 options ,there is no password policy.



## **5.4 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.

# **6.IDOR**

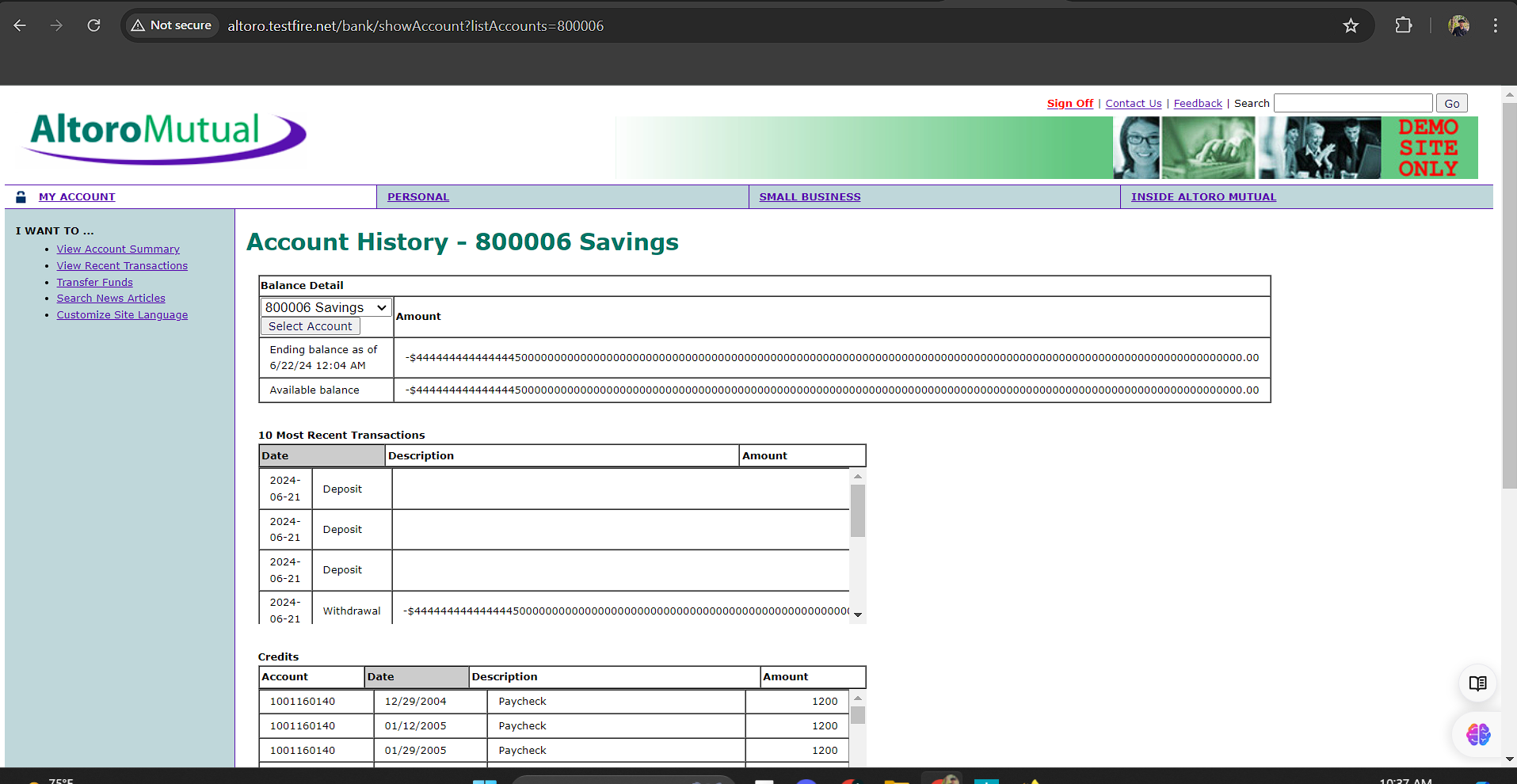
## **6.1 Description**

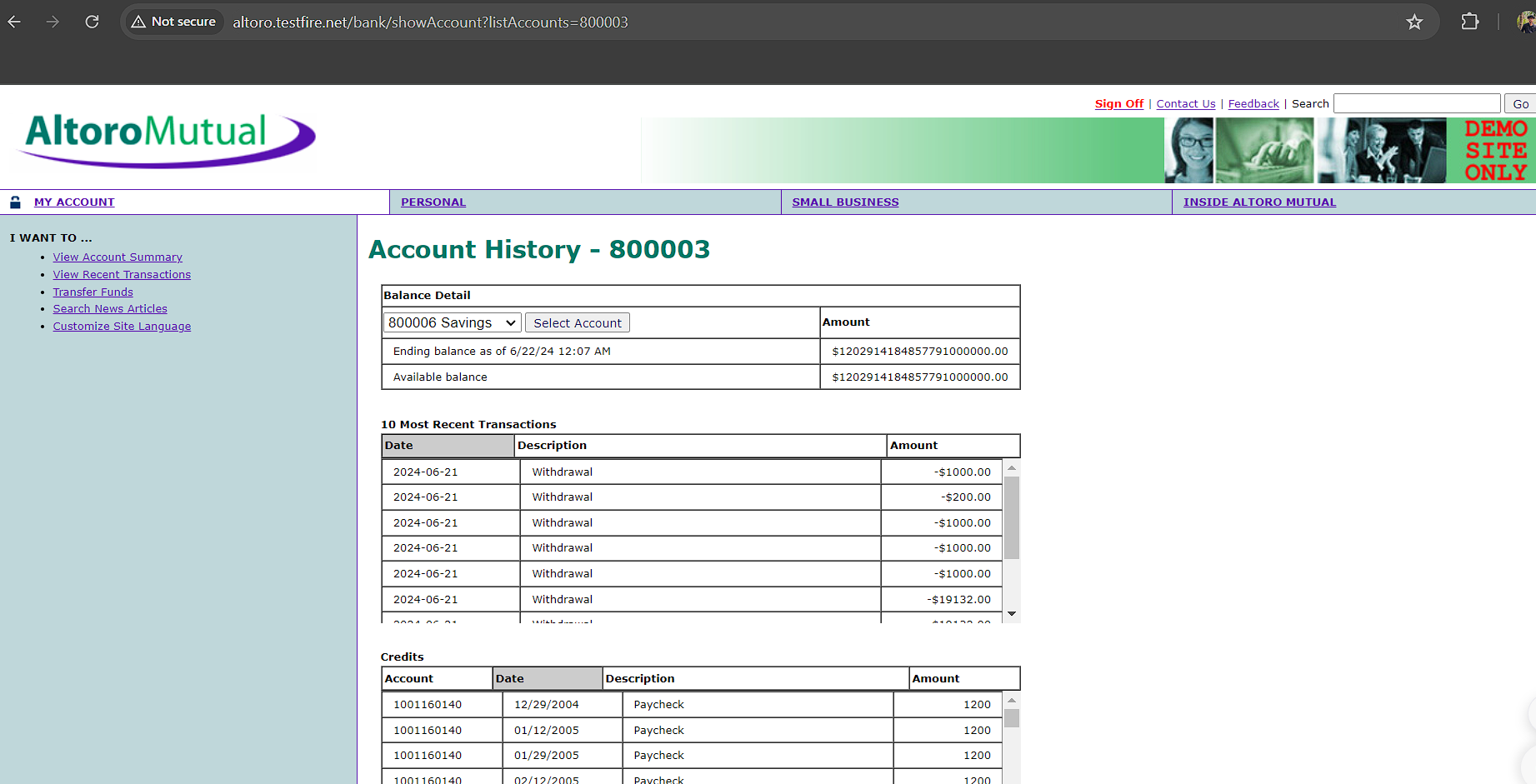
A critical security vulnerability lurks within this web application. The user id is in plain text so anyone can easily change the id. This lack of security measures makes them vulnerable. An attacker who intercepts the data transfer (request) during a purchase could steal the username cookie.

## **6.2 Vulnerable instance**

<http://altoro.testfire.net/bank/showAccount?listAccounts=800006>

## **6.3 Proof of concept.**

**Step 1:**  login to any user then go to the bank details page . we can see the user id is in plain text.

**Step 2 :**  change the user id . then we can see other user details.

## **6.4 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.

# **7.Duplicate user login**

## **7.1 Description**

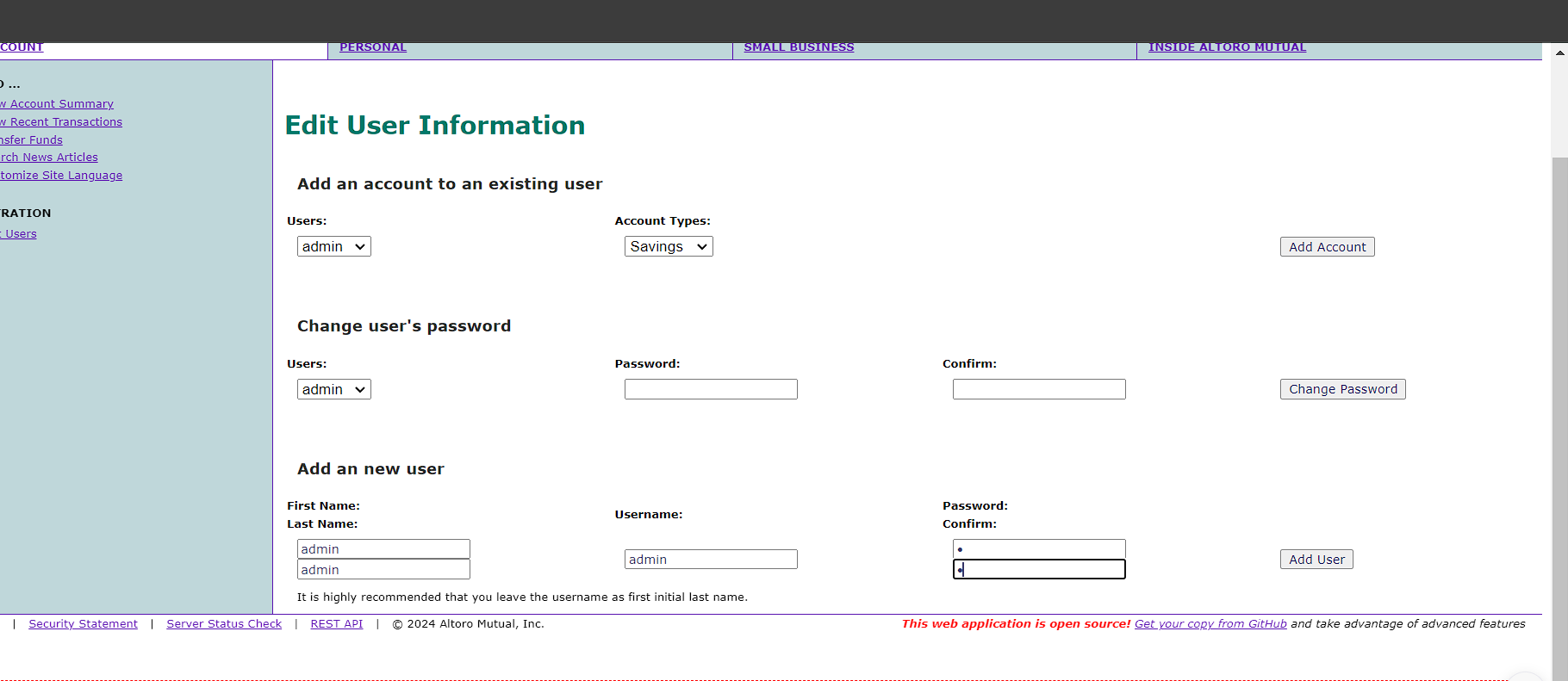
In this web application, there is a vulnerability in the option to add a new user. The system does not check for duplicate usernames, allowing multiple users to be created with the same username.

## **7.2 Vulnerable instance**

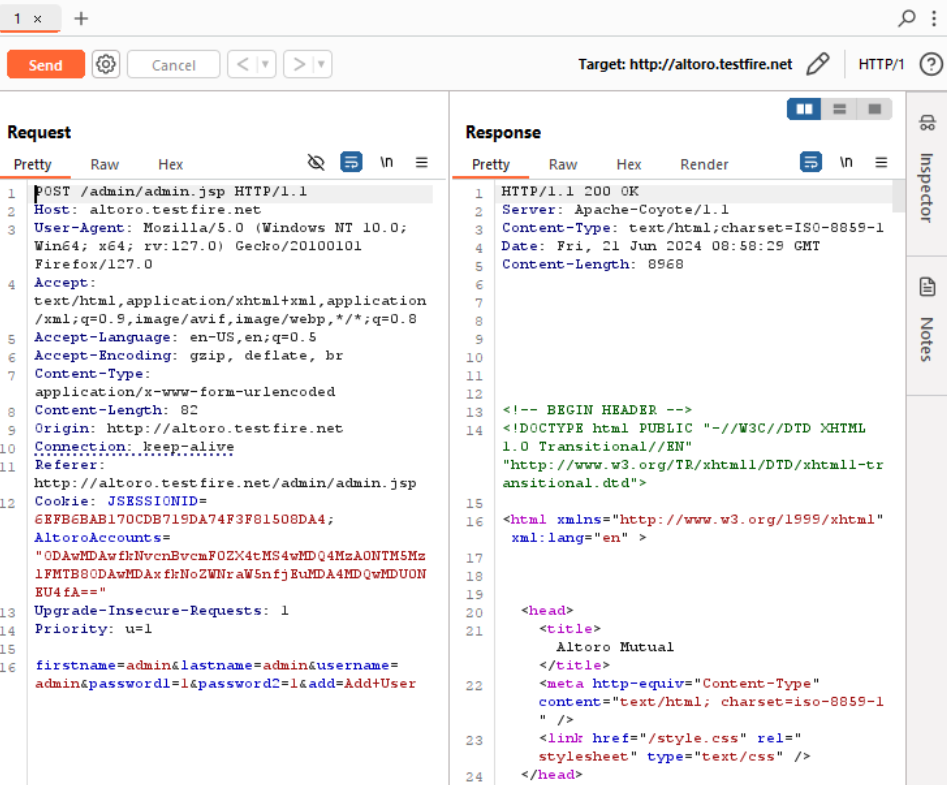
<http://altoro.testfire.net/admin/admin.jsp>

## **7.3 Proof of concept.**

**Step 1:** in admin page there is a option to add new option .add new user details



**Step 2:** we can assume the admin is already a user . There is no validation for duplicate users.



The new duplicate user registered successfully.

## **7.4 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.

# **8. Broken authorization**

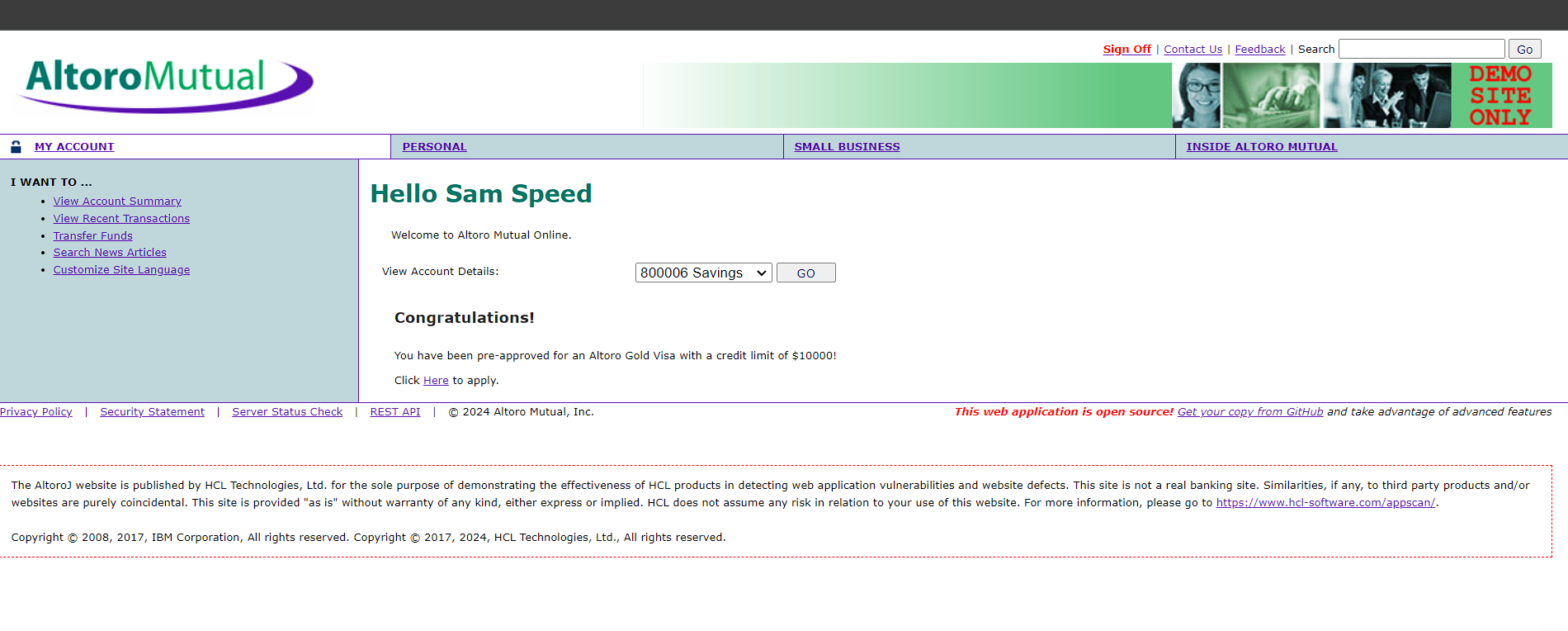
## **8.1 Description**

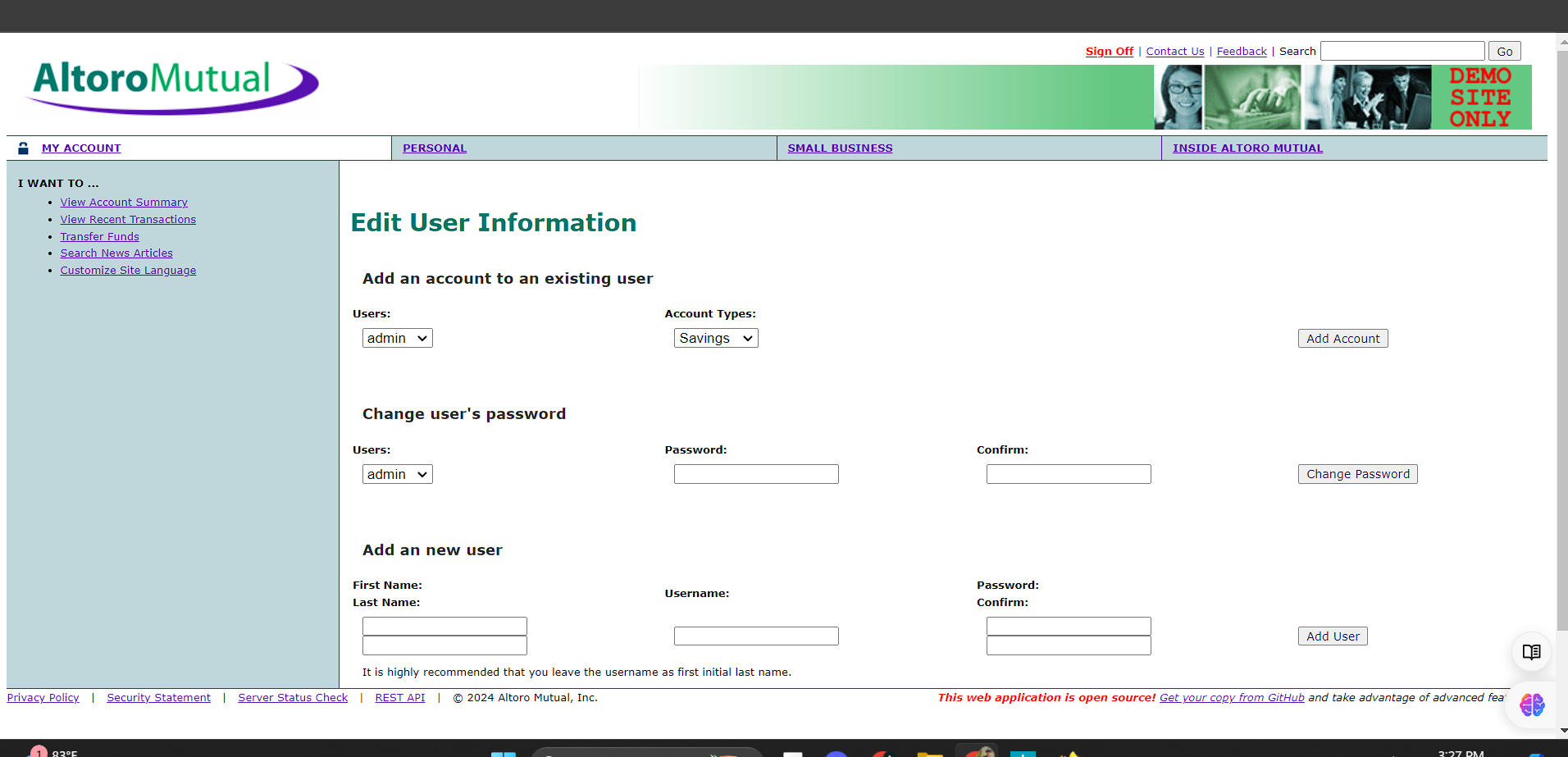
In this web application there is an option to change user details and create new users. On that page there is no authorization . login to any user and change the url to “admin/admin.jsp” so we can directly access the page .

## **8.2 Vulnerable instance**

<http://altoro.testfire.net/bank/main.jsp>

## **8.3 Proof of concept.**

**Step1 :** login to any user ****

**Step 2:** change the url to **“**admin/admin.jsp” we can directly access the administrator page

## **8.4 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.

# **9.XSS**

## **9.1 Description**

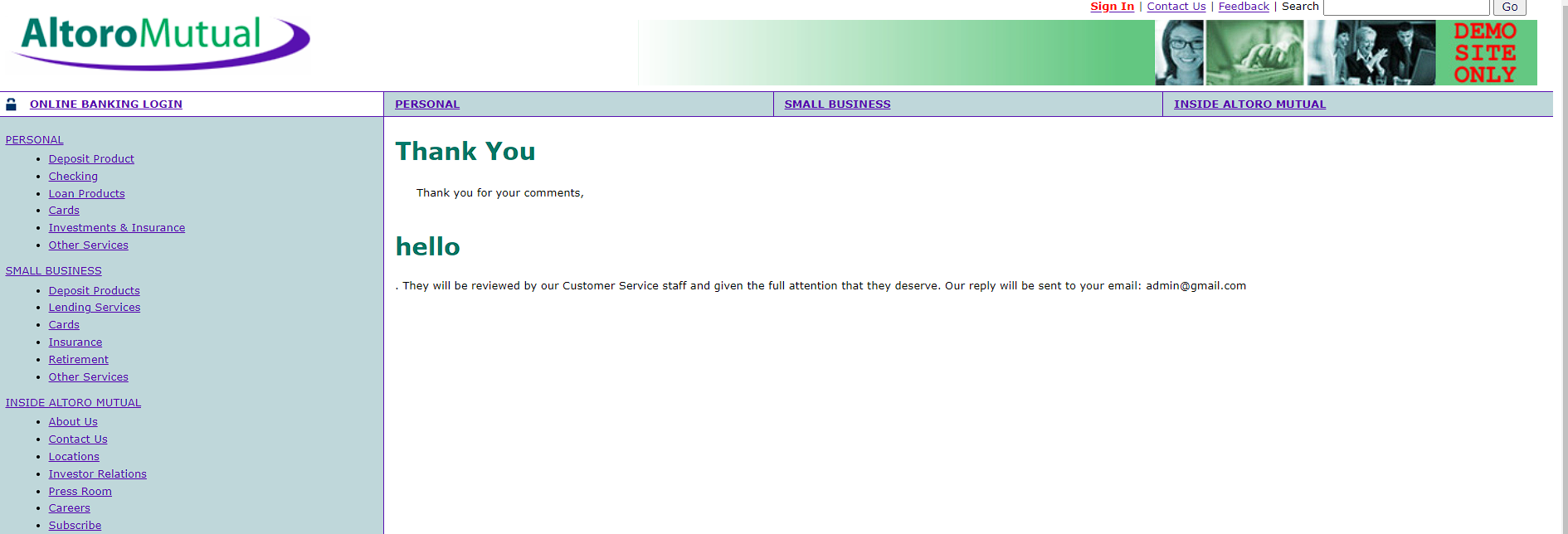
The web application suffers from a critical security issue known as an XSS (Cross-Site Scripting) vulnerability in the feedback page. This means an attacker could potentially inject malicious scripts into the input box. These scripts would then be executed by the web application itself, granting the attacker unauthorized access to sensitive user data or even compromising the entire system.

## **9.2 Vulnerable instance**

<http://altoro.testfire.net/feedback.jsp>

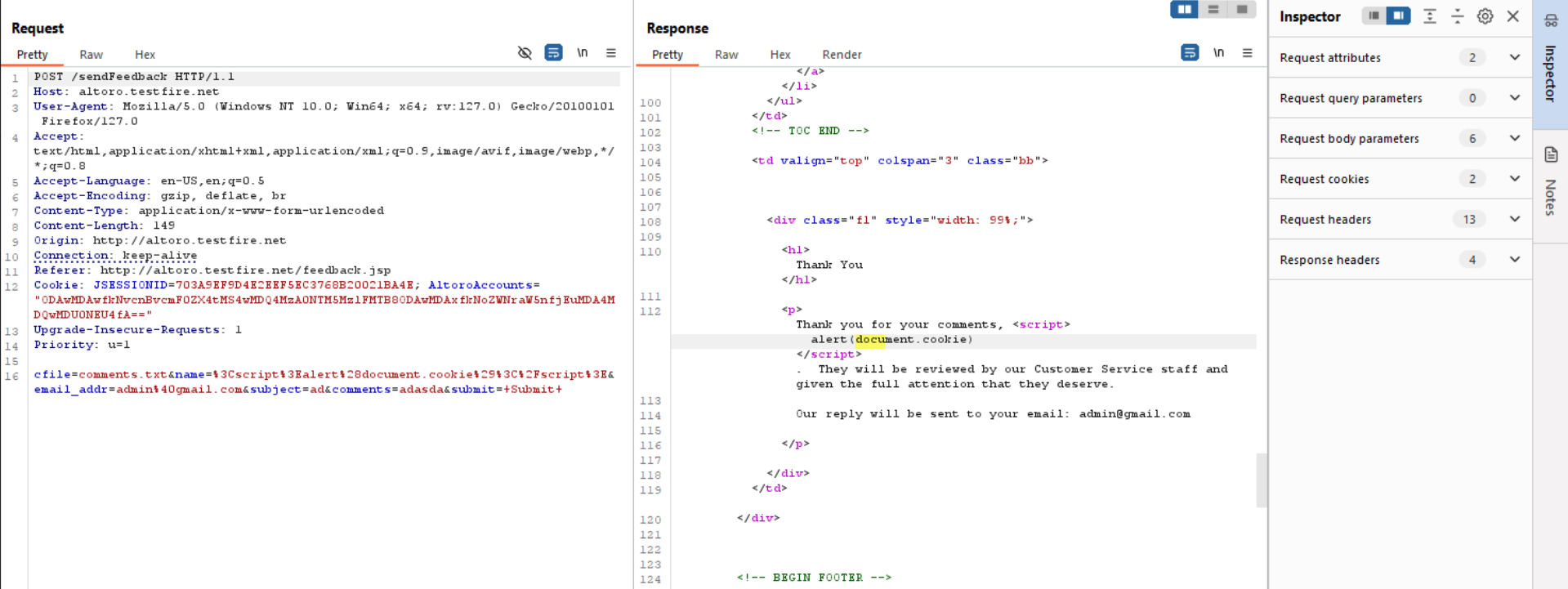
## **9.3 Proof of concept.**

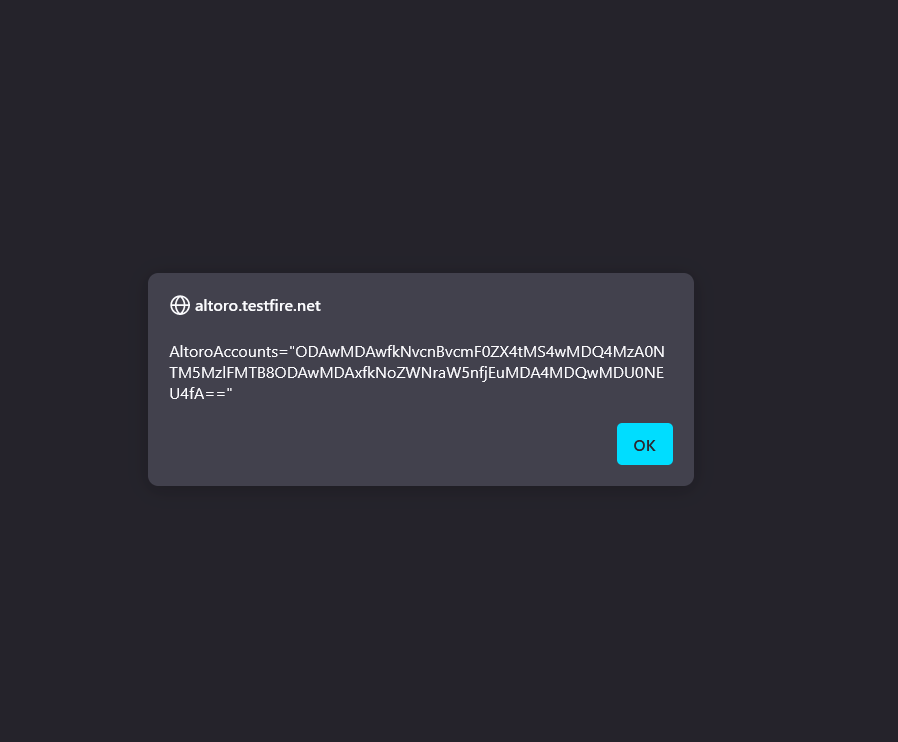
**Step 1 :** go to the feedback page. There are some input boxes , in the name option add a simple script “<h1>hello</h1> .



**Step 2 :** We can see that the script is executed on the web page .

**Step 3 :** go to the register page and add “<script>alert(document.cookie)</script>” to the input box .





## **9.4 Mitigation**

**Validate User Input:** Implement robust validation on the server-side to ensure user input conforms to expected formats .

**Sanitize User Input:** Before displaying or storing user input, sanitize it to remove any harmful code or scripts

# **10.Improper validation**

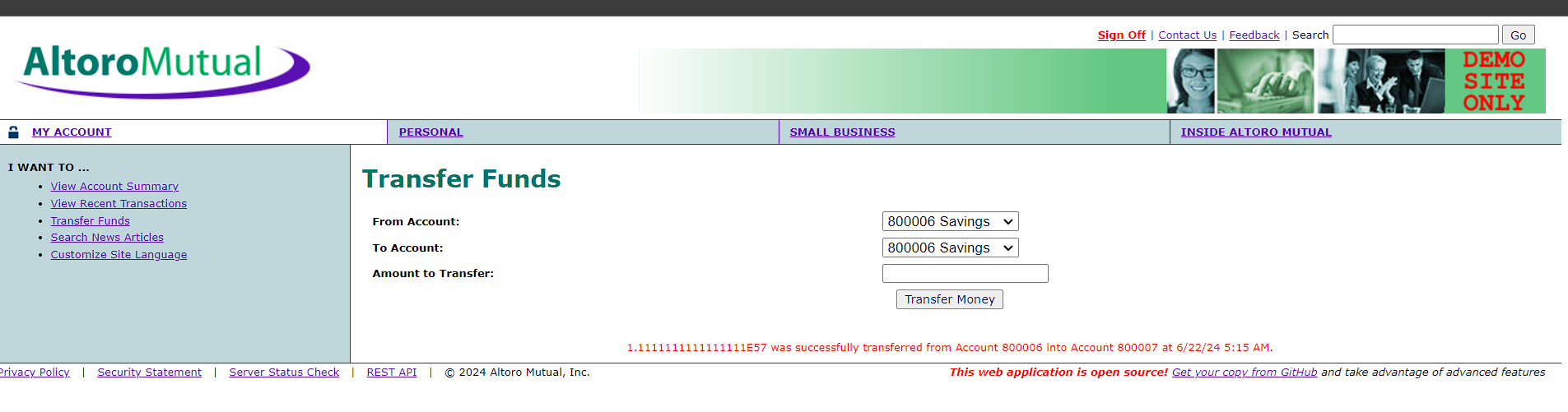
## **10.1 Description**

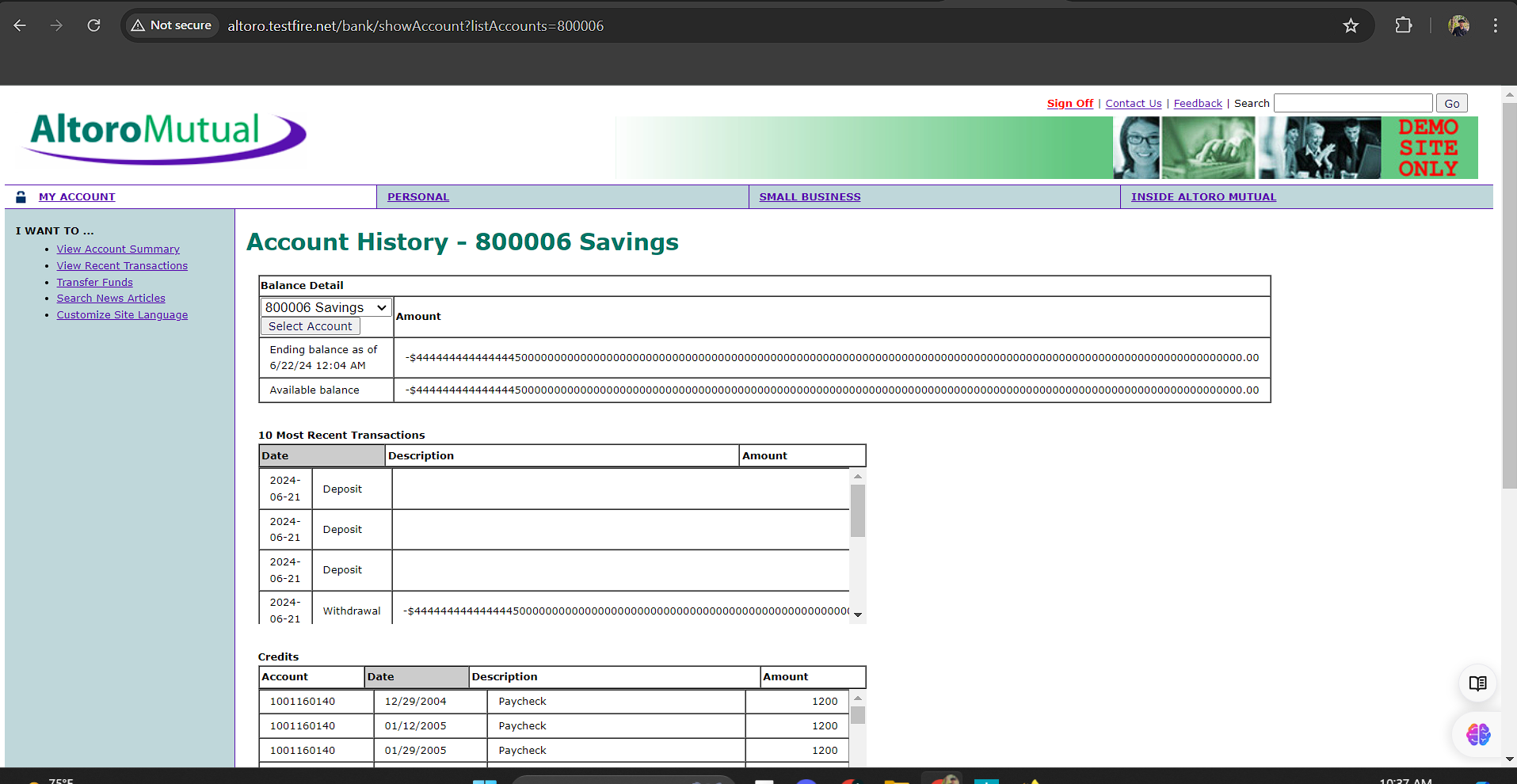
A critical security flaw exists within the money transfer function of this banking application. The functionality lacks validation for the transfer amount. This means users can enter any value into the transfer field, regardless of how large or negative the number might be. This absence of validation creates a significant vulnerability.

## **10.2 Vulnerable instance**

<http://altoro.testfire.net/bank/transfer.jsp>

## **10.3 Proof of concept.**

**Step 1:**  go to the fund transfer page, there is an option to add an amount . There is no limit add amount we can unlimited number of amounts.



## **10.4 Mitigation**

**Set Limits:** Define a minimum and maximum allowed amount for transfers. This restricts users from entering unreasonably high or negative numbers.

**Data Type Validation:** Enforce the data type for the amount field. For instance, only allow numeric characters and a decimal point (if applicable) to prevent unexpected inputs.

# **11.Information Disclosure.**

## **11.1 Description**

This application exposes sensitive server details in every response, creating a vulnerability that attackers could exploit.

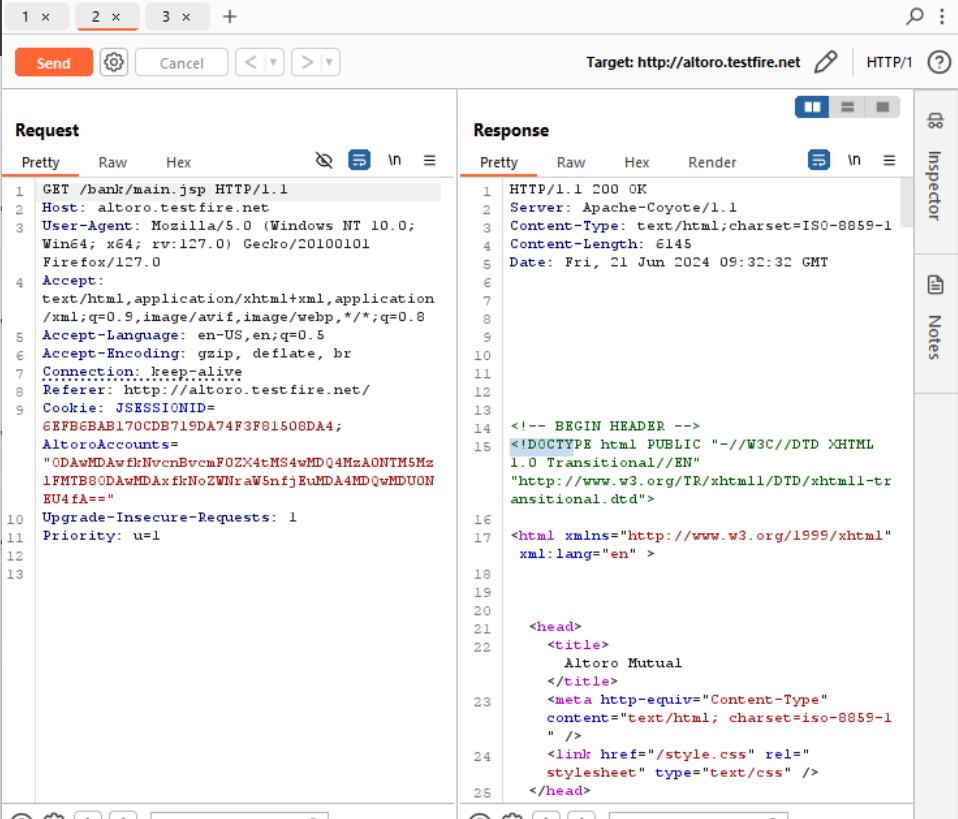
## **11.2 Vulnerable instance**

<http://altoro.testfire.net/bank/doTransfer>

## **11.3 Proof of concept.**

Capture the request and send it to the repeater.

Then we can see the server details in response.



## **11.4 Mitigation**

**Filter Response Data:** Implement server-side logic to filter out any sensitive information from the response before sending it back to the application.

**Minimize Information Disclosure:** Review the application logic and identify why server details are being included in responses in the first place.

# **12.sensitive data exposure.**

## **12.1 Description**

In this web application there is a vulnerability .

## **12.2 Vulnerable instance**

<http://altoro.testfire.net/bank/showAccount?listAccounts=80>

## **12.3 Proof of concept.**

Go to the translation details page ,then change the id to “80”.



## **12.4 Mitigation**

**Customize the 404 Error Page:** Most web servers allow customization of the 404 error page. This allows you to design a user-friendly page that doesn't reveal any server details.

**Web Application Framework Configuration:** If the web application is built using a framework, there might be framework-specific settings to control what information is displayed in error messages.

# **13.Unencrypted url.**

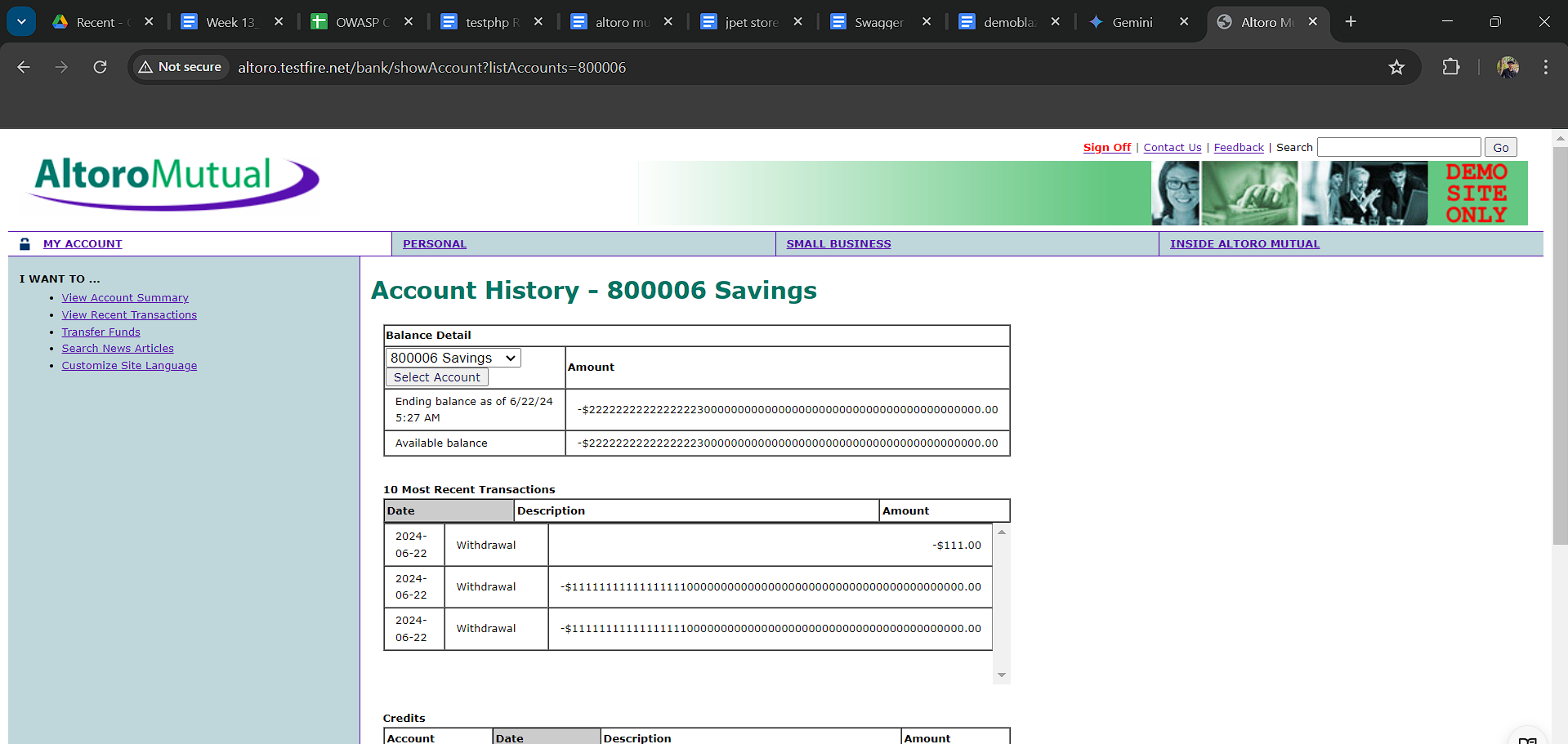
## **13.1 Description**

This web application's money transfer functionality transmits data in unencrypted URLs, exposing sensitive transfer details during the process.

## **13.2 Vulnerable instance**

<http://altoro.testfire.net/bank/showAccount?listAccounts=800006>

## **13.3 Proof of concept.**

Go to the summary page we can the id is in plain text .attacker easily change the id to another

## **13.4 Mitigation**

**Implement HTTPS:** This is the primary solution. Switch all communication between the web application and the server to HTTPS

**URL Redirection with Tokens:** Instead of directly embedding sensitive information in the URL, consider using a secure redirection approach

# **14.Missing of ssl/tls**

## **14.1 Description**

In the banking application using http. That data transfering is not in encrypted form. Transfer in plaintext.

## **14.2 Vulnerable instance**

<http://altoro.testfire.net/index.jsp>

## **14.3 Proof of concept.**

We can see the web application not using ssl/tls. Data passing in plain text.



## **14.4 Mitigation**

**Redirect Traffic to HTTPS:** Once HTTPS is enabled, the server can be configured to automatically redirect all HTTP traffic to HTTPS. This ensures all communication happens through the secure channel.

**Redirect Traffic to HTTPS:** Once HTTPS is enabled, the server can be configured to automatically redirect all HTTP traffic to HTTPS. This ensures all communication happens through the secure channel.