ZERO BANKVAPT 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 zero bank 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 zero bank 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://zero.webappsecurity.com/>

**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** | **Missing of ssl/tls** | **9.3** |
| **2** | **Unencrypted url** | **5.7** |
| **3** | **IDOR** | **8.3** |
| **4** | **Privilege access** | **9.2** |
| **5** | **Improper payment validation** | **8.5** |
| **6** | **improper validation** | **8.7** |
| **7** | **Validation error** | **8.5** |
| **8** | **Server Status page access** | **7.6** |
| **9** | **XSS** | **8.9** |
| **10** | **Information Disclosure** | **7.6** |
| **11** | **outdated version** | **8.9** |

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

## **1.1 Description**

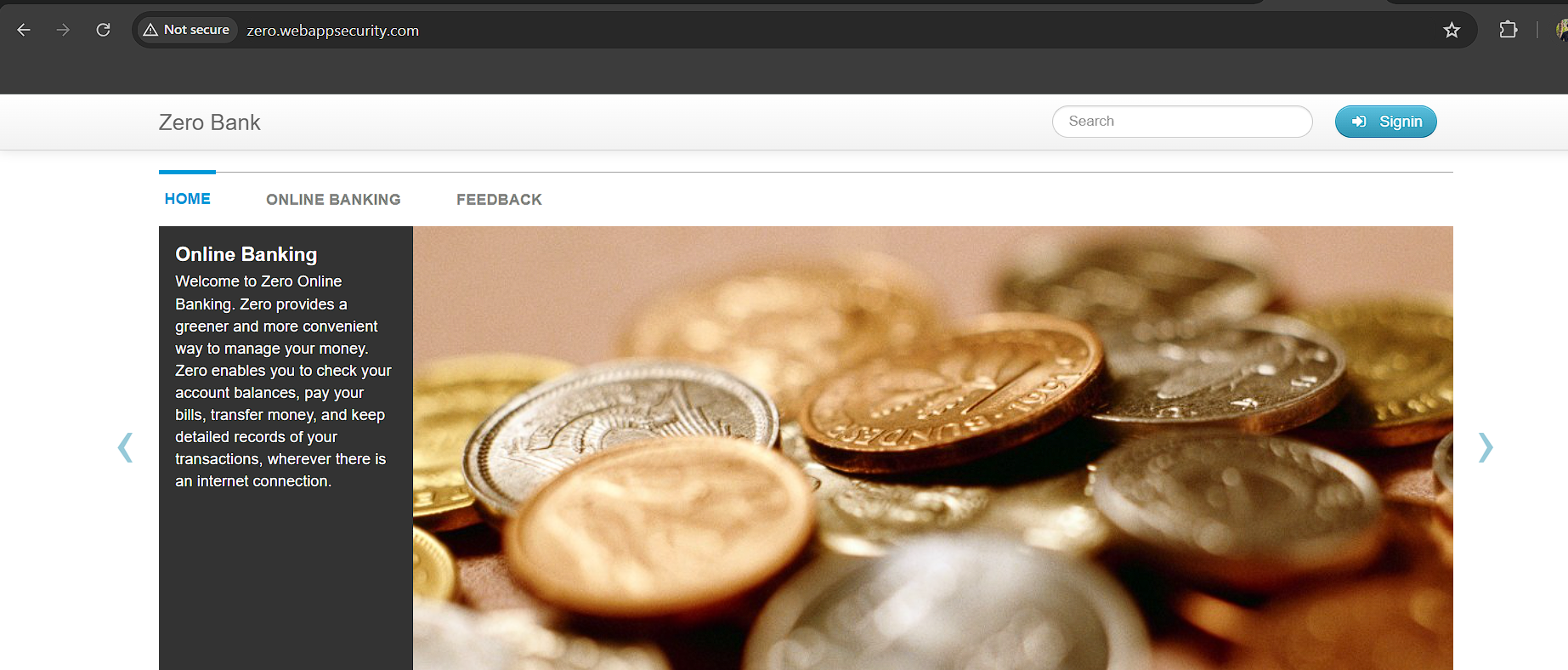
In the banking application using http. That data transfering is not in encrypted form. Transfer in plaintext.

## **1.2 Vulnerable instance**

<http://zero.webappsecurity.com/>

## **1.3 Proof of concept.**

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



## **1.4 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.

# **2.Unencrypted url.**

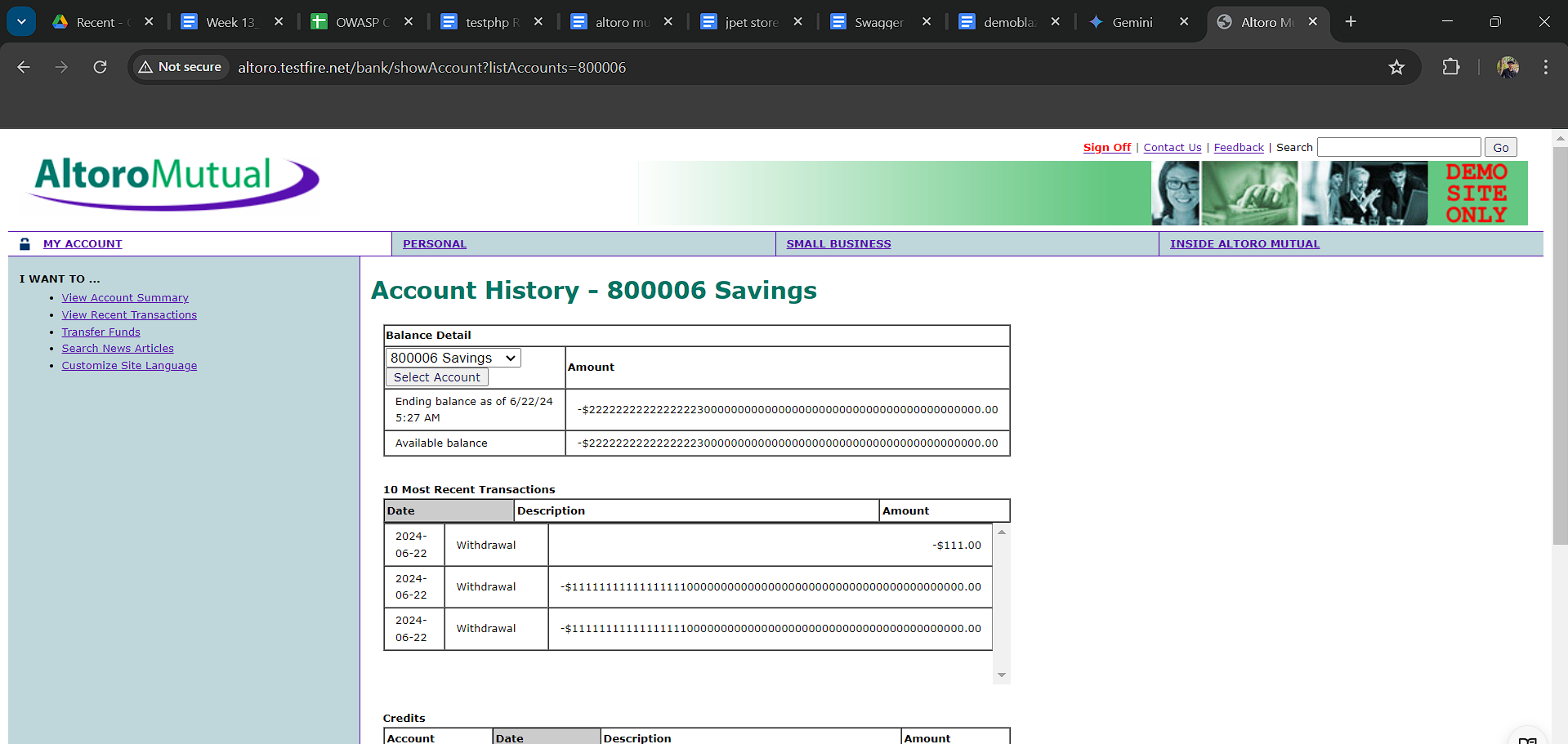
## **2.1 Description**

This web application , in the login page the token is passed in the url. unencrypted URL, exposing sensitive transfer details during the process.

## **2.2 Vulnerable instance**

<https://zero.webappsecurity.com/auth/accept-certs.html?user_token=a5435c67-015f-4f90-bef6-c2c959f1be71>

## **2.3 Proof of concept.**

Go to the login page, after entering the username and password.we can see the token is in url. 

## **2.4 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

## 

# **3.IDOR**

## **3.1 Description**

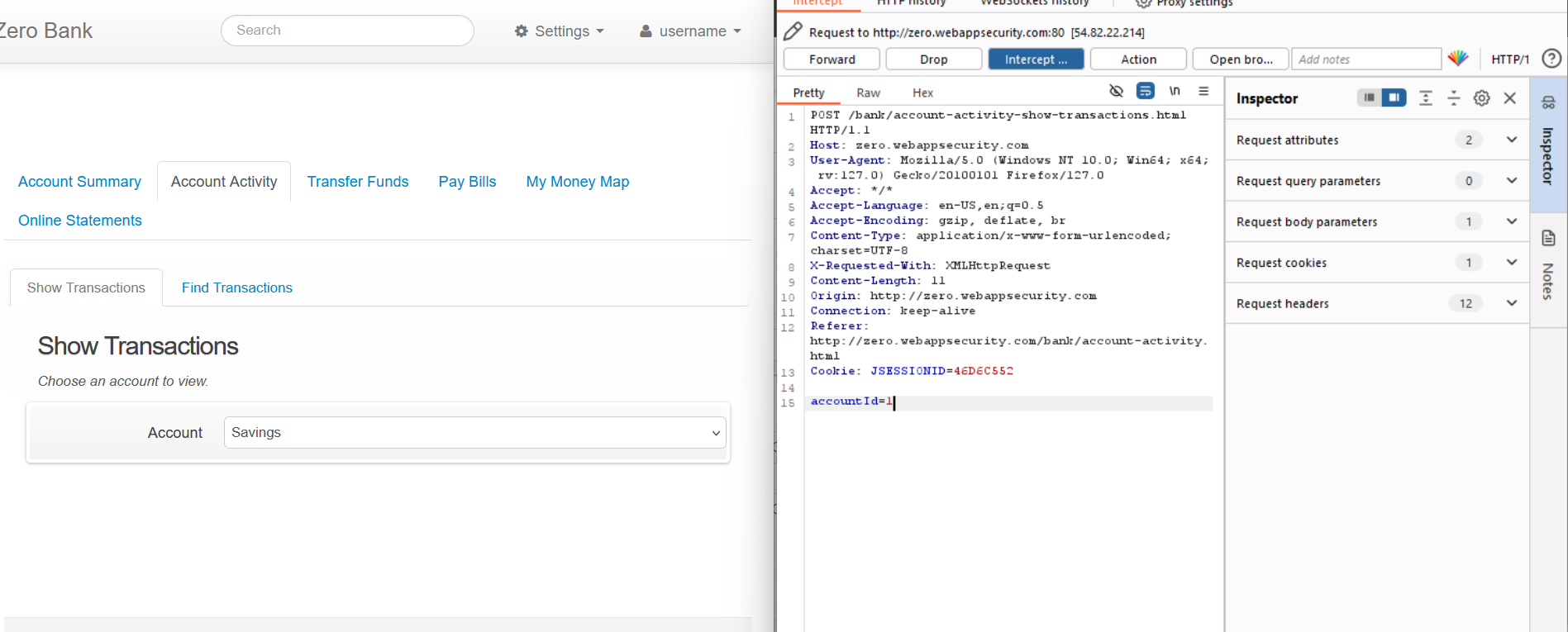
This web application transmits user IDs unencrypted in the summary page, allowing attackers to intercept requests, modify IDs, and potentially access sensitive data of other users.

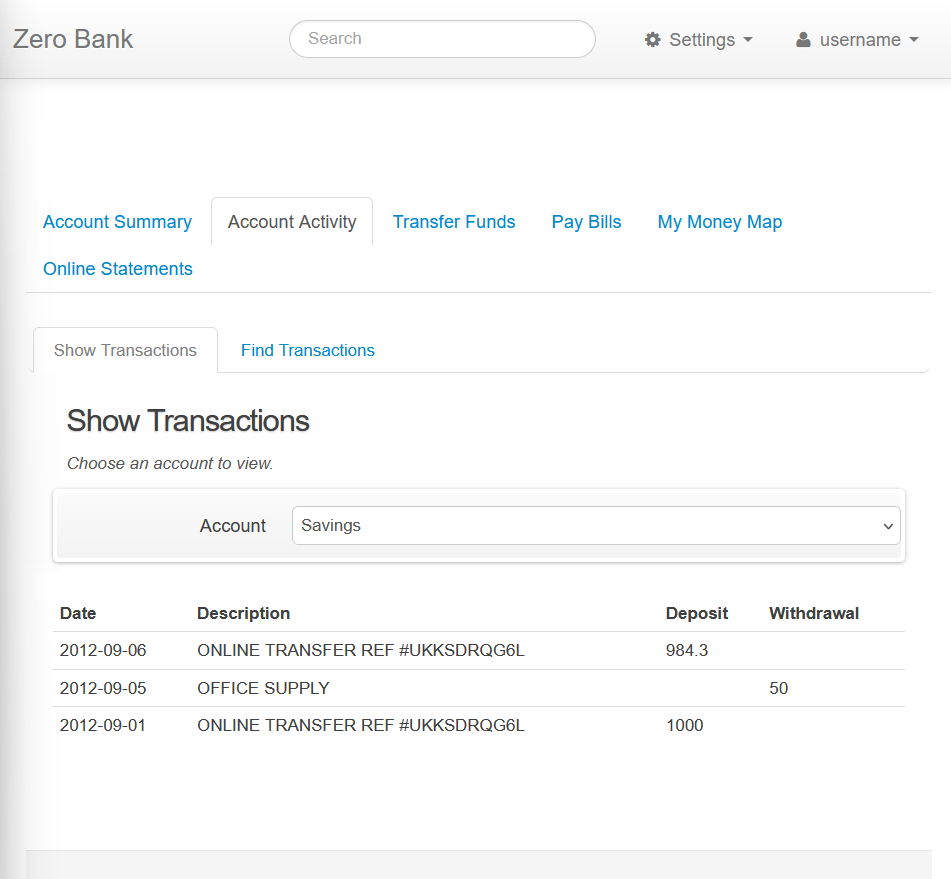
## **3.2 Vulnerable instance**

<http://zero.webappsecurity.com/bank/account-activity.html>

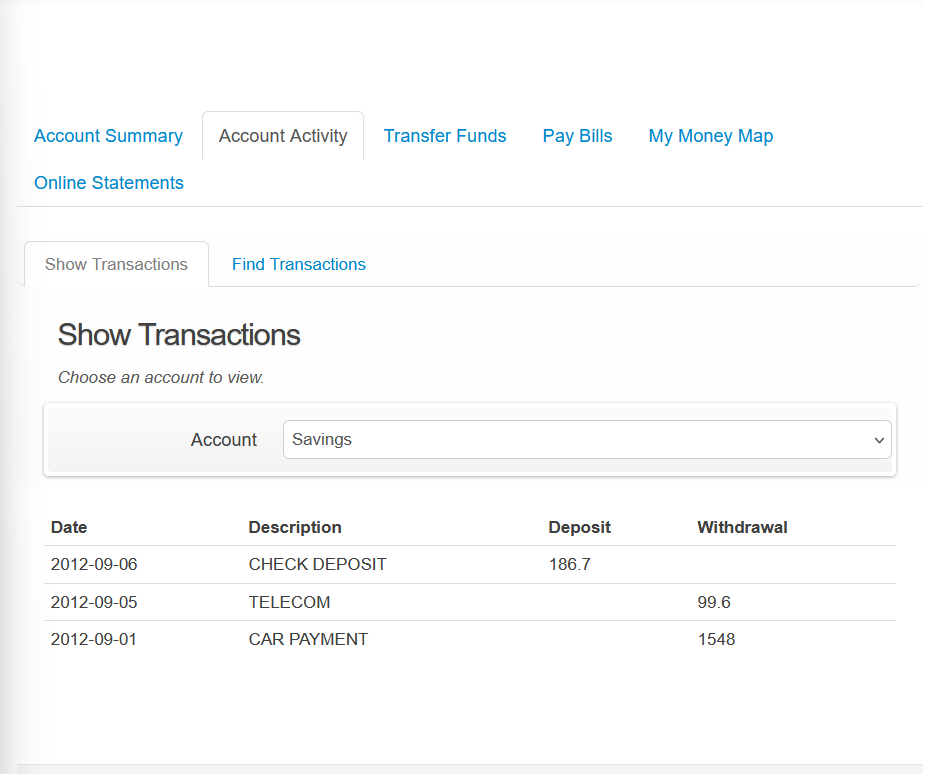
## **3.3 Proof of concept.**

Step 1: go to the summary page , interrupt the request. The id is in plain text.





Step 2 : change the id to”2”.we get another account details.



## **3.4 Mitigation**

**Use Session Tokens:** Instead of user IDs directly in the URL, consider using secure session tokens. These are temporary, unique identifiers generated by the server and stored on the user's side.

**Server-Side Data Access Control:** Even if session tokens are used, ensure proper access control mechanisms exist on the server-side.

# **4.Privilege access.**

## **4.1 Description**

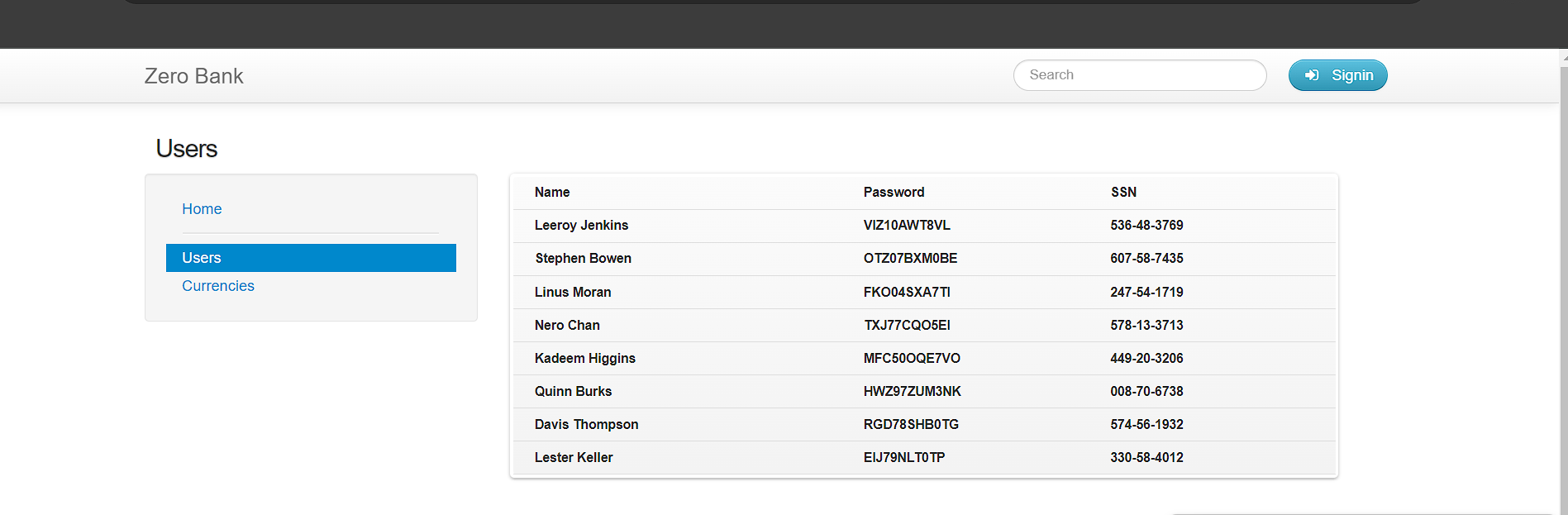
This web application possesses a critical security vulnerability that leaves the administrative panel wide open for exploitation. By simply appending "/admin" to the website's URL, any unauthorized user can gain unrestricted access to the admin directory.

## **4.2 Vulnerable instance**

<http://zero.webappsecurity.com/admin/>

## **4.3 Proof of concept.**

Add “/admin” to the url .Then we can directly access the admin page. In that directory we can see the user details, password,etc..



## **4.4 Mitigation**

**Authentication:** Enforce a robust login mechanism for accessing the admin panel. This could involve username and password login with strong password policies (minimum length, complexity requirements) or multi-factor authentication (MFA) that requires a secondary verification code for added security.

**Authorization:** Implement role-based access control (RBAC). This ensures that only authorized users with specific roles have access to specific functionalities within the admin panel.

**Minimize Access:** Grant administrative access only to users who absolutely require it for their job functions. This minimizes the number of potential targets for attackers and the potential damage they could cause.

# **5.Improper payment validation**

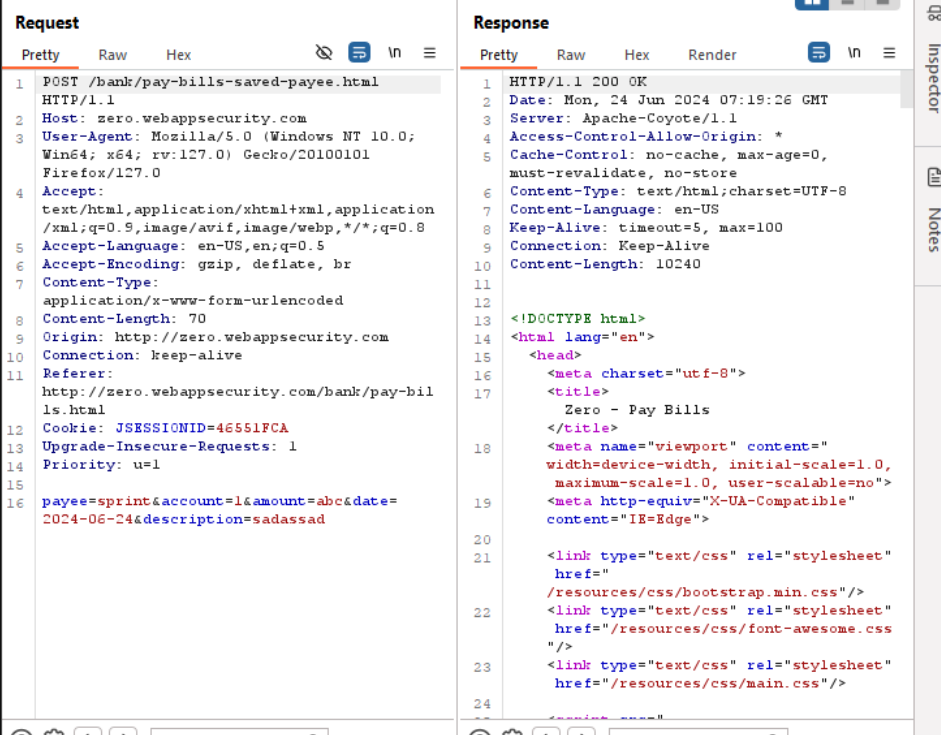
## **5.1 Description**

In this application there is an option for paying bills. There is an input box to add an amount. There is no validation in that input box.

## **5.2 Vulnerable instance**

<http://zero.webappsecurity.com/bank/pay-bills.html>

## **5.3 Proof of concept.**

Go to the bill payment page ,there is an option for adding an amount . we can add “abc “ to that option that will execute successfully.

## **5.4 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.

# **6.improper validation**

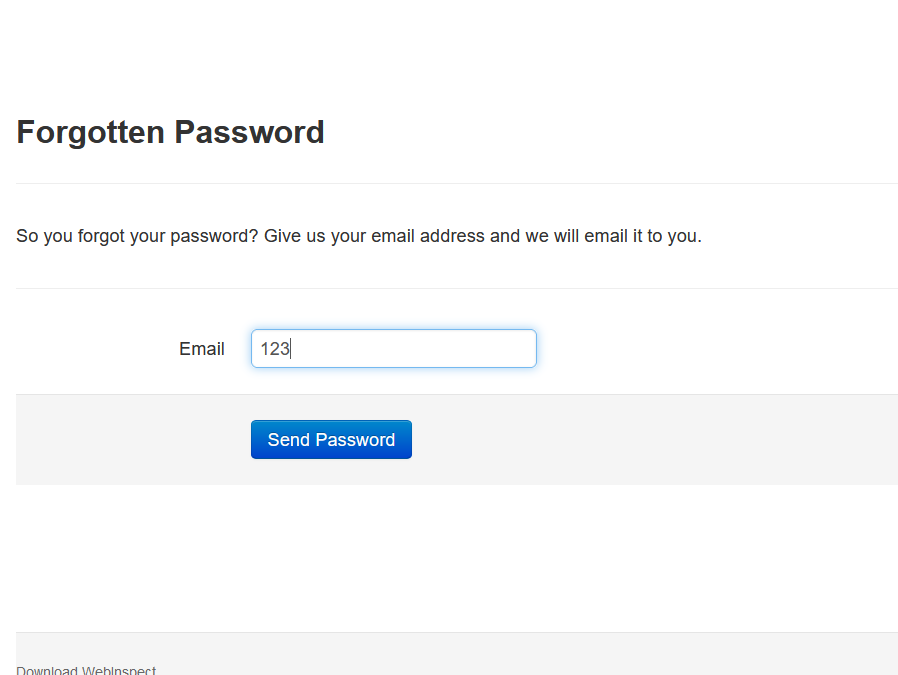
## **6.1 Description**

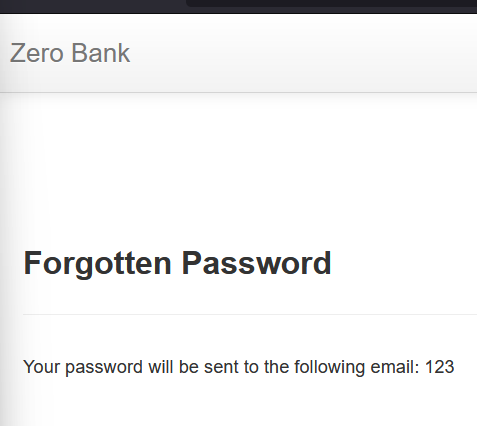
The web application suffers from a security vulnerability due to improper input validation. Specifically, the registration process doesn't validate email addresses effectively. This means a user could enter anything, including letters or symbols, in these fields.

## **6.2 Vulnerable instance**

<http://zero.webappsecurity.com/forgot-password.html>

## **6.3 Proof of concept.**

**Step 1:** Go to the forgot password page . There is adoption to add email.

We can see there is no validation in that page . 

## **6.4 Mitigation**

**Email Verification:** For email addresses, consider sending a verification email to confirm the user's ownership and prevent fake accounts.

**Sanitize user input:** After validation, sanitize user input to remove any remaining malicious code or characters that might have slipped through. This helps prevent further vulnerabilities down the line.

# **7.Validation error**

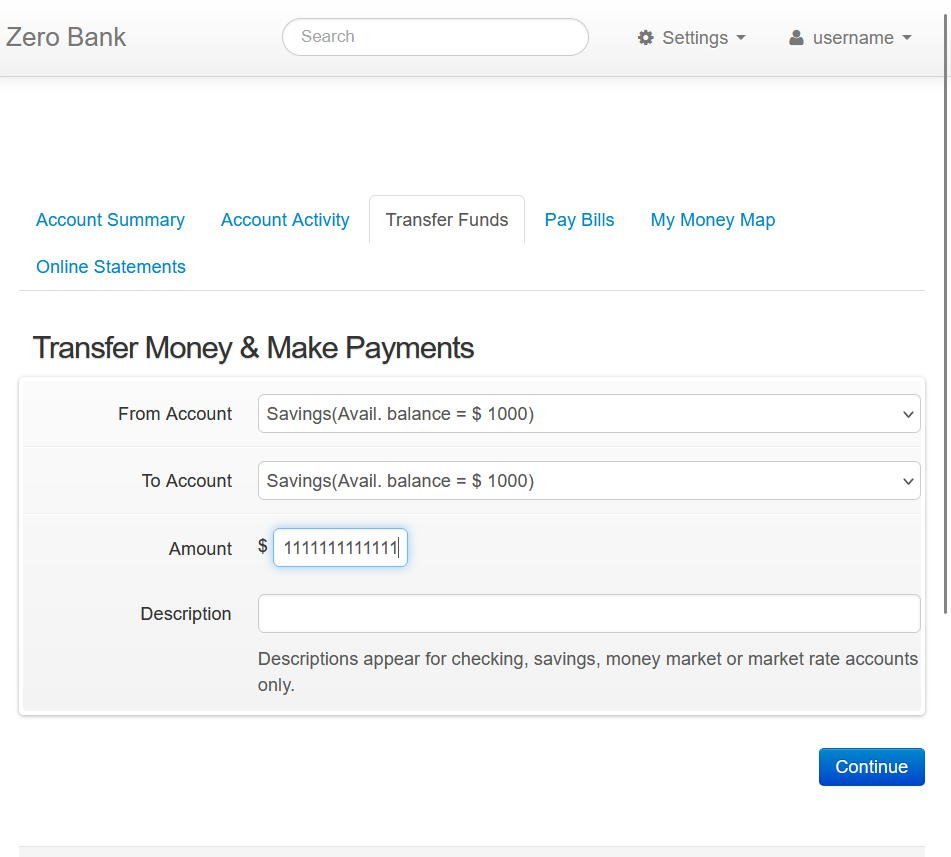
## **7.1 Description**

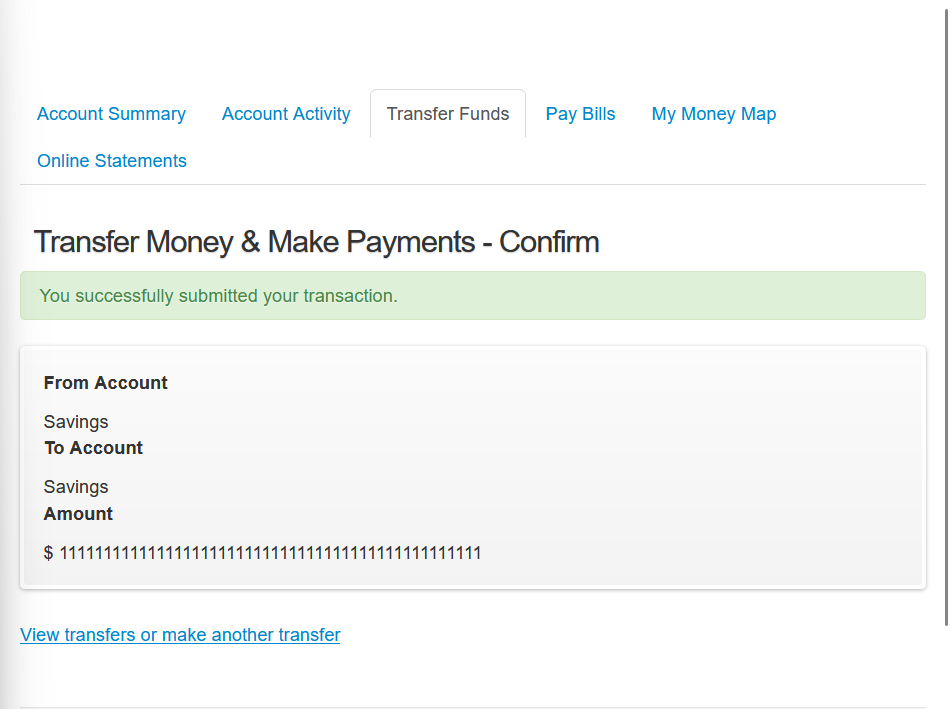
A critical security flaw exists within the money transfer function of this banking application. The functionality lacks validation for the transfer amount. This means users can enter any value into the transfer field, regardless of how large or negative the number might be. This absence of validation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

## **7.2 Vulnerable instance**

<http://zero.webappsecurity.com/bank/transfer-funds.html>

## **7.3 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.



## **7.4 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.

# **8.Server Status page access**

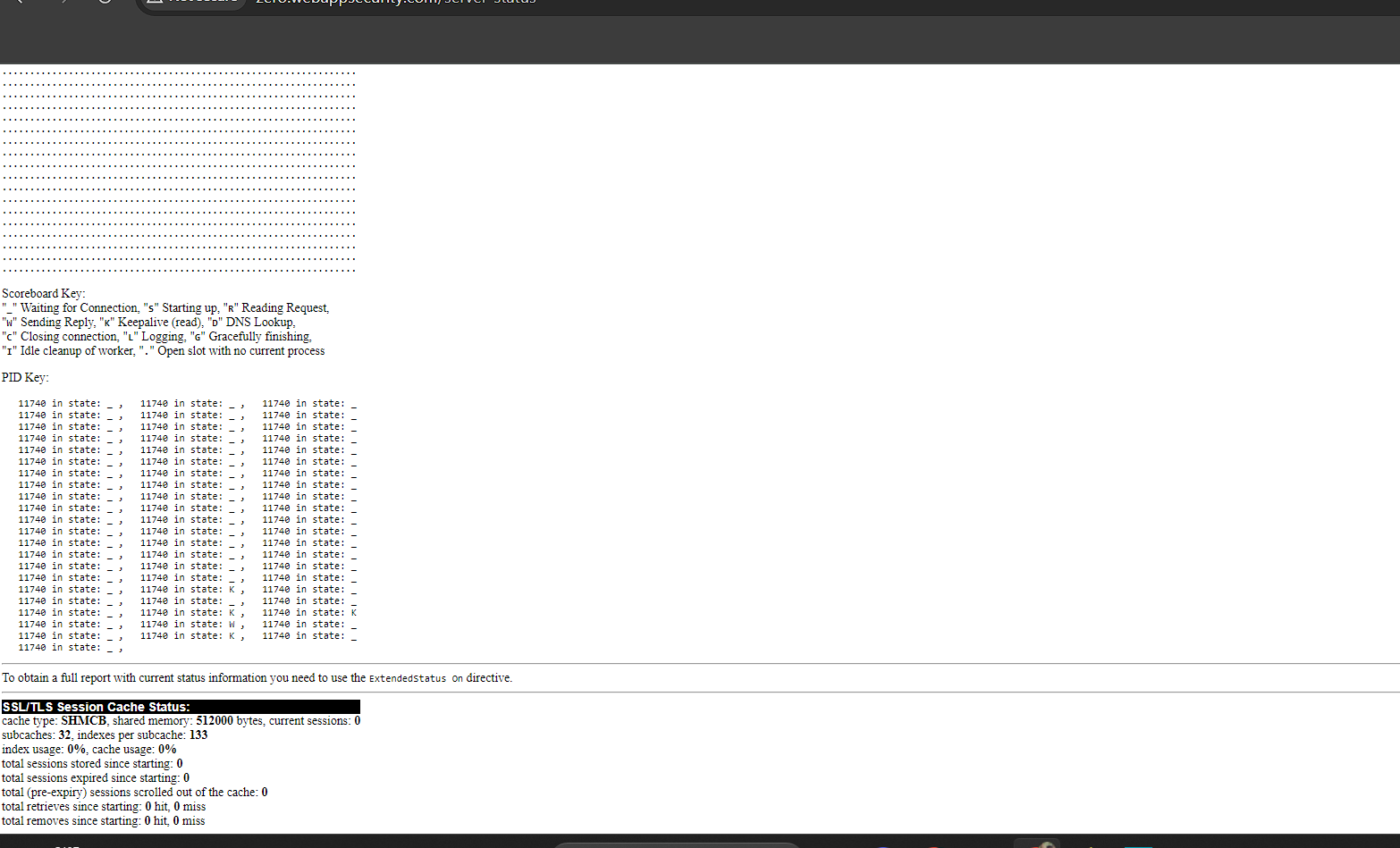
## **8.1 Description**

Securing server status page access involves authentication and authorization controls to restrict entry. Implementing encryption protocols like HTTPS ensures secure communication channels. Access logs and monitoring tools help track and detect suspicious activities in real-time..

## **8.2 Vulnerable instance**

<http://zero.webappsecurity.com/server-status>

## **8.3 Proof of concept.**

We can access the server status page.

## **8.4 Mitigation**

Mitigation strategies for securing server status page access include implementing strong authentication methods like multi-factor authentication (MFA) and role-based access controls.

# **9.XSS**

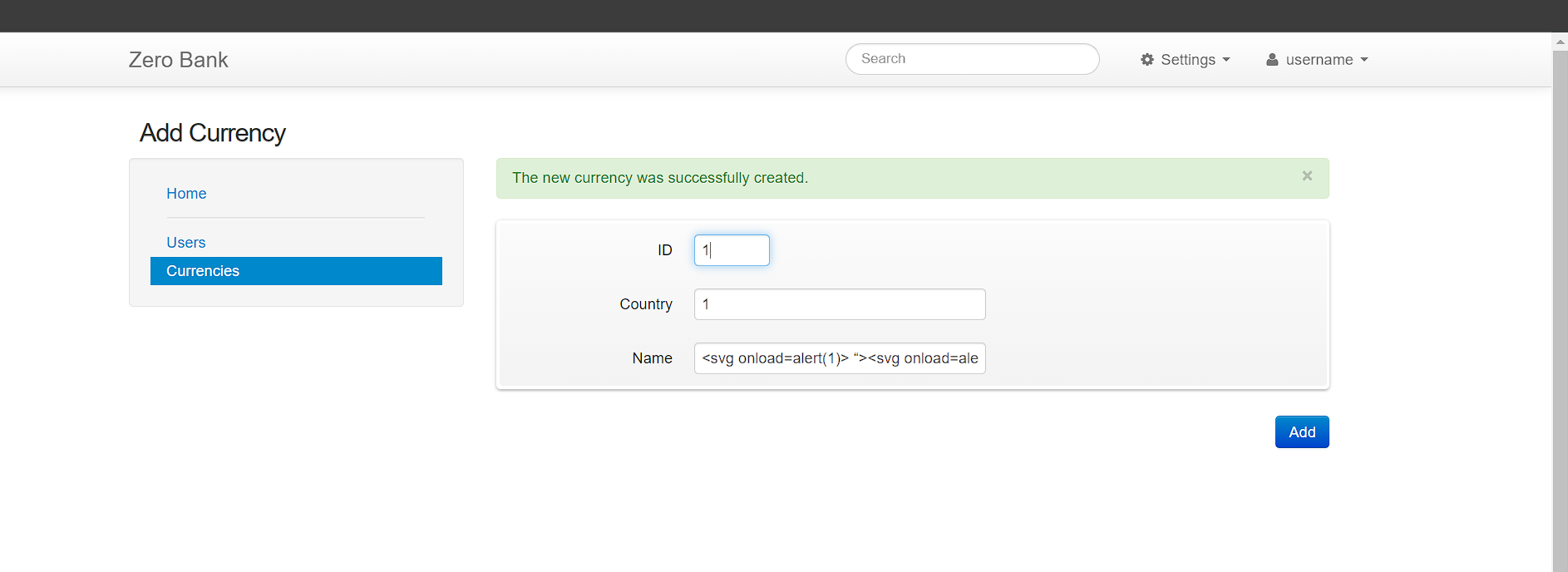
## **9.1 Description**

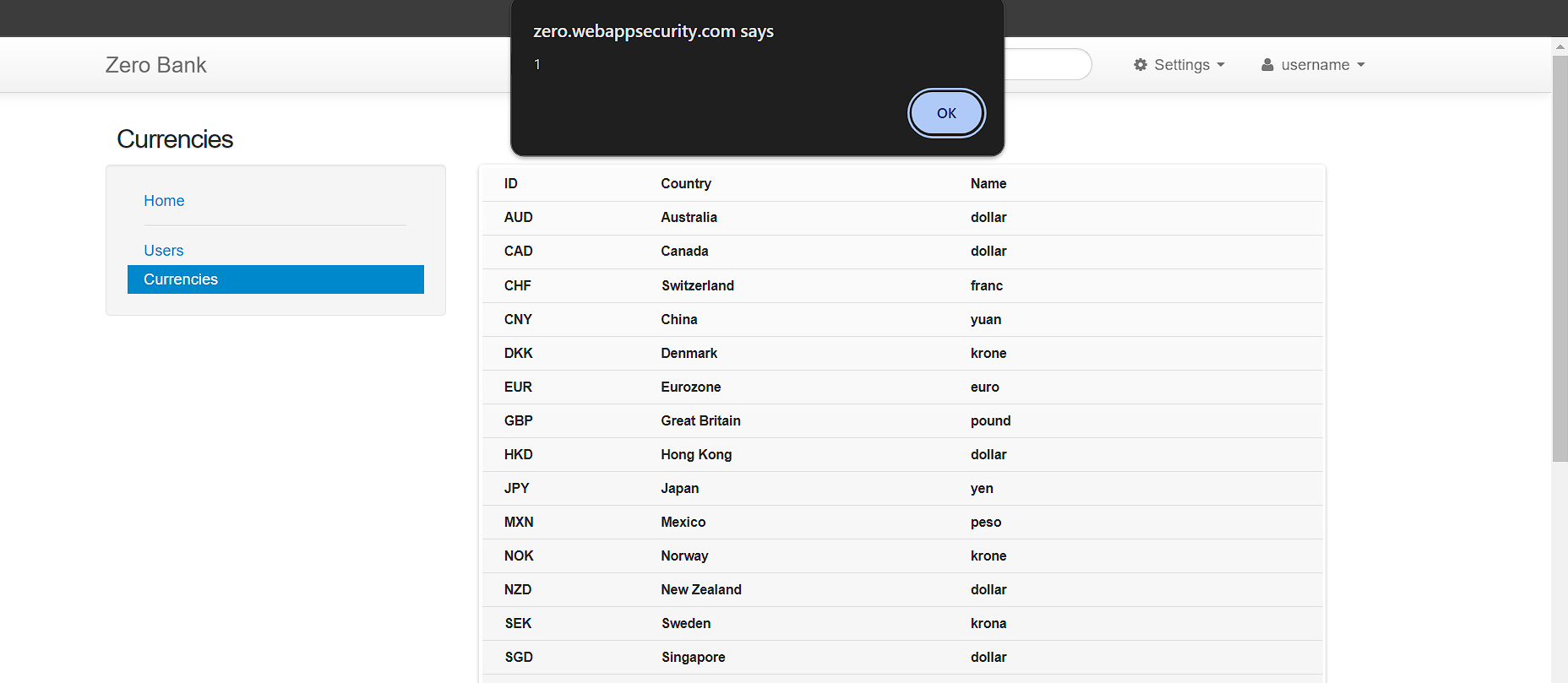
The web application suffers from a critical security issue known as an XSS (Cross-Site Scripting) vulnerability within its search functionality. This means an attacker could potentially inject malicious scripts into the search box. These scripts would then be executed by the web application itself, granting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

## **9.2 Vulnerable instance**

<http://zero.webappsecurity.com/admin/currencies.html>

## **9.3 Proof of concept.**

Go to the admin page .There is an option to add currencies. Add this payload to that option 

We can see that payload is successfully executed 

## **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.Information Disclosure.**

## **10.1 Description**

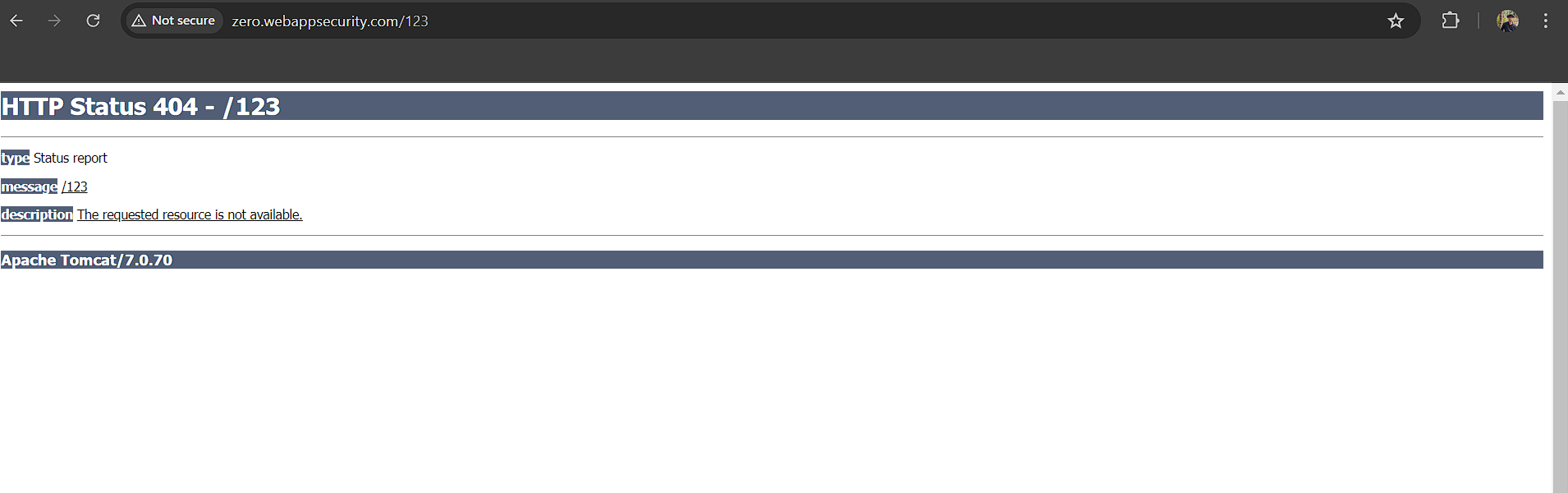
In the 404 page we can see the server version .

## **10.2 Vulnerable instance**

<http://zero.webappsecurity.com/123>

## **10.3 Proof of concept.**

Go to the 404 page and we can see the server version.



## **10.4 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.

# **11.outdated version.**

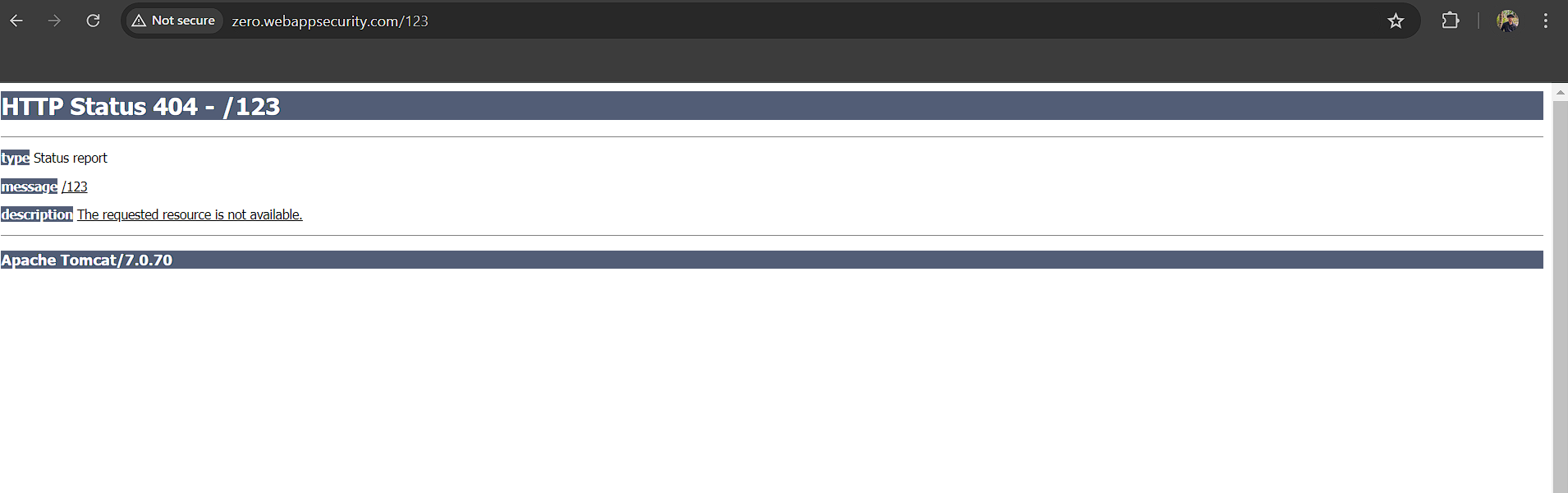
## **11.1 Description**

In the 404 page we can see the server version. In this web application version is Apache Tomcat/7.0.70 the latest version is Tomcat /10.1.0 .

## **11.2 Vulnerable instance.**

<http://zero.webappsecurity.com/123>

## **11.3 Proof of concept.**



## **11.4 Mitigation**

**Upgrade Tomcat:** While mitigating information disclosure is important, it's also crucial to address the outdated Tomcat version (7.0.70). This older version might have known security vulnerabilities that could be exploited by attackers.