**NAME :** SHAIK.M.BILALDEEN

**COLLEGE :** ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES, TIRUPATI.

**BRANCH :** E.C.E

**ASSIGNMENT-4**

**Overview of OWASP Top 10 Vulnerabilities:**

* **Top 10 OWASP Vulnerability and their potential impacts:**

**Broken Access Control (A01:2021)**: Ranked as the foremost vulnerability, Broken Access Control pertains to insufficiently managed access controls. It's crucial to verify that users are only able to access permitted resources and functionalities.

**Business Impact**: Inadequate management of access controls may result in unauthorized entry, data breaches, and compromised user accounts, leading to financial harm, damage to reputation, and legal ramifications.

**Solution:** Implement robust access controls, role-based permissions, and thorough testing to prevent unauthorized access.

**Cryptographic Failures (A02:2021)**: Formerly labeled as "Sensitive Data Exposure," this classification emphasizes cryptographic challenges. Encryption failures have the potential to result in the exposure of sensitive data or compromise the system.

**Business Impact:** Flaws in cryptography can expose confidential information, like customer data or payment particulars. Such breaches can undermine trust, damage brand credibility, and result in legal repercussions.

**Solution:**  Use strong encryption algorithms, secure key management, and regular security audits.

**Injection (A03:2021)**: Injection vulnerabilities continue to be a major issue. 94% of applications underwent testing for some type of injection. Cross-site Scripting (XSS) is also included in this classification.

**Business Impact**: Injection vulnerabilities (such as SQL injection or command injection) enable attackers to tamper with data, run unauthorized code, or obtain unauthorized entry. These actions can disrupt services, compromise data integrity, and adversely affect business operations.

**Solution**: Verify and cleanse user input, employ parameterized queries, and steer clear of executing dynamic code.

**Insecure Design (A04:2021)**: A recent inclusion in the list, this category highlights design weaknesses. Threat modeling, secure design patterns, and reference architectures are pivotal in mitigating this risk.

**Business Impact**: Systems with inadequate design may possess inherent vulnerabilities that are challenging to rectify in the future. These weaknesses can result in expensive overhauls, security breaches, and delays in project completion.

**Solution**: Conduct threat modeling, follow secure design principles, and involve security experts from the beginning of the development stage.

**Security Misconfiguration (A05:2021)**: As customizable software becomes increasingly prevalent, misconfigurations are on the rise. Ninety percent of applications underwent testing for various types of misconfigurations.

**Business Impact:** Misconfigurations expose sensitive information, weaken security measures, and establish pathways for attackers to exploit. They can lead to data breaches, service interruptions, and breaches of regulatory requirements.

**Solution**: Consistently assess configurations, adhere to the highest security standards, and automate configuration evaluations.

**Vulnerable and Outdated Components (A06:2021)**: This category, formerly known as "Using Components with Known Vulnerabilities," advances from its previous ranking.

**Business Impact**: Employing obsolete or susceptible components can introduce security vulnerabilities. Breaches stemming from unpatched libraries can result in financial setbacks, legal responsibilities, and reputational harm.

**Solution**: Keep track of dependencies, promptly apply patches, and utilize tools to identify vulnerable components.

**Identification and Authentication Failures (A07:2021)**: Formerly labeled as "Broken Authentication," this category now encompasses CWEs associated with identification lapses. The implementation of standardized frameworks is aiding in mitigating this risk.

**Business Impact**: Inadequate authentication methods can enable unauthorized entry, resulting in data breaches, compromised accounts, and financial fraudulent activities.

**Solution**: Introduce multi-factor authentication, enforce robust password guidelines, and regularly conduct security evaluations.

**XML external entities (XXE) (A08:2021):** XML External Entities (XXE) serve as a method to reference data in an XML document through an identifier, rather than embedding the actual data content.

**Business Impact:** This situation could result in the exposure of confidential information, the execution of SSRF attacks, the disruption of service, and the potential for endangering the server or its supporting systems.

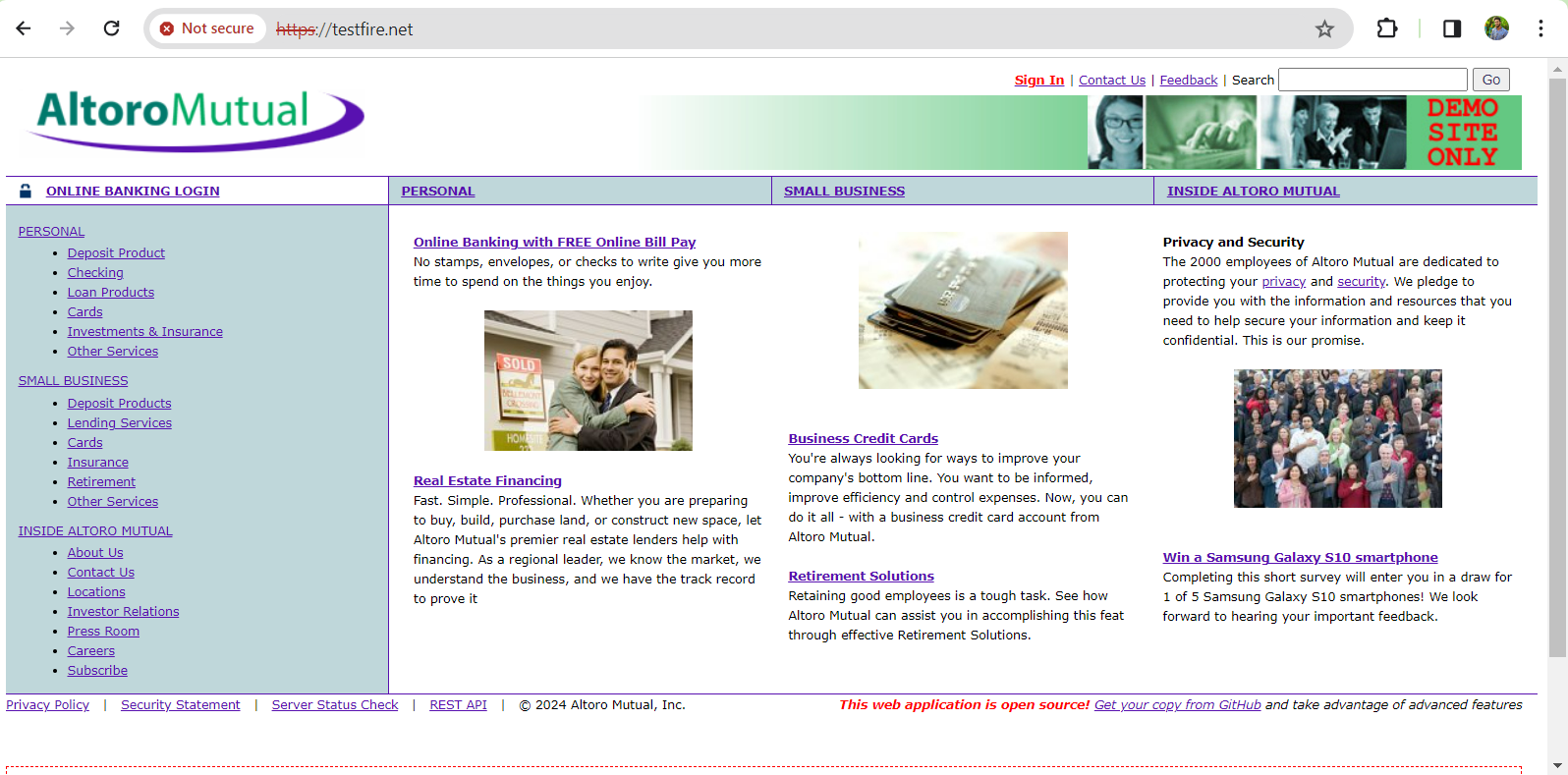
**Solution:** By using less complex data formats like JSON when possible. By upgrading all XML processors and libraries to patch known vulnerabilities. And by disabling XML external entity and DTD processing in all XML parsers.

* **Altro Mutual Website Analysis:**

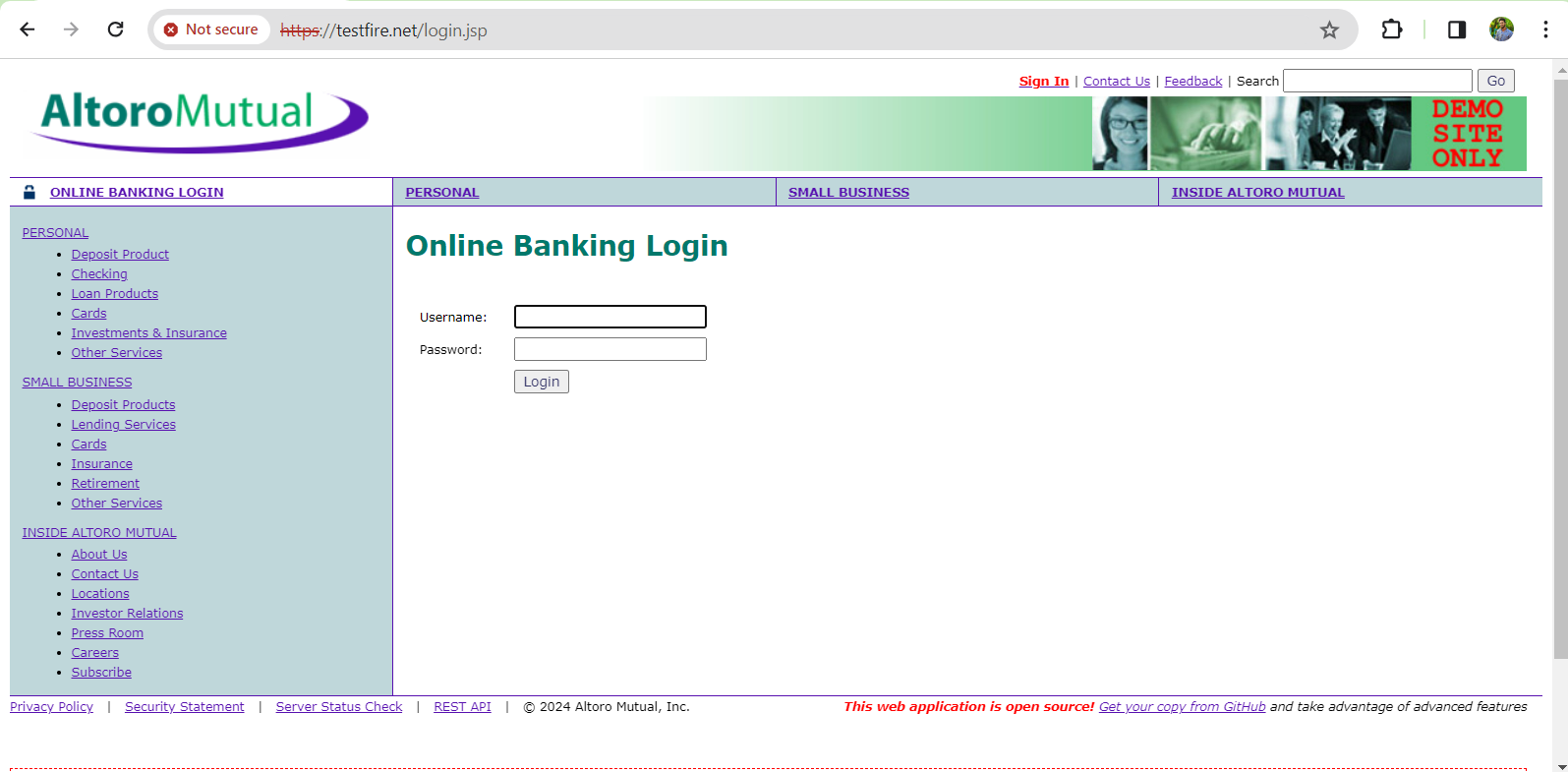
The review of Altoro Mutual’s website reveals that Oxytis Forensics carried out an extensive security evaluation. This evaluation incorporated penetration tests guided by the OWASP Top 10, a recognized document for developer awareness and web application security. The findings spotlighted several security gaps that could potentially lead to unauthorized system access or data breaches.

Moreover, Altoro Mutual underscores its dedication to safeguarding client privacy and security through its array of financial services and protective measures for customer data. There is also a reference to a GitHub repository where a detailed Vulnerability Assessment & Penetration Testing (VAPT) was executed on Altoro Mutual, Inc., employing a range of web application security testing methods.

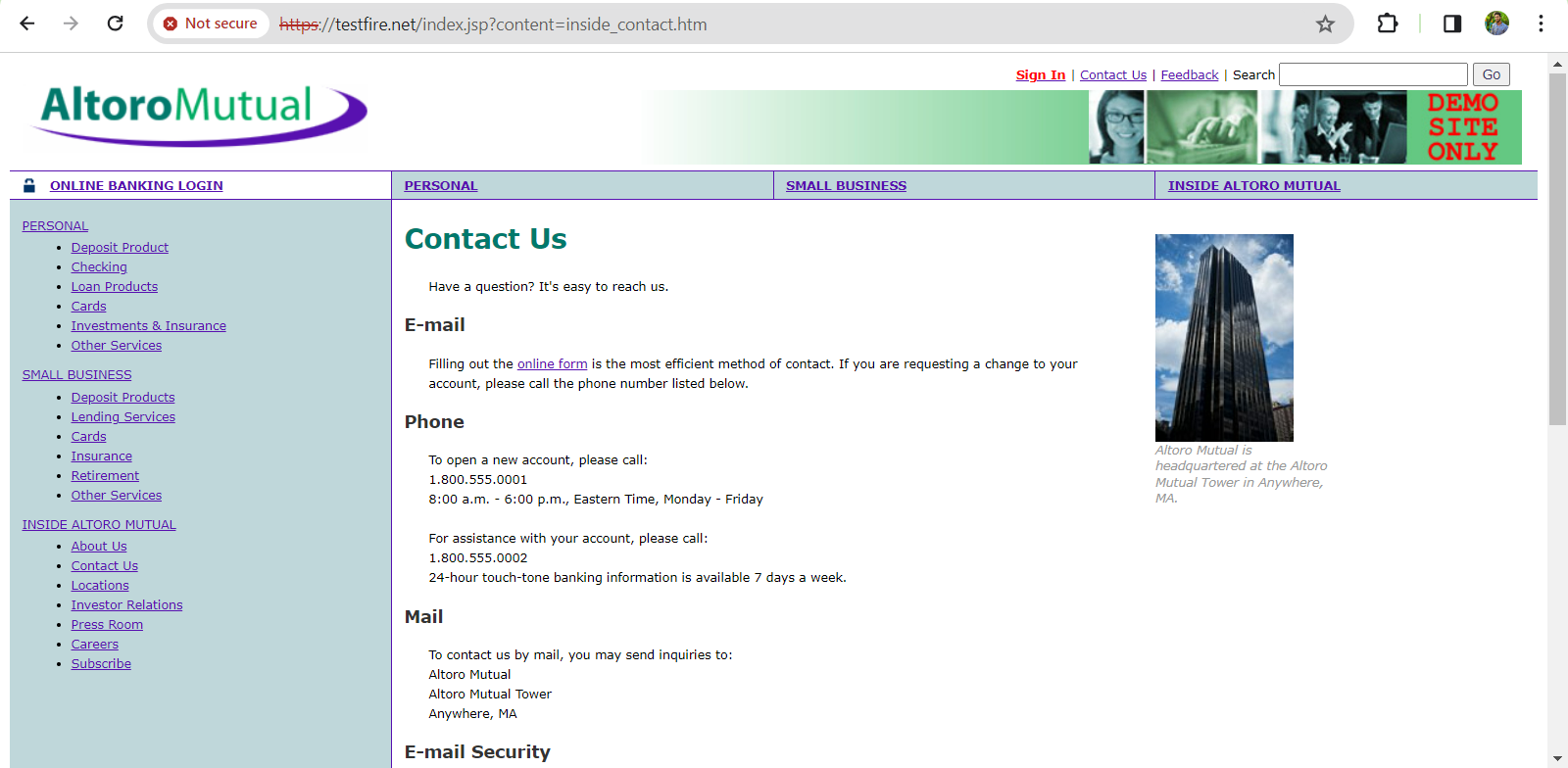
The vulnerabilities such as SQL injection, Cross-Site Scripting(XSS), Insecure authentication mechanisms, insecure direct object references and many other were identified in the Altro Mutual Website.



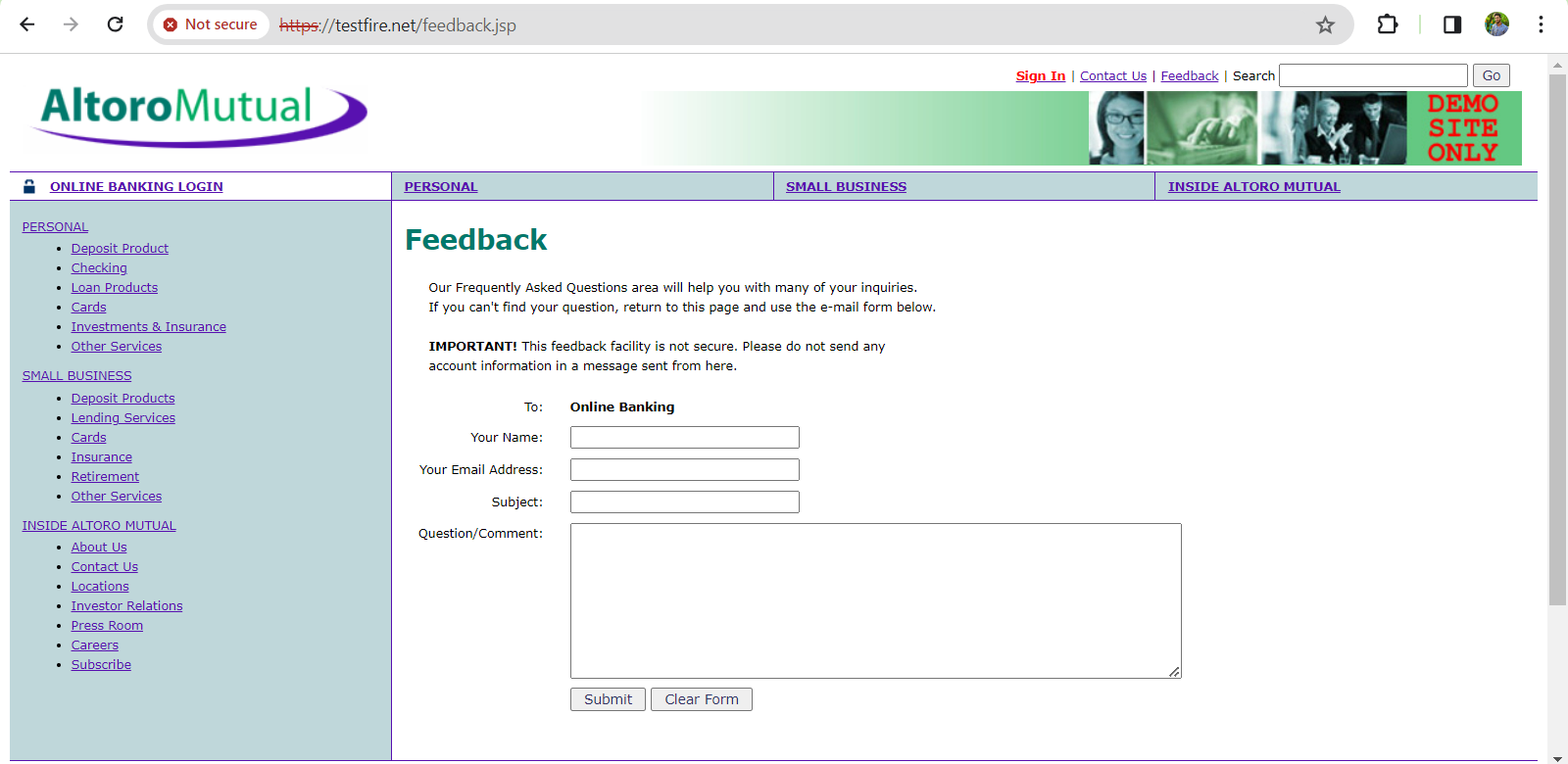
**Fig.1: Altoro Mutual Website’s home page**

****

**Fig.2: Login Page**

****

**Fig.3: Contact Info Page**

****

**Fig.4: Feedback Page**

* **Vulnerability Identification Report and Vulnerability Exploitation Demonstration:**

Altoro Mutual’s website is designed to offer a comprehensive suite of banking services, encompassing individual and commercial banking, investment opportunities, insurance options, and various loan offerings. The platform includes features for secure online banking access, detailed information about the bank’s legacy, avenues for contacting the institution, and a suite of resources dedicated to ensuring privacy and security. Additionally, the site provides practical tools for online bill settlement and property financing solutions.

The site’s design aims to streamline interactions with these offerings, enabling patrons to oversee their finances, seek loans, and obtain financial counsel. Despite this focus on user engagement, the site maintains a strong commitment to security, vowing to shield user data and privacy through robust protective measures.

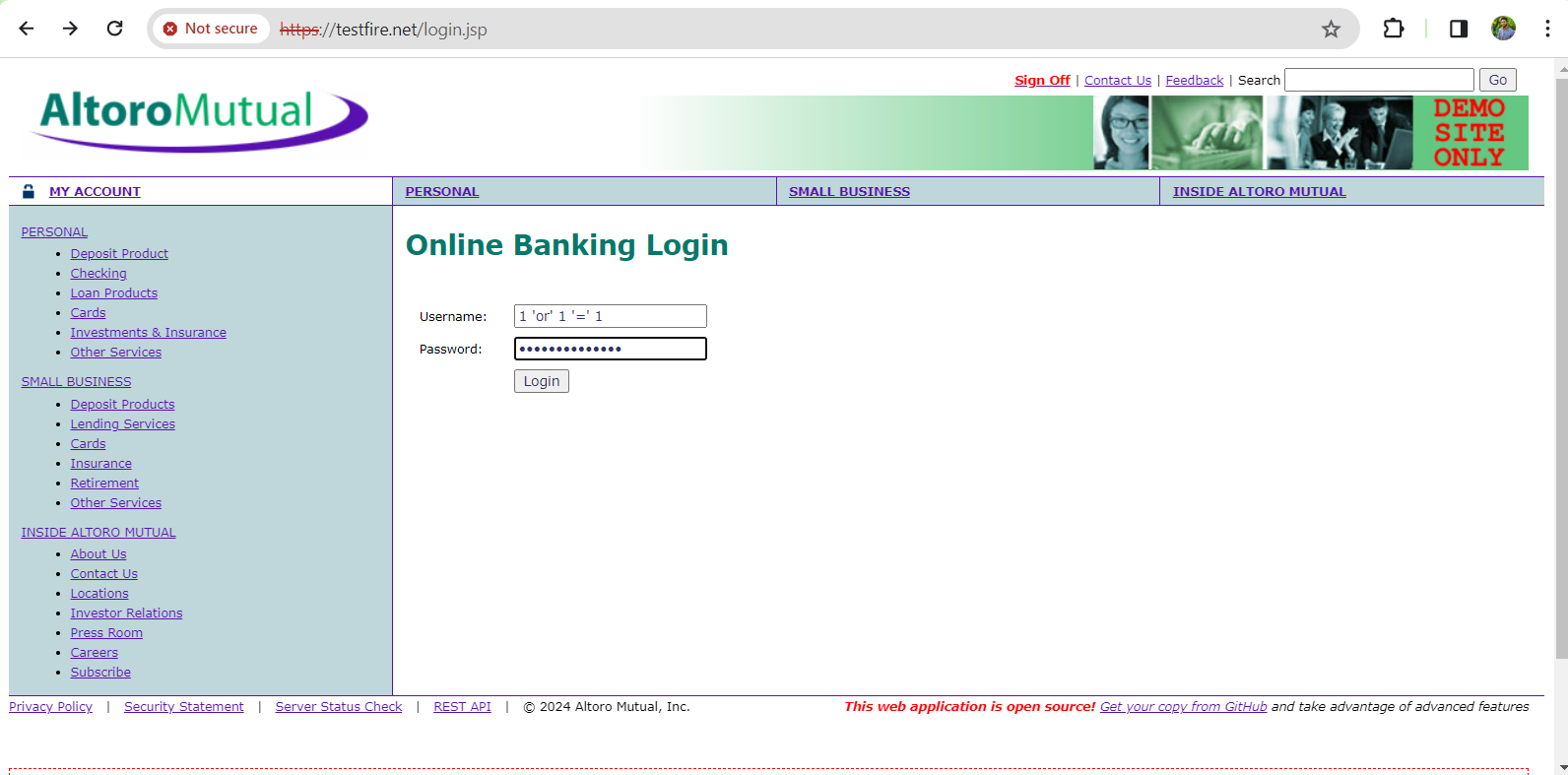
The website’s security posture reveals several potential vulnerabilities:

* **Cross-Site Scripting (XSS)**: Identified XSS flaws could permit malevolent script injection into web pages, affecting other users.
* **Insecure File Inclusions**: Potential file inclusion weaknesses might enable intruders to incorporate files on the server, leading to unauthorized access or data leakage.
* **Exposed Directory Listings**: Visible directory listings could disclose the application’s blueprint and sensitive files.
* **Misconfigured Permissions**: Incorrectly set permissions might allow unauthorized file access in specific directories, jeopardizing data security and privacy.

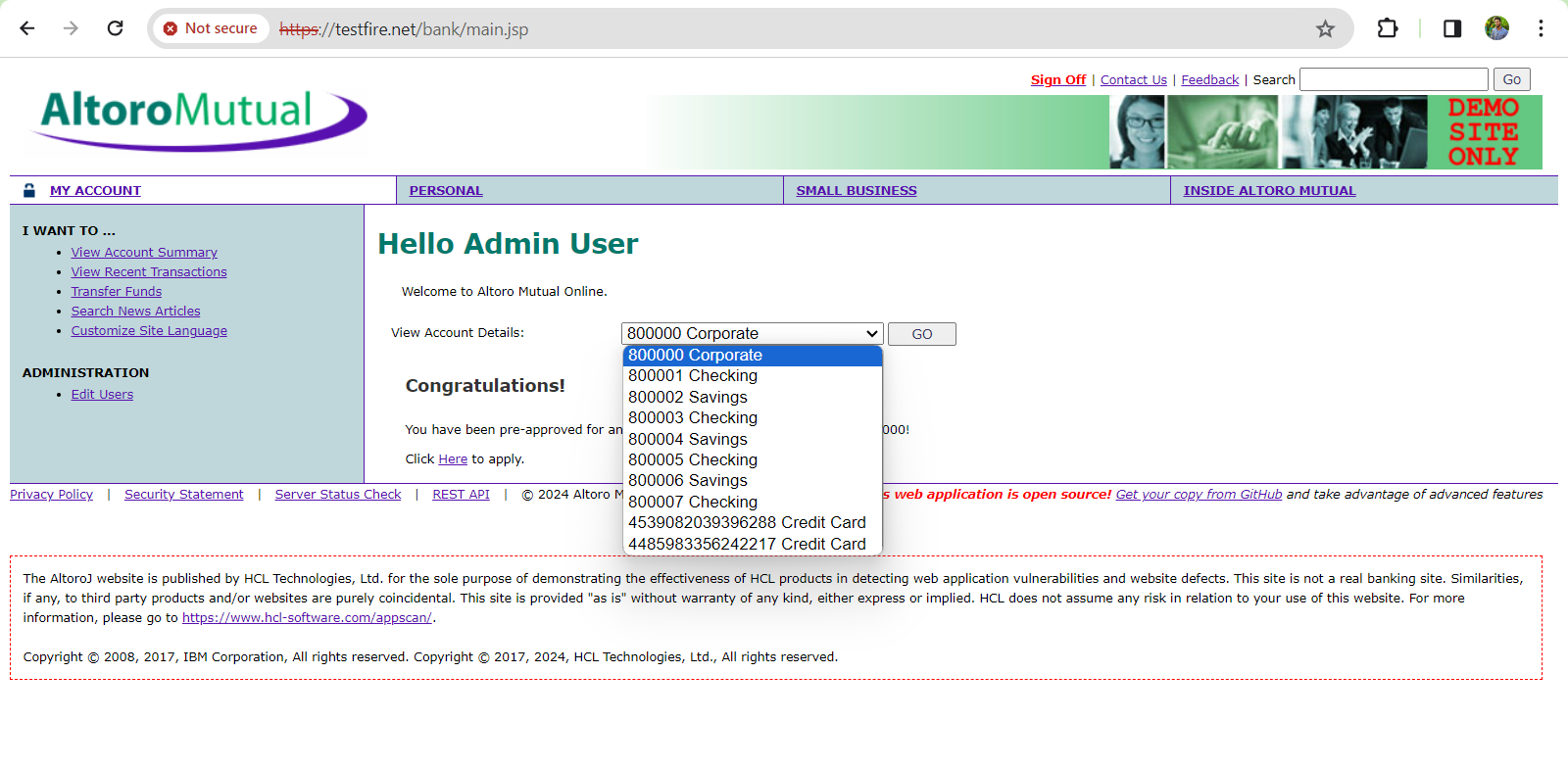
To fortify against these vulnerabilities, it is imperative to deploy stringent security protocols, including thorough input checks, secure output handling, and precise configuration of file and directory permissions. Ongoing security evaluations and adherence to web application development best practices are also vital in reducing these security risks.

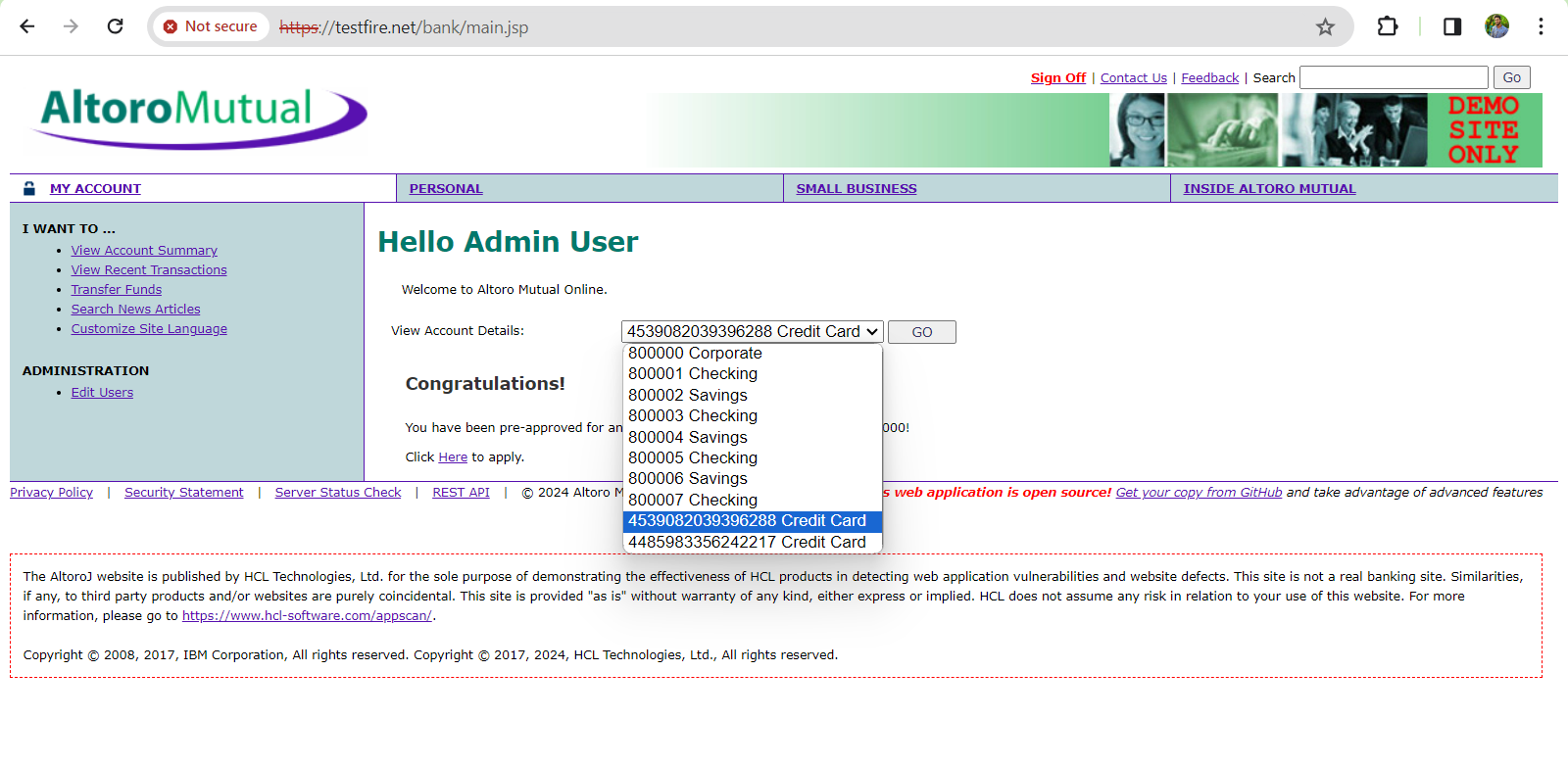
**Identification of vulnerabilities:**

We’ll be testing if website login is possible. Successful login indicates a “Broken Authorization” vulnerability, which may also result in other security issues such as Sensitive Data Exposure and Broken Access Control.

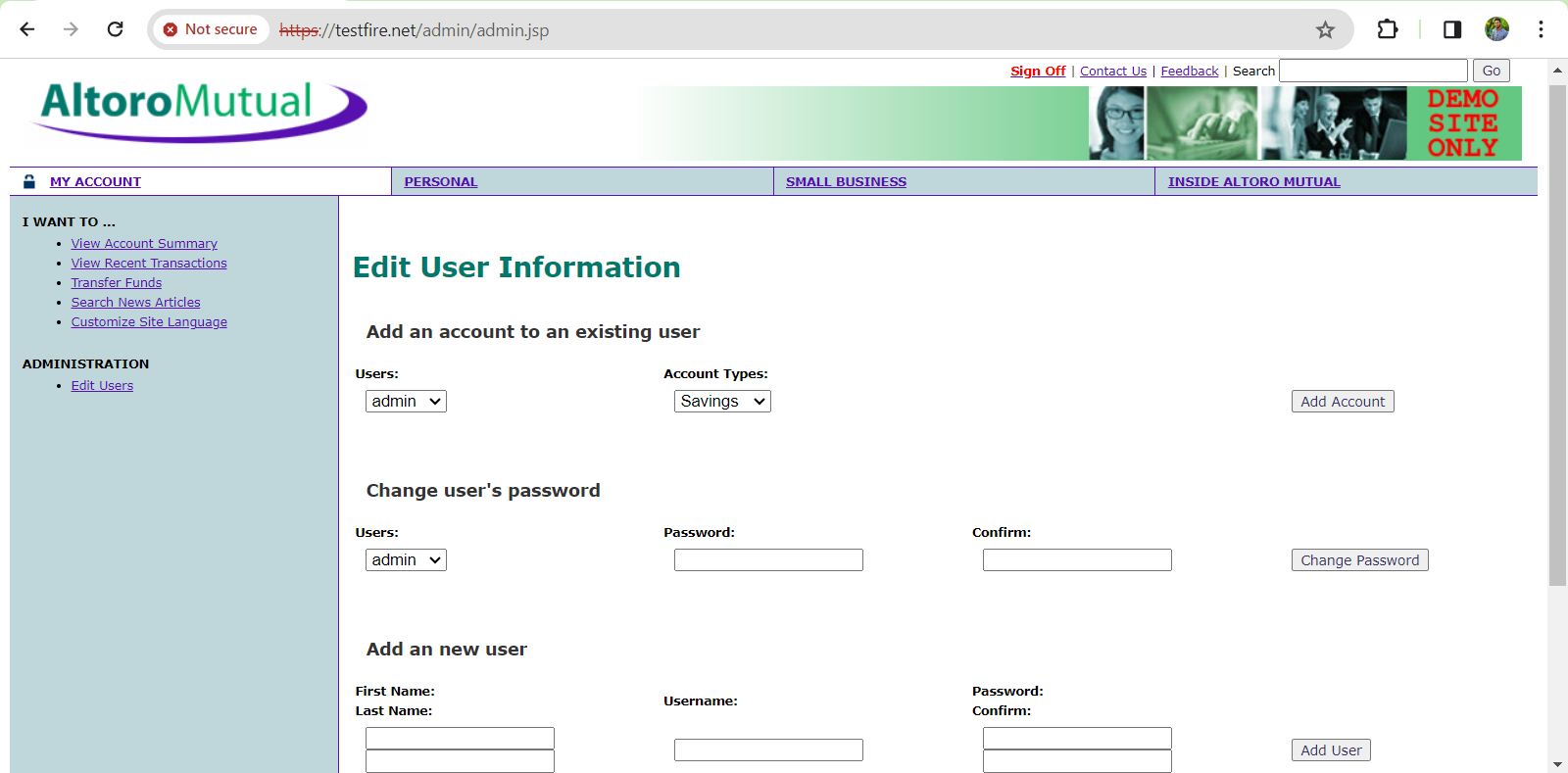


In this scenario, we utilized the payload **“1 ‘or’ 1 ‘=’ 1”** for both username and password, which allowed us to log in as an admin. This access enables us to view other user accounts, add new users, modify their passwords, and manage their accounts.

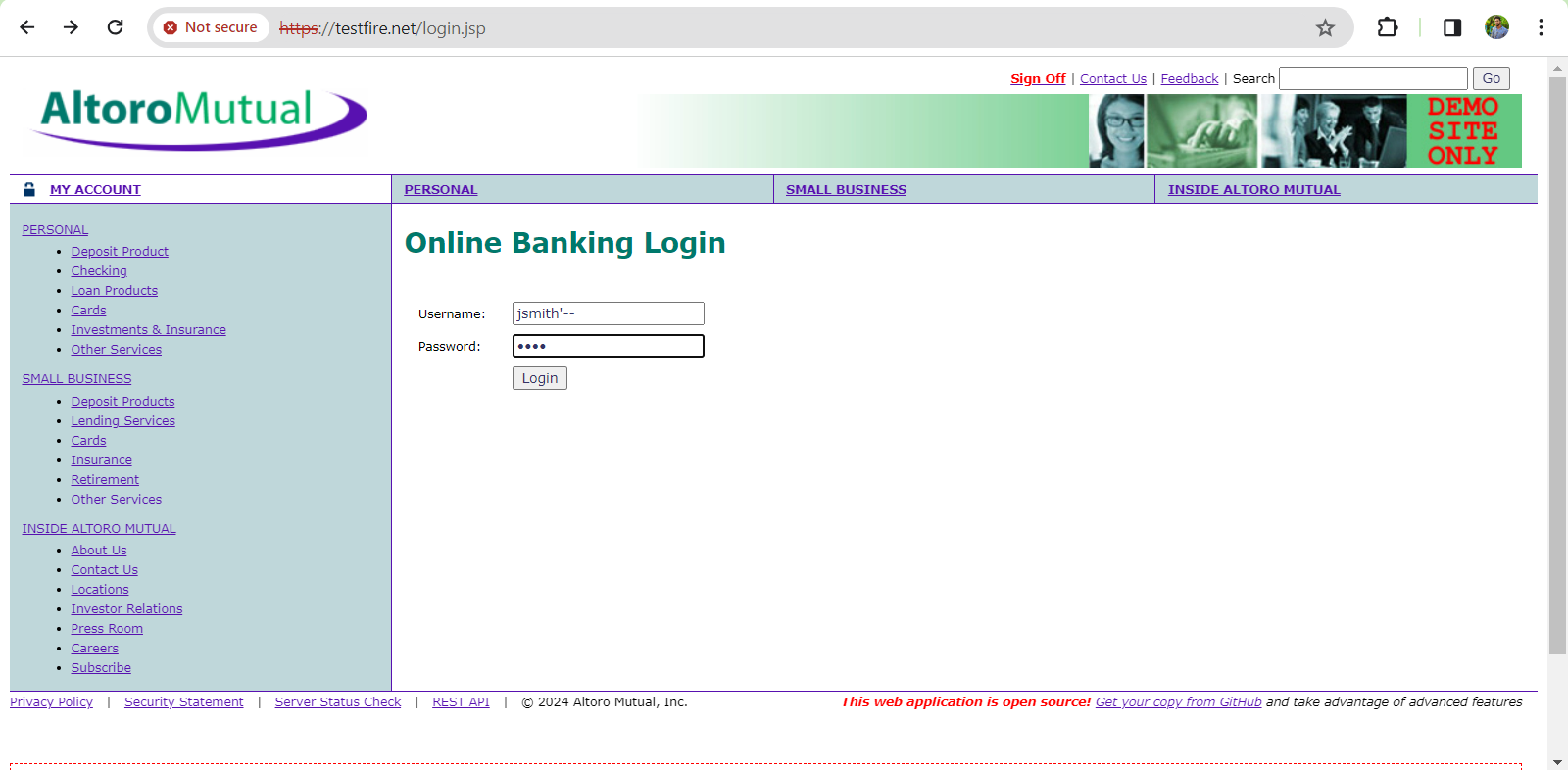
The image reveals the bank users’ account information. The emphasized section in the image corresponds to the pages detailing fund transfers, account summaries, and recent transactions. Also we can able to access different types of functionalities of the user by selecting the function. E.g.: Credit Card, Savings, Checking, etc.



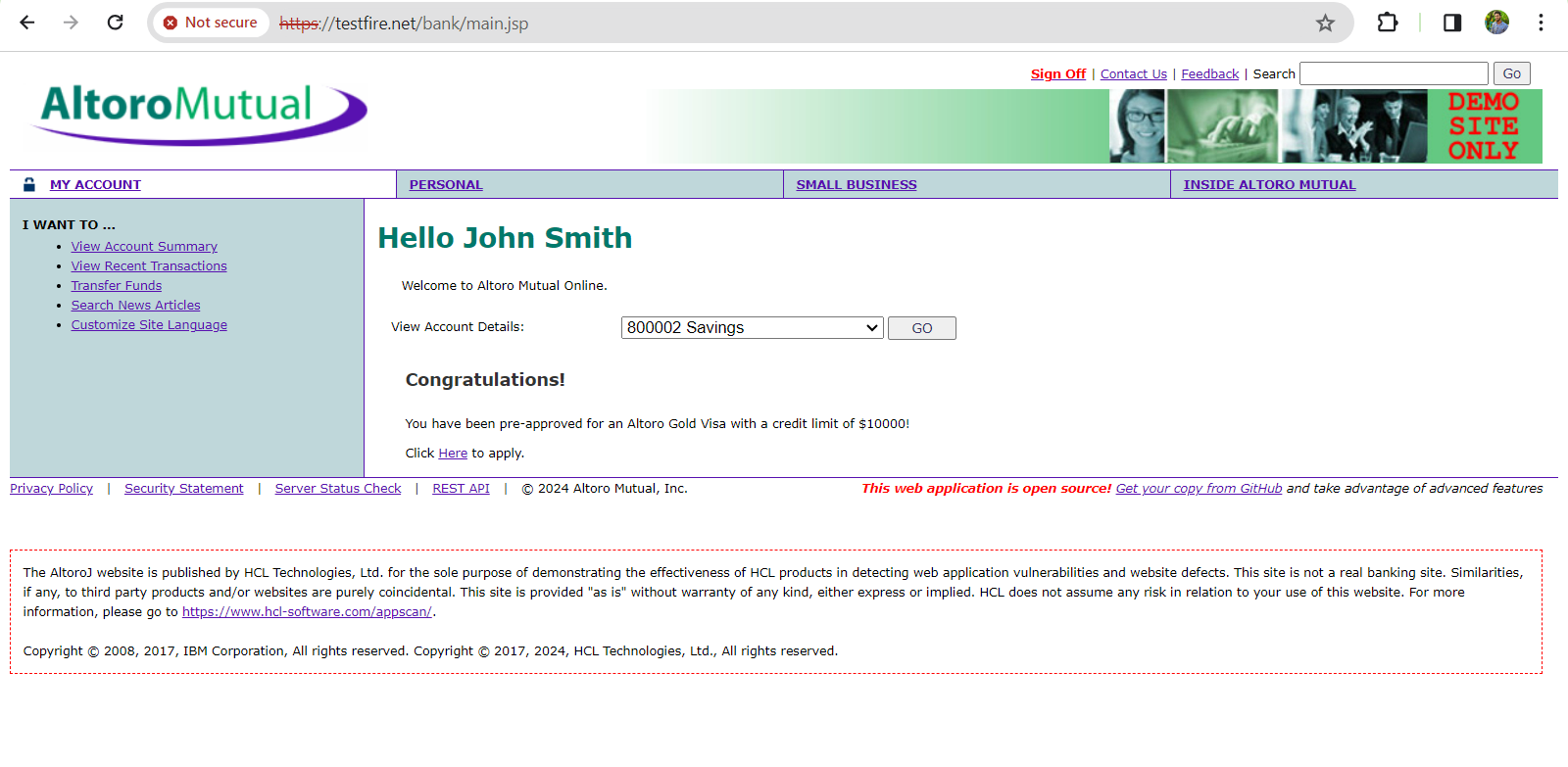
The above image indicates that we have the capability to modify user profiles and their information. Further details will be provided in the subsequent image.



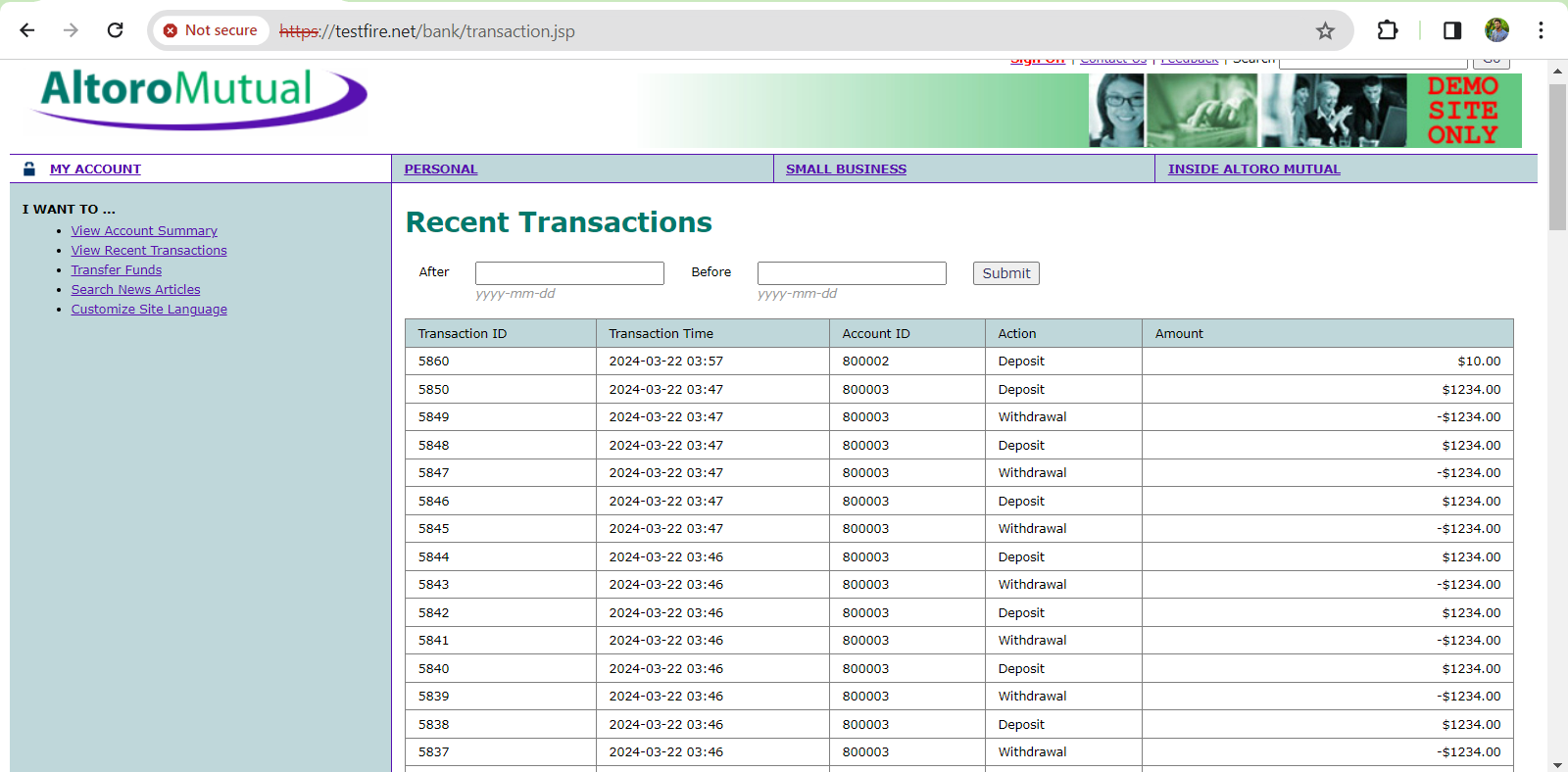
Based on the information obtained, we can alter the passwords of users and access their data. Next, we will examine the user’s authorization levels.



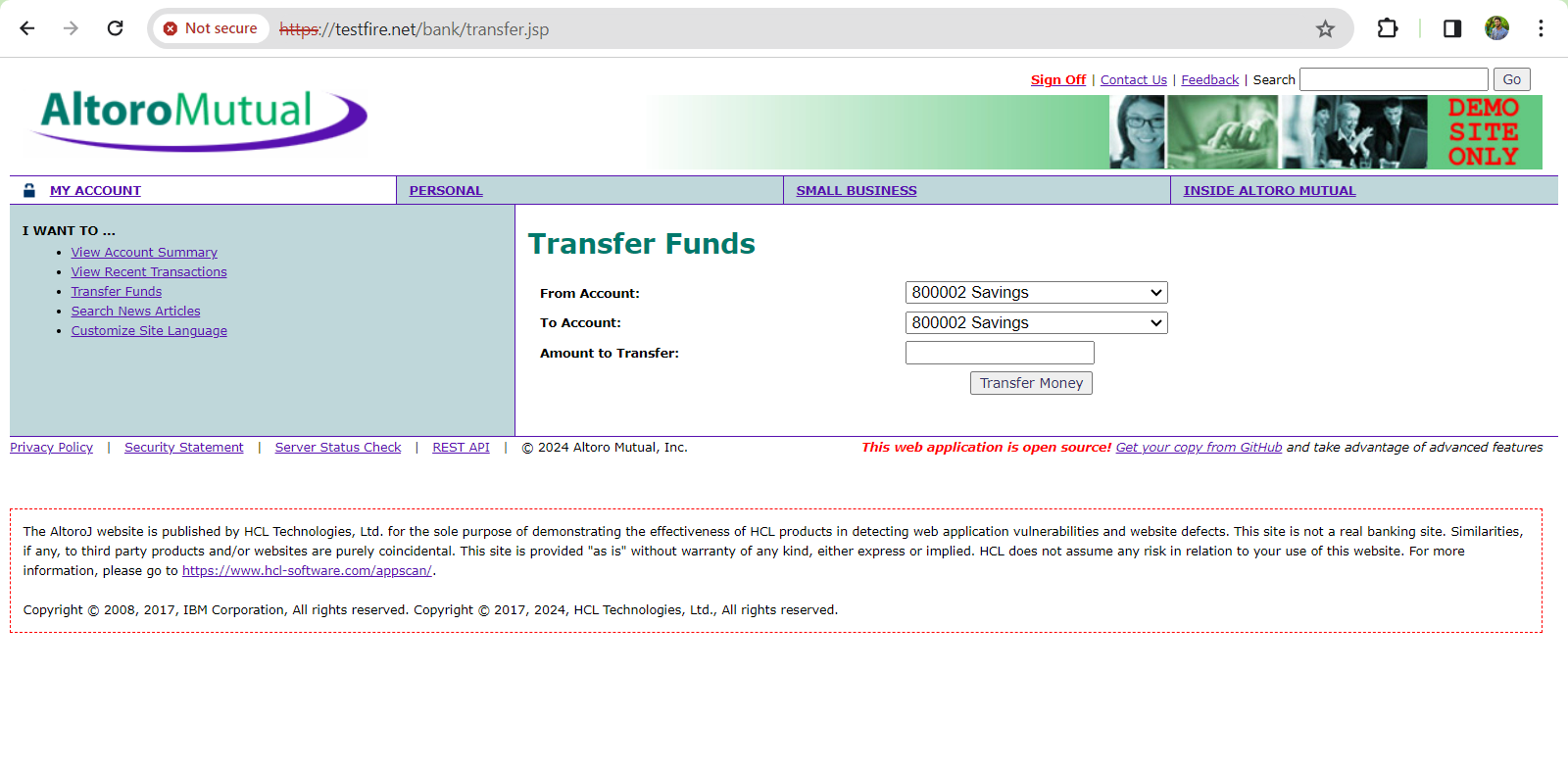
We’ve identified a user named “jsmith” but lack the password. To bypass this, we’re employing a payload that disrupts the authorization code, while attempting to use “1234” as the password. To construct the payload, we append ‘–’ to the actual username, which in this case is “jsmith”.

The image above demonstrates a successful login under the user name John Smith. This allows us to review his transaction records and execute fund transfers to other accounts, as depicted in the images below.

We have visibility into the user’s financial activities, including a detailed history of their transactions.



**Transferring funds**

****

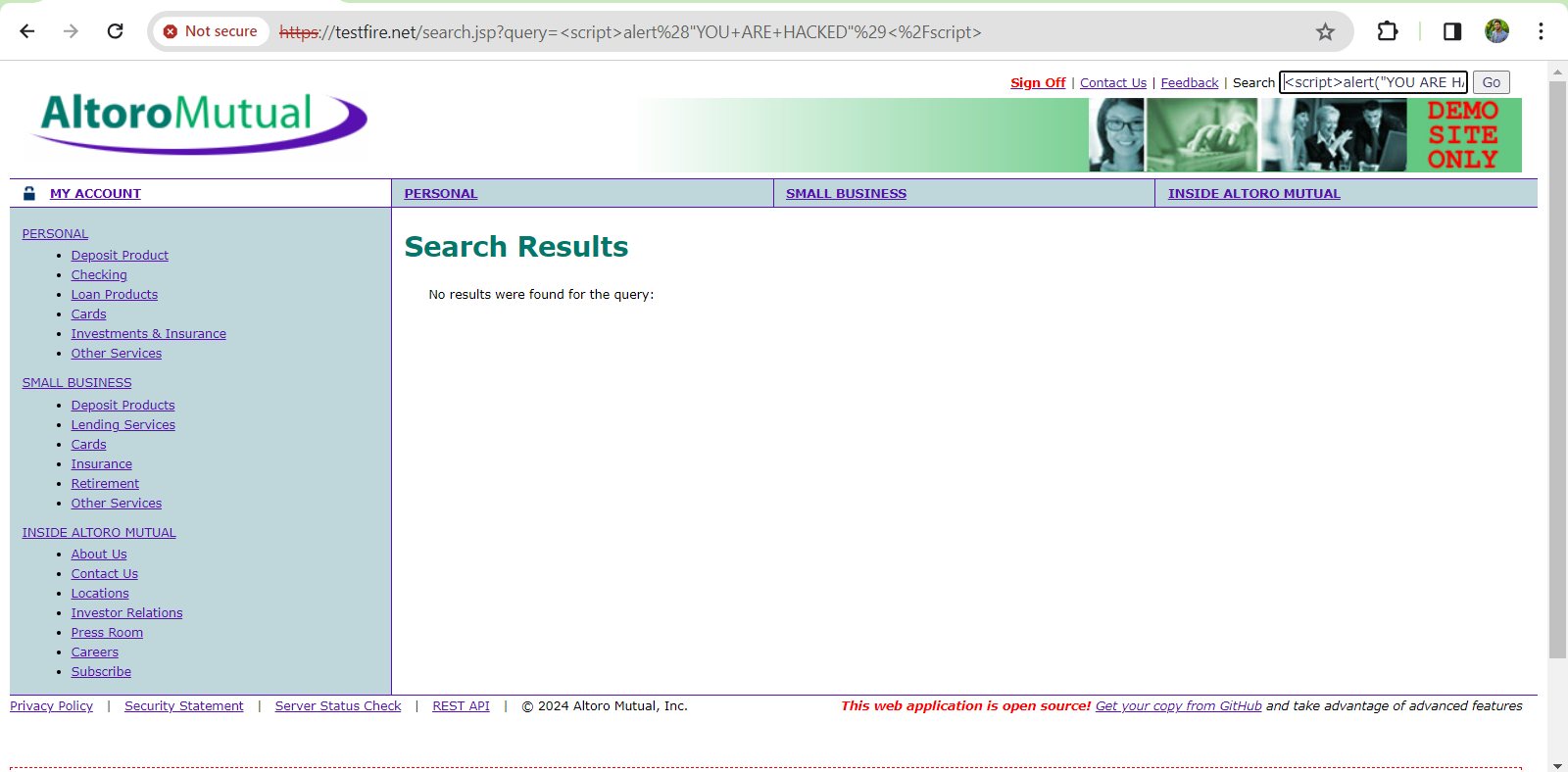
The website exhibits vulnerabilities such as broken authorization and broken access control, as observed.

**Mitigations :**

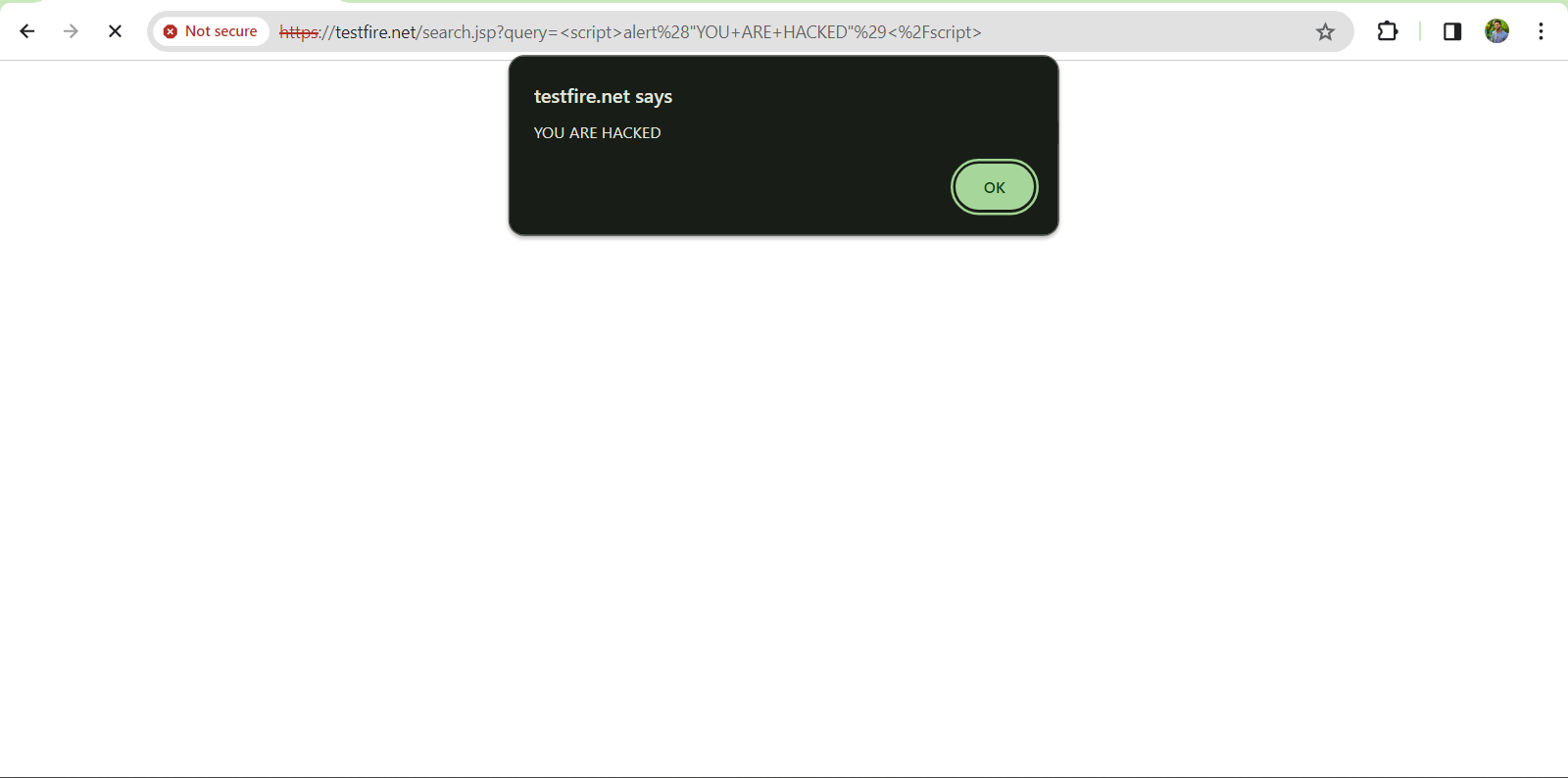
* **Establish Robust Authentication Protocols:**
  + Implement robust authentication like multi-factor authentication (MFA) to confirm system access is limited to verified users. Use advanced password protection methods, such as salted hashing, to safeguard user credentials.
* **Adopt the Principle of Least Privilege:**
  + Provide users with only the essential access needed for their specific roles. Continuously monitor and adjust permissions to adhere to the least privilege principle.
* **Apply Role-Based Access Control (RBAC):**
  + Define user roles and permissions using RBAC, ensuring detailed control over each role’s permitted actions.
* **Utilize Attribute-Based Access Control (ABAC):**
  + Base access decisions on attributes like user traits, environmental factors, or resource characteristics. Adjust access dynamically with ABAC policies as conditions change.
* **Implement Access Control Lists (ACLs) and Whitelisting:**
  + Use ACLs to detail access permissions for users or groups to certain resources. Apply whitelisting to limit actions to only those that are explicitly permitted.
* **Secure Session Management:**
  + Securely manage session tokens from creation to validation, employing HTTPS and HTTP-only cookies to shield session information from unauthorized interference.
* **Perform Regular Security Audits and Code Reviews:**
  + Regularly execute security checks, such as penetration tests and code reviews, to find and fix access control weaknesses using both automated and manual inspection methods.
* **Ensure Proper Error Handling:**
  + Prevent the disclosure of sensitive data in error messages that could assist in exploiting vulnerabilities. Use non-specific error messages for users, while capturing detailed logs for administrators.
* **Protect APIs and Web Services:**
  + Secure APIs and web services with appropriate authentication and authorization, using methods like OAuth or JWT tokens for secure API interactions.
* **Monitor and Audit Access Events:**
  + Track and log access events to spot and investigate unauthorized or suspicious activities. Regularly examine access logs and audits for potential security **breaches.**

**XSS(Cross-Site-Scripting) :**

We’re about to test the Altoro Mutual website for cross-site scripting (XSS) vulnerabilities. By injecting malicious scripts into the website’s search bar, we can determine if it’s susceptible to XSS attacks based on its response to the scripts.

****

**Resulted Output:**



* **Implement Input Validation and Sanitization:**
  + On the server side, ensure all user inputs are validated and sanitized to conform to expected formats and are free of malicious code.
* **Adopt a Content Security Policy (CSP):**
  + Use CSP to define which resources should be executed or loaded by the browser, reducing the risk of executing harmful scripts.
* **Practice Output Encoding:**
  + Before displaying user-generated content in an HTML context, encode it to prevent it from being treated as executable code by browsers.
* **Set HTTPOnly and Secure Flags on Cookies:**
  + Apply the HttpOnly flag to cookies to block access from client-side scripts, which helps protect against session token theft through XSS.
* **Utilize Frameworks with XSS Protection:**
  + Regularly update framework dependencies to take advantage of the latest security features and fixes.
* **Employ Contextual Output Escaping:**
  + Use output encoding that’s tailored to the context where the data will appear, such as HTML, JavaScript, or CSS, for targeted XSS prevention.
* **Conduct Regular Security Training:**
  + Train developers in secure coding practices and the dangers of XSS, promoting the use of secure HTML templates and avoiding the mixing of untrusted data with HTML.
* **Use Analysis Tools for Security:**
  + Employ static analysis and web vulnerability scanning tools to detect XSS flaws in code and web applications. Perform ongoing security evaluations and penetration tests to address XSS risks proactively.
* **Follow a Secure Development Lifecycle (SDLC):**
  + Integrate security-centric coding standards and best practices throughout the development process to reduce the likelihood of introducing XSS vulnerabilities.