OWASP Top 10 vulnerabilities

1. Injection: Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. This can lead to data loss, data manipulation, or unauthorized access to data.

2. Broken Authentication: Broken authentication vulnerabilities occur when authentication mechanisms are poorly implemented, allowing attackers to compromise passwords, keys, or session tokens, leading to unauthorized access.

3. Sensitive Data Exposure: Sensitive data exposure vulnerabilities arise when sensitive information, such as passwords or credit card numbers, is not adequately protected. This can lead to data breaches, identity theft, or financial fraud.

4. XML External Entities (XXE): XXE vulnerabilities occur when an application processes XML input from untrusted sources without proper validation, allowing attackers to access sensitive data or perform server-side request forgery (SSRF) attacks.

5. Broken Access Control: Broken access control vulnerabilities occur when restrictions on what authenticated users are allowed to do are not properly enforced. This can lead to unauthorized access to sensitive functionality or data.

6. Security Misconfiguration: Security misconfiguration vulnerabilities occur when security settings are not properly configured, leaving the application vulnerable to attacks such as unauthorized access, data breaches, or denial of service (DoS).

7. Cross-Site Scripting (XSS): XSS vulnerabilities occur when attackers inject malicious scripts into web pages viewed by other users. This can lead to session hijacking, data theft, or unauthorized actions on behalf of the user.

8. Insecure Deserialization: Insecure deserialization vulnerabilities occur when untrusted data is deserialized by an application without proper validation. This can lead to remote code execution, denial of service (DoS), or unauthorized access.

9. Using Components with Known Vulnerabilities: Using components with known vulnerabilities, such as libraries, frameworks, or plugins, can expose the application to attacks targeting those vulnerabilities.

10. Insufficient Logging and Monitoring: Insufficient logging and monitoring vulnerabilities occur when an application does not generate enough logs or fails to monitor those logs for suspicious activity. This can lead to delayed detection of security incidents or insufficient data for forensic analysis.

Potential Impact of These vulnerabilities on web applications:

1. Injection:

- Potential Impact: Injection vulnerabilities, such as SQL injection, can allow attackers to execute arbitrary commands or queries against the application's database. This can result in data loss, data manipulation, unauthorized access to sensitive information, or even complete compromise of the application.

2. Broken Authentication:

- Potential Impact: Broken authentication vulnerabilities can lead to unauthorized access to user accounts, allowing attackers to steal sensitive data, manipulate user settings, or perform actions on behalf of legitimate users.

3. Sensitive Data Exposure:

- Potential Impact: Sensitive data exposure vulnerabilities can result in the unauthorized disclosure of confidential information, such as passwords, credit card numbers, or personal data. This can lead to identity theft, financial fraud, or other malicious activities.

4. XML External Entities (XXE):

- Potential Impact: XXE vulnerabilities allow attackers to access sensitive files, conduct server-side request forgery (SSRF) attacks, or perform other malicious activities. This can lead to data breaches, information disclosure, or unauthorized access to internal systems.

5. Broken Access Control:

- Potential Impact: Broken access control vulnerabilities can allow attackers to bypass authorization checks and access sensitive functionality or data that they are not authorized to access. This can lead to data breaches, unauthorized transactions, or other security incidents.

6. Security Misconfiguration:

- Potential Impact: Security misconfigurations can expose sensitive information, allow unauthorized access to resources, or weaken the overall security posture of the application. This can lead to data breaches, unauthorized access, or other security incidents.

7. Cross-Site Scripting (XSS):

- Potential Impact: XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users, leading to session hijacking, data theft, or unauthorized actions on behalf of the user. This can result in compromised user accounts, data breaches, or other security incidents.

8. Insecure Deserialization:

- Potential Impact: Insecure deserialization vulnerabilities can lead to remote code execution, denial of service (DoS) attacks, or unauthorized access to sensitive data. This can result in compromised systems, data breaches, or other security incidents.

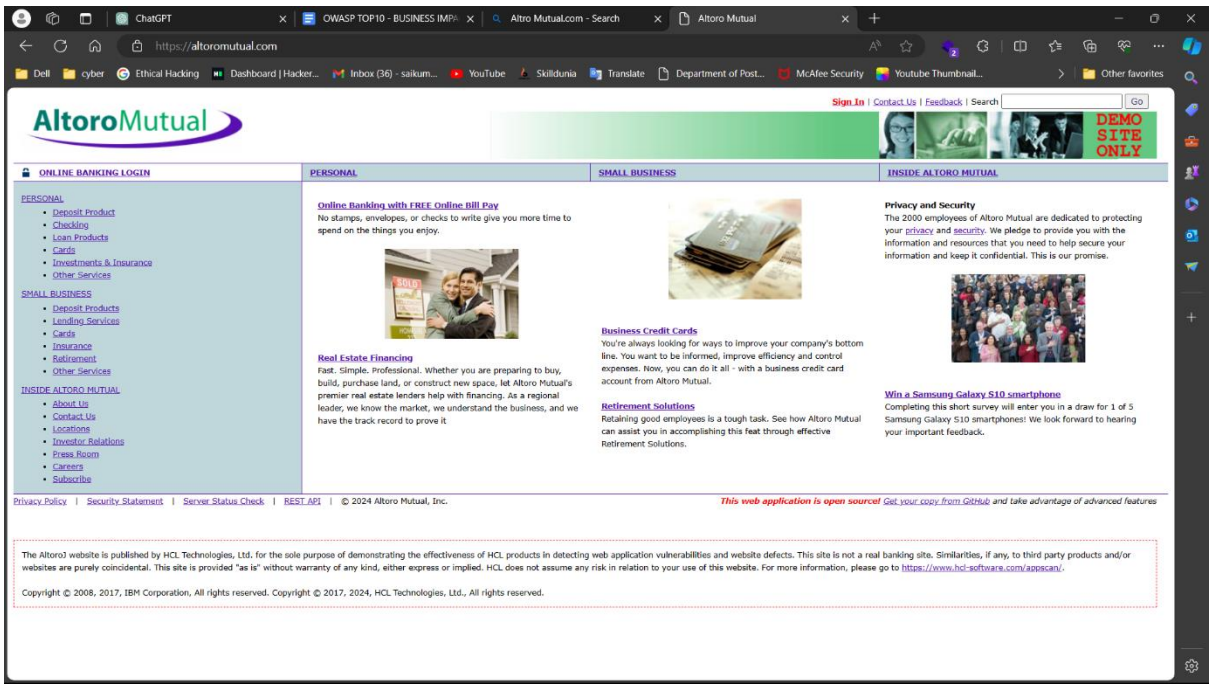
9. Using Components with Known Vulnerabilities:

- Potential Impact: Using components with known vulnerabilities can expose the application to attacks targeting those vulnerabilities, leading to data breaches, unauthorized access, or other security incidents.

10. Insufficient Logging and Monitoring:

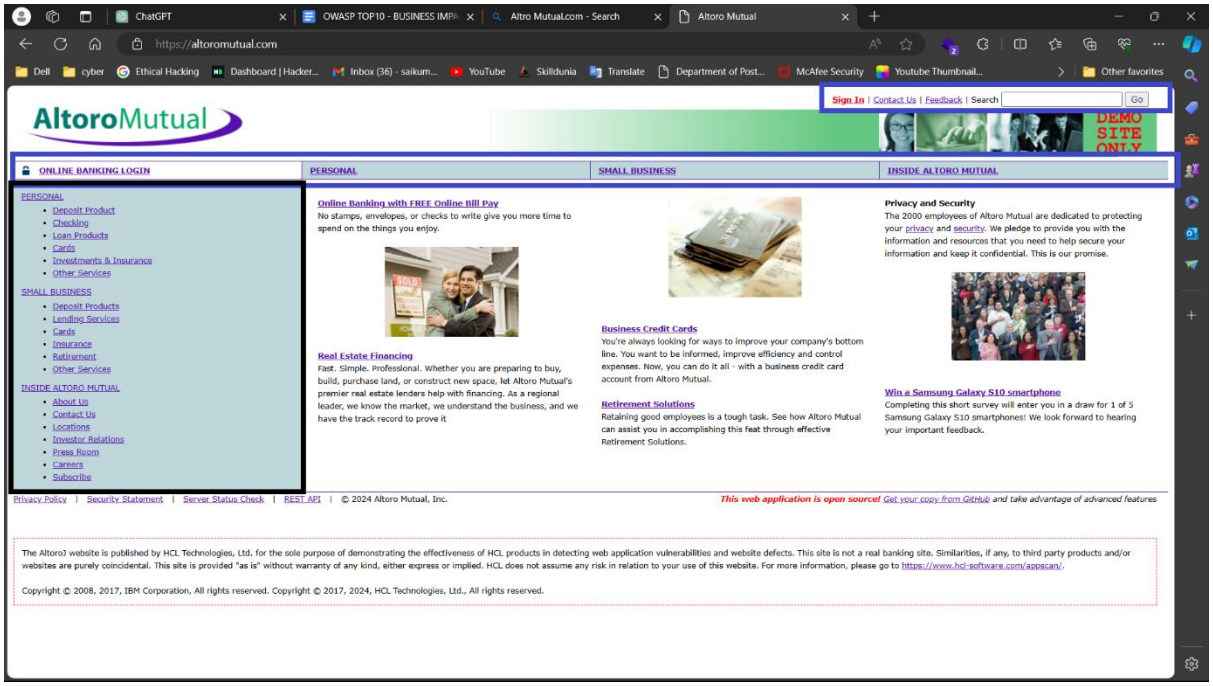
- Potential Impact: Insufficient logging and monitoring can result in delayed detection of security incidents, inadequate response to attacks, or insufficient data for forensic analysis. This can lead to prolonged security breaches, data loss, or other negative consequences.

Altoro Mutual Website Analysis Report



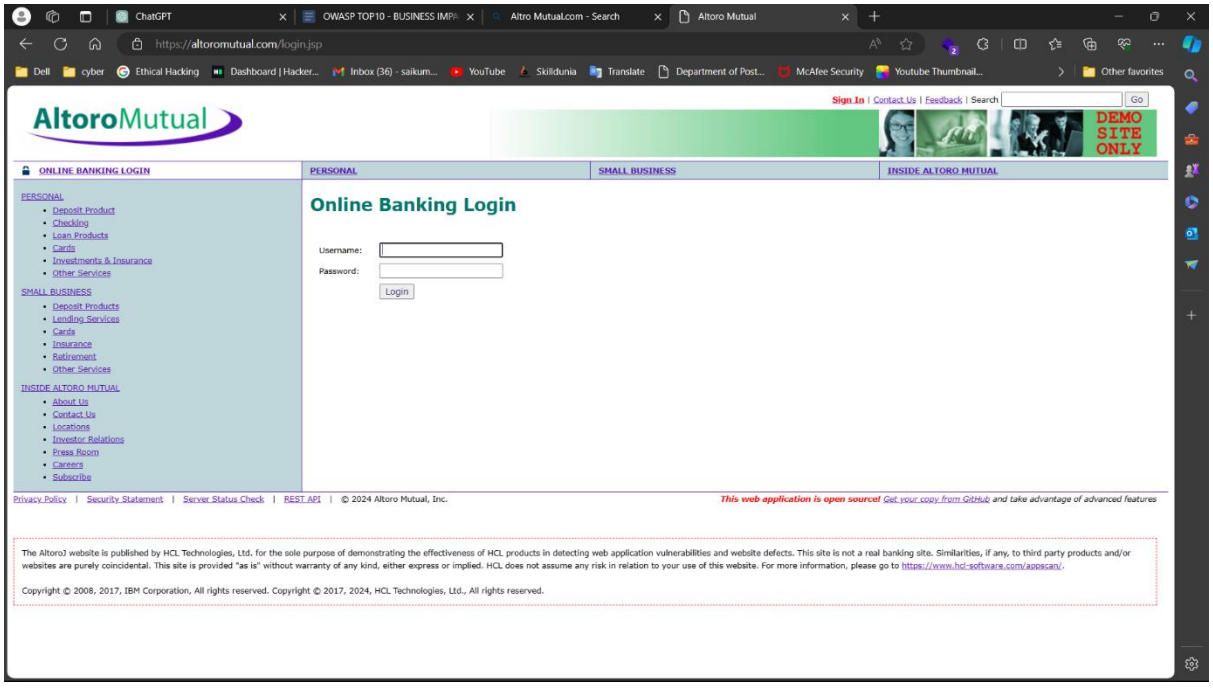
It is Altoro Mutual Website's home page. It has "sign in" option, "contact" option, "feedback" option. Search is given to search for components in the website, here the search option should be removed and add after login. There might be a chance of "Cross-site scripting" attack.

As shown in the below image, there are some categories like personal, small business and inside alotoro mutual. For every category there are sub-divisions mentioned in the left side column.

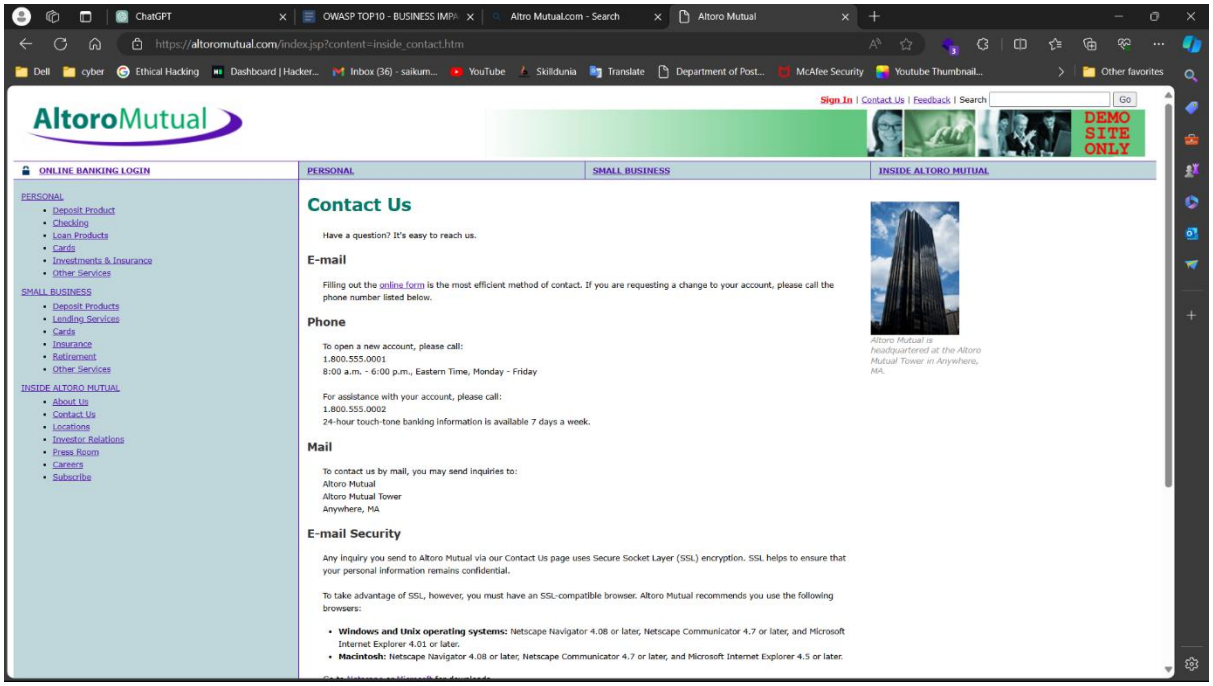


At the end of the column, we can see some options which are actually links for “privacy policy”, “Security Statement”, “Server Status Check” and “Rest API”.

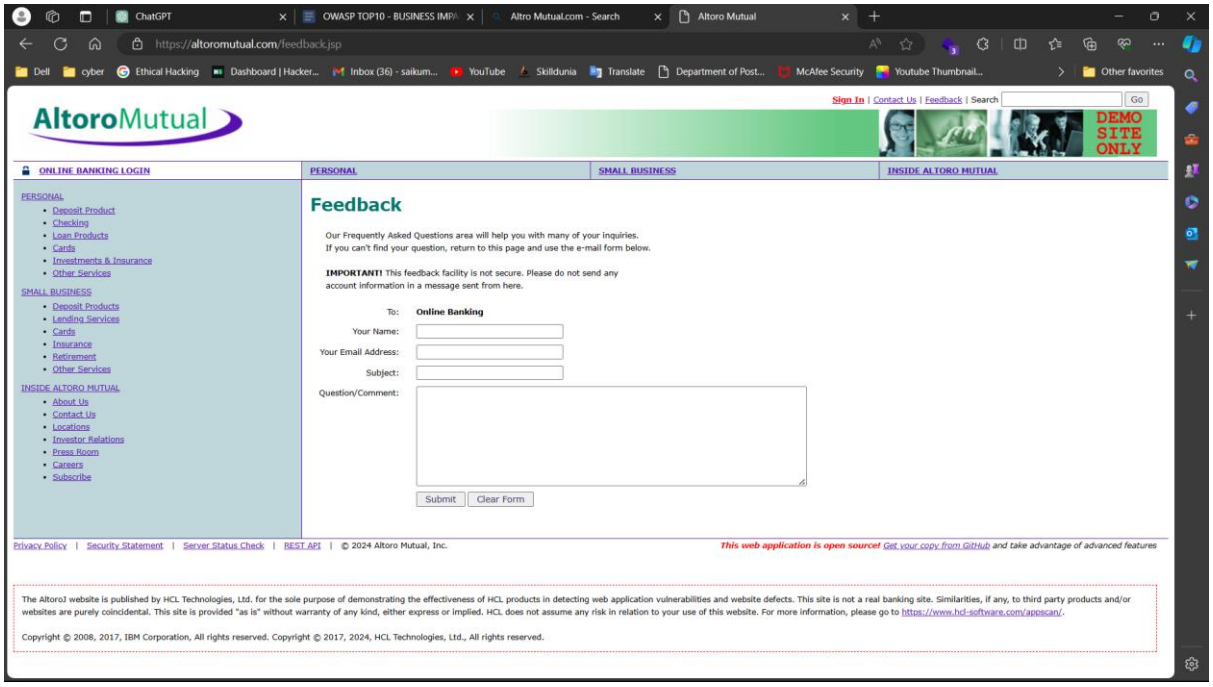
It is the login page.



It is the Contact Info page.

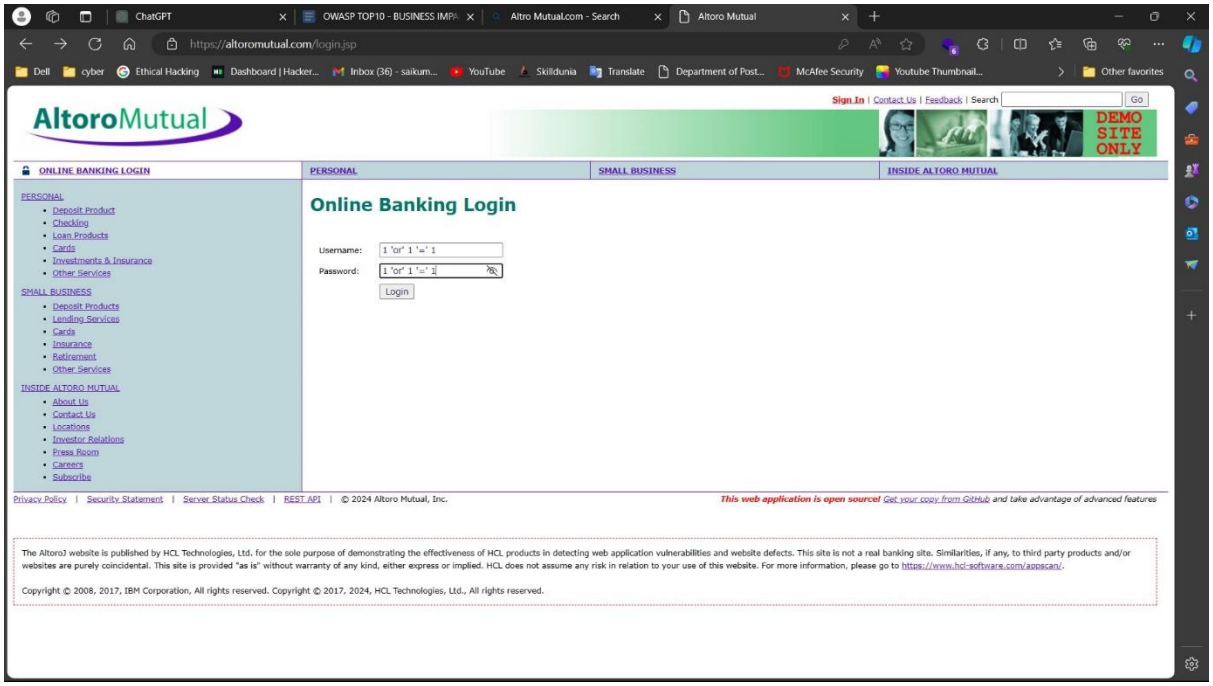


Feedback form

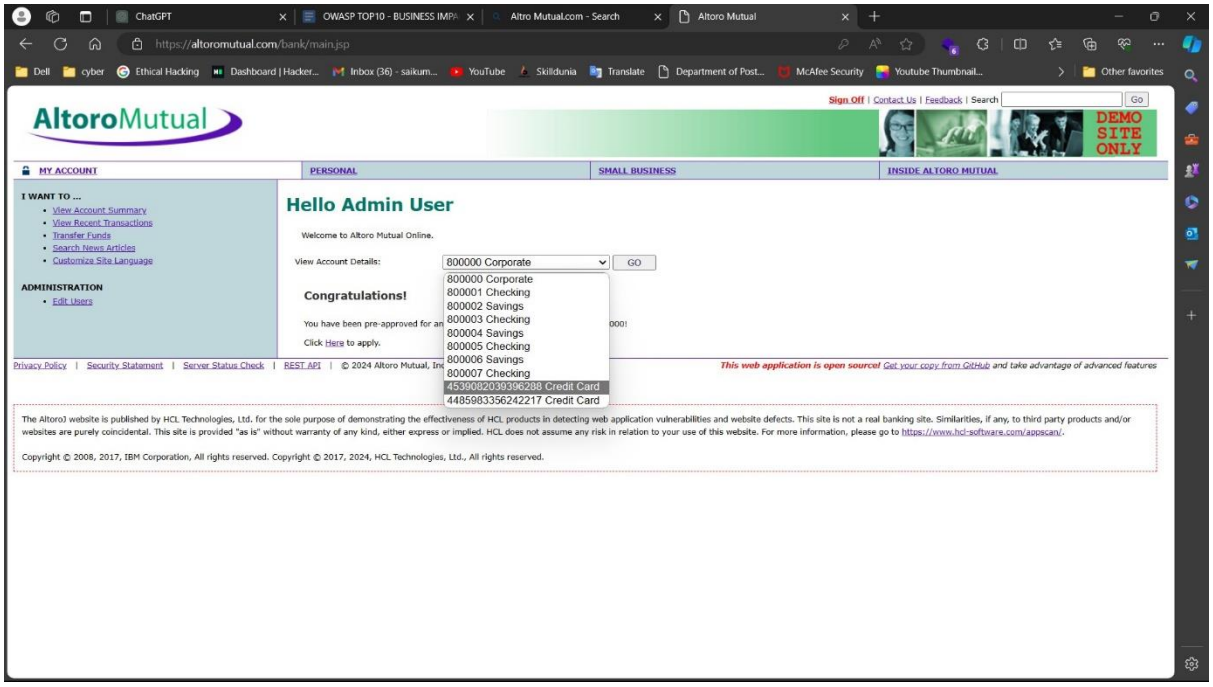


Checking for the vulnerabilities.

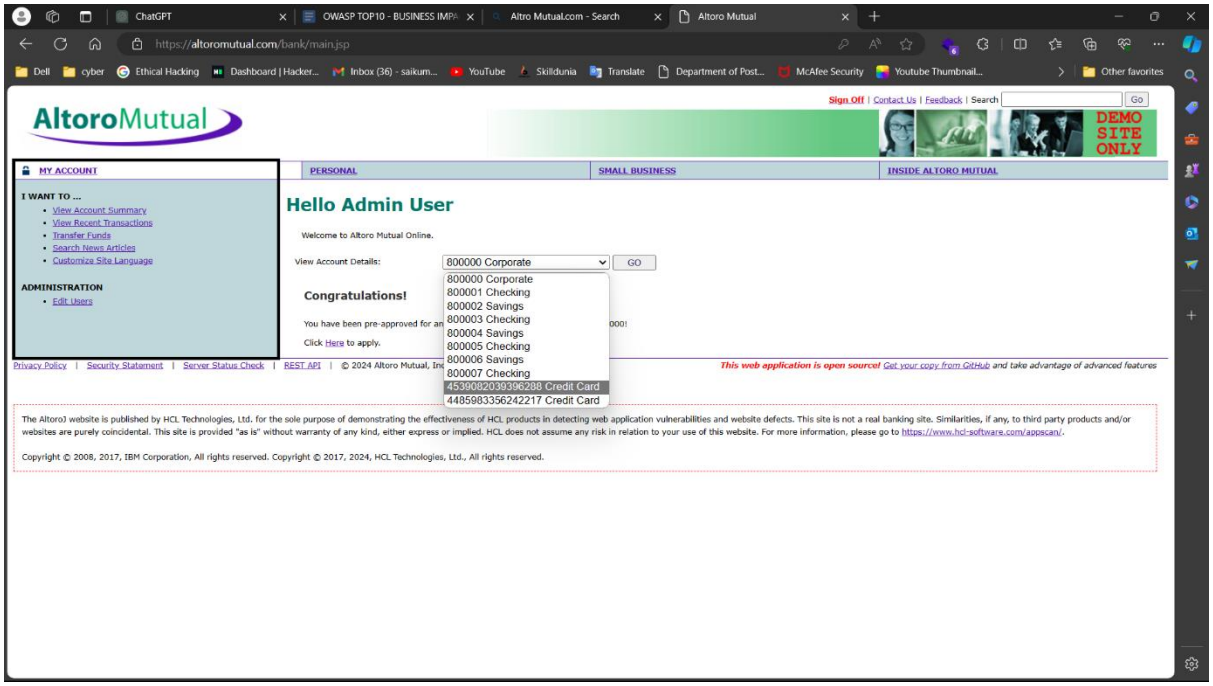
Here, we are going to check whether we can login into website, if we can login then the vulnerability is called “Broken Authorization”, it can lead to other vulnerabilities like Sensitive Data Exposure, Broken Access Control.



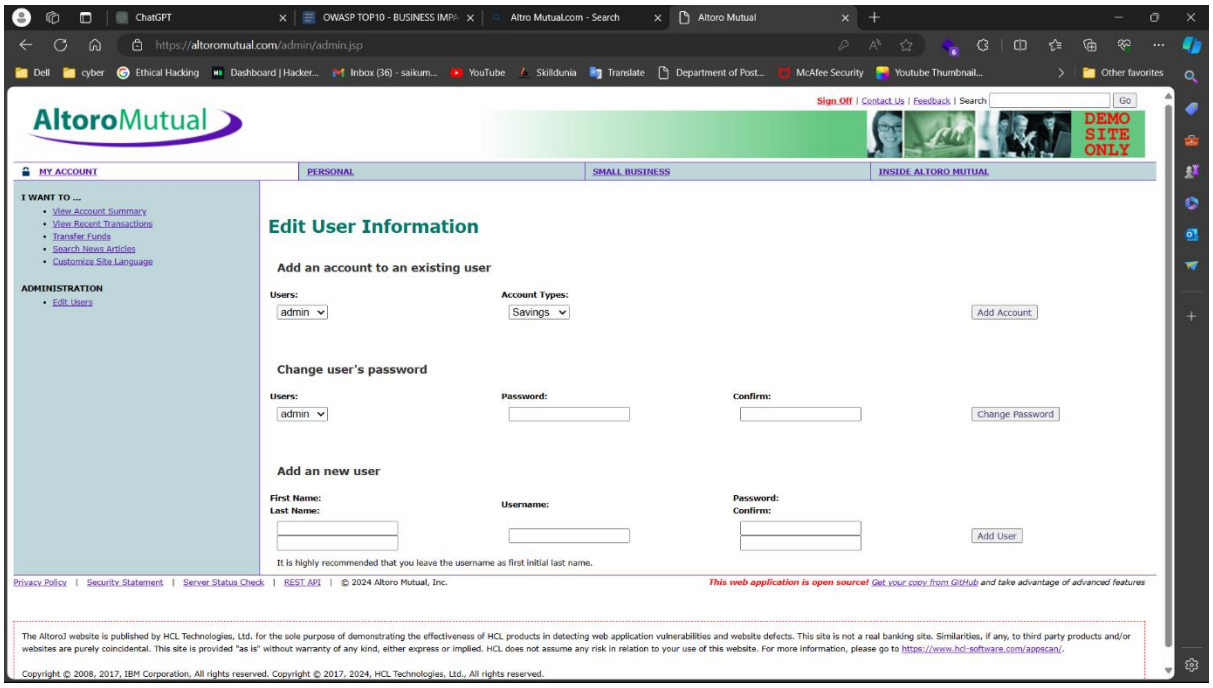
Here we used a payload 1 ‘or’ 1 ‘=’ 1 as username and password and logging as admin. We can see other users. We can add users, change their passwords and can control their account.



Here we can see account details of users of the bank. In the below image, highlighted column displays the content pages of fund transfer, account summary, recent transactions.

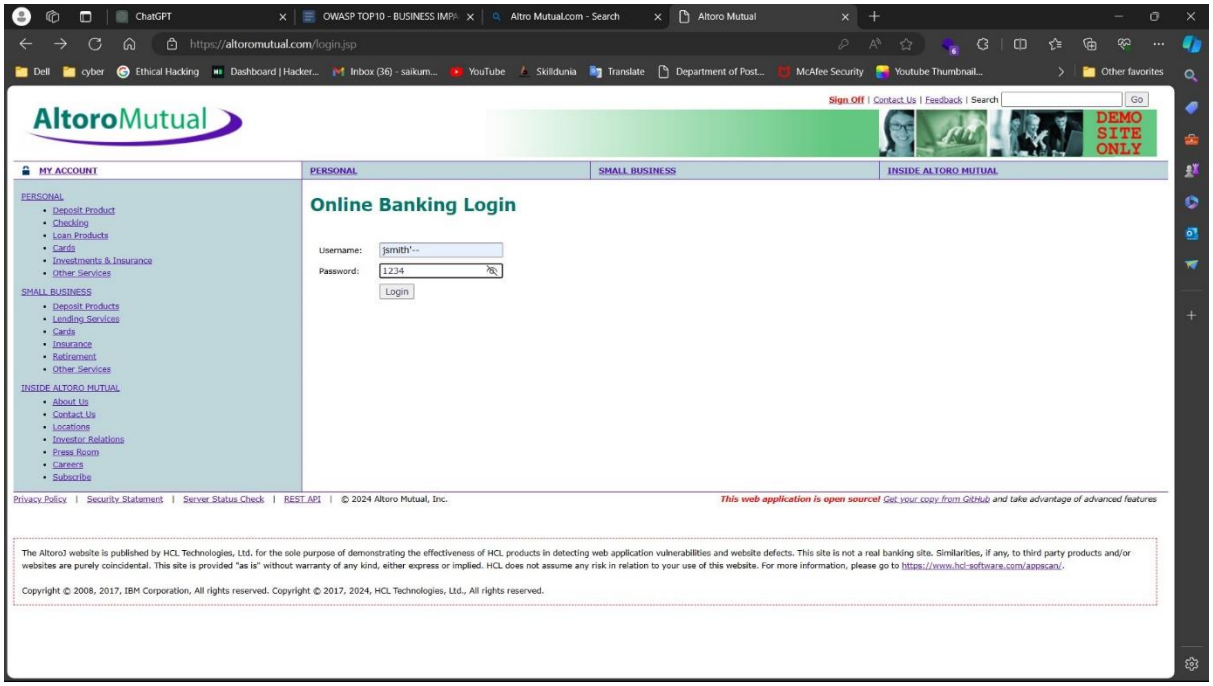


As shown in the above image, we can edit users and their details. Those will be explained in below image.



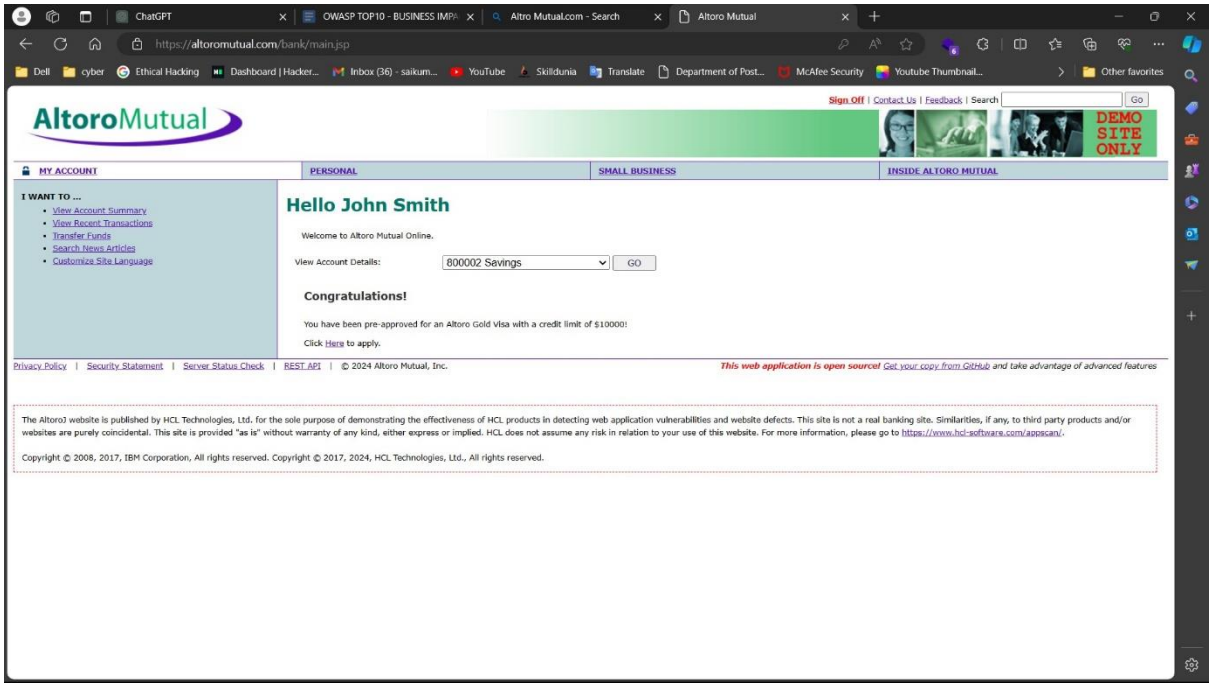
From above, we got users. We can change users password and access their data.

Now we are going to check the user's authorization.



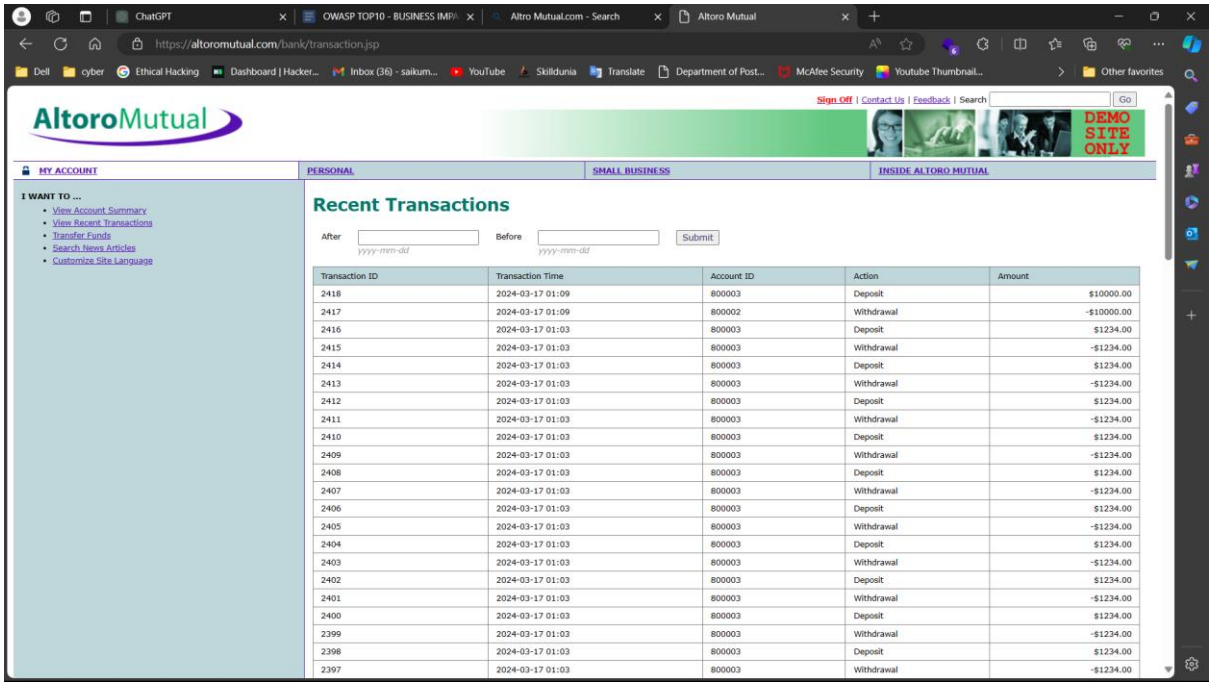
One of the users we found is “jsmith”. But we don’t know the password, so we are using a payload to confuse the authorization code and using password as “1234”.

For payload we attach ‘-- to the actual username which is “jsmith”.

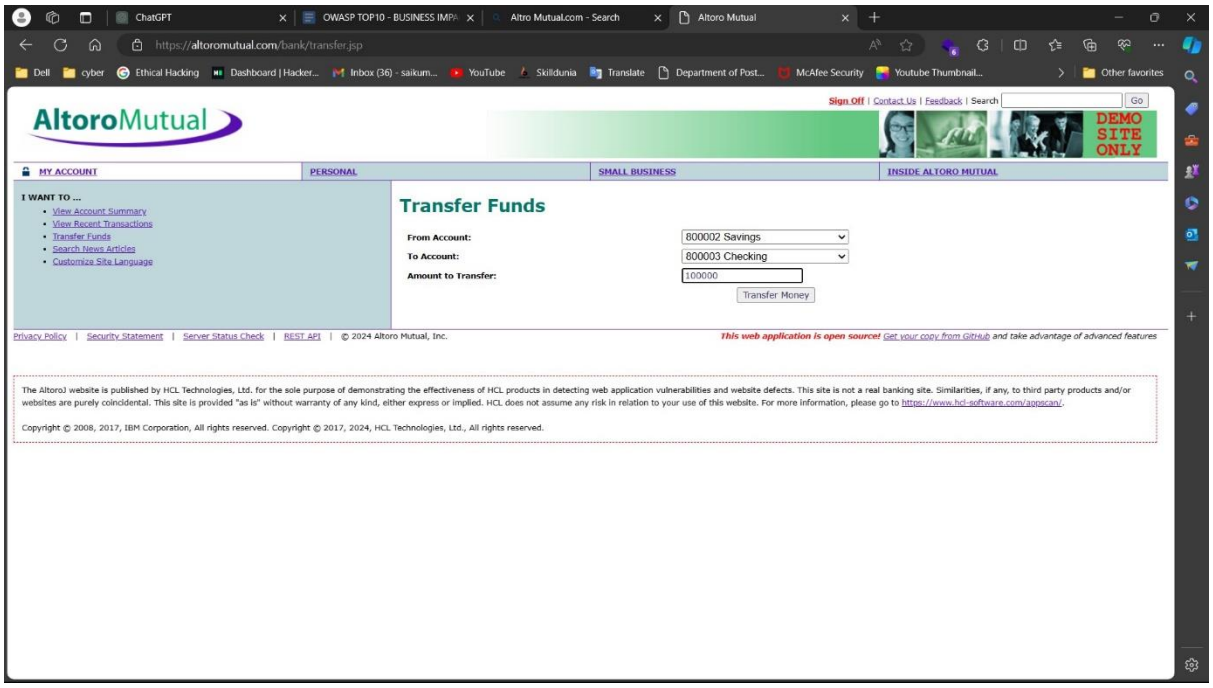


As you can in the above image, we have successfully logged in as John Smith. Now we can view his transaction history, also can transfer the funds to others as shown in the below images.

we can see user transactions and history of transaction of the user.



Transferring funds



As we can see, this website has broken authorization vulnerability, Broken access control.

Mitigations :

- Implement Proper Authentication Mechanisms:
    - Use strong authentication methods such as multi-factor authentication (MFA) to ensure that only authorized users can access the system. Employ secure

password storage techniques like hashing with salts to protect user credentials from being compromised.

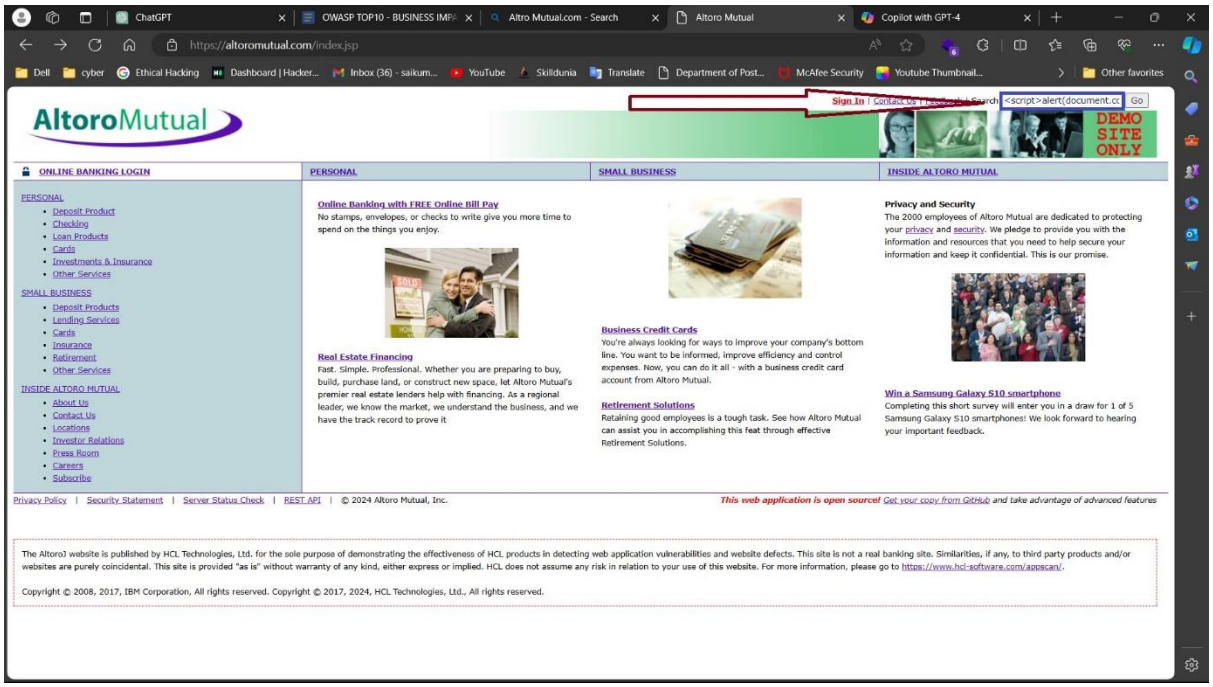
- Enforce Least Privilege Principle:
  - Assign the minimum level of access required for each user or role within the system. Users should only have access to the resources and functionalities necessary for their roles. Regularly review and update permissions to ensure they align with the principle of least privilege.
- Implement Role-Based Access Control (RBAC):
  - Use RBAC to define and enforce user roles and permissions within the system. Ensure that access controls are granular, allowing for fine-grained control over what actions each role can perform.
- Implement Attribute-Based Access Control (ABAC):
  - ABAC allows access control decisions to be based on various attributes such as user characteristics, environmental conditions, or resource properties. Implement ABAC policies to dynamically adjust access control decisions based on changing conditions.
- Use Access Control Lists (ACLs) and Whitelisting:
  - Employ ACLs to specify which users or groups have access to specific resources. Utilize whitelisting to explicitly define allowed actions or entities, thereby restricting access to only approved entities or actions.
- Implement Secure Session Management:
  - Ensure that session tokens are securely generated, transmitted, and validated. Use secure mechanisms such as HTTPS and HTTP-only cookies to protect session data from unauthorized access or tampering.
- Regular Security Testing and Code Reviews:
  - Conduct regular security assessments, including penetration testing and code reviews, to identify and address potential access control vulnerabilities.' Use automated tools and manual techniques to uncover security flaws in access control mechanisms.
- Implement Proper Error Handling:
  - Avoid revealing sensitive information in error messages that could aid attackers in exploiting access control vulnerabilities. Provide generic error messages to users while logging detailed error information for system administrators.
- Secure APIs and Web Services:
  - Implement proper authentication and authorization mechanisms for APIs and web services. Use techniques such as OAuth or JWT tokens to securely authenticate and authorize API requests.
- Monitor and Audit Access Control Events:
  - Implement logging and monitoring mechanisms to track access control events and detect suspicious or unauthorized activities. Regularly review access logs and audit trails to identify and investigate potential security incidents.

Cross – Site – Scripting:

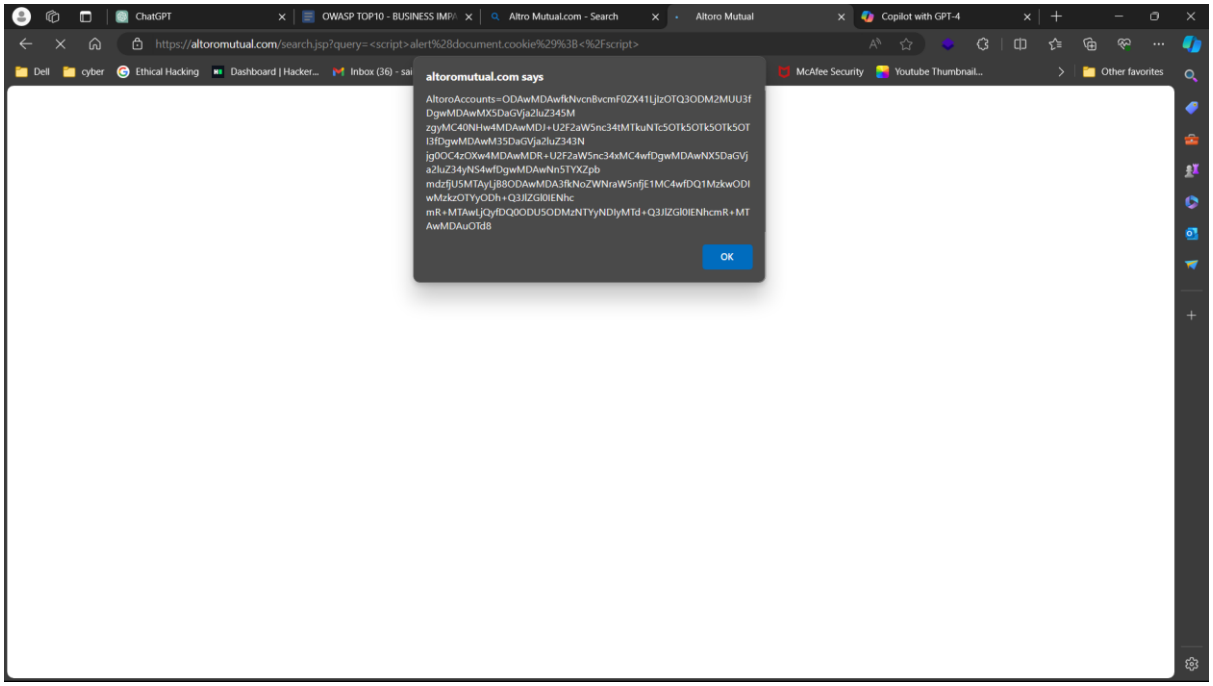
Now we are going to perform cross site scripting on the Altoro Mutual website.

Here we use malicious scripts to bypass the security website. If the website reacts to those scripts, then we can say that this website vulnerable to “cross – site – scripting”.

In the below image, we can see that the website contains search bar. We can use scripts there.



Resulted Output:



As we can see, it gave information about cookies which are stored in the website.

Mitigations :

- Input Validation and Sanitization:
  - Validate and sanitize all user input on the server-side to ensure that it meets expected formats and does not contain malicious code
- Content Security Policy (CSP):
  - Implement a Content Security Policy (CSP) to specify which resources the browser should execute or load, thereby mitigating the execution of injected scripts.
- Output Encoding:
  - Encode user-generated content before rendering it in HTML context to prevent browsers from interpreting it as executable code.
- HTTPOnly and Secure Cookies:
  - Set the `HttpOnly` flag on cookies to prevent client-side scripts from accessing them, thereby mitigating the risk of stealing session tokens via XSS attacks.
- Use Frameworks with Built-in XSS Protection:
  - Keep framework dependencies up-to-date to ensure that you benefit from the latest security enhancements and patches.
- Contextual Output Escaping:
  - Apply output encoding specific to the context in which the data will be rendered (e.g., HTML, JavaScript, CSS) to provide precise protection against XSS attacks.
- Regular Security Training and Awareness:
  - Educate developers about secure coding practices and the risks associated with XSS vulnerabilities. Encourage developers to use safe HTML templating mechanisms and avoid concatenating untrusted data with HTML markup.
- Static and Dynamic Analysis Tools:
  - Use static code analysis tools and web vulnerability scanners to identify XSS vulnerabilities in code and web applications. Conduct regular security assessments and penetration tests to identify and remediate XSS vulnerabilities before they can be exploited by attackers.
- Secure Development Lifecycle (SDLC):
  - Implement security-focused coding guidelines and best practices to minimize the introduction of XSS vulnerabilities during development.