Common Web Application Vulnerabilities

1. SQL Injection :

\*) **SQL Injection (SQLi)** is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.

\*) SQL Injection allowing attackers to manipulate the SQL queries and extract sensitive data, modify database records, or even gain unauthorized access.

Steps :

1. Open the website URL : <http://testfire.net/bank/main.jsp>
2. Navigate to the Online banking login on the top left corner.
3. In username and password input field try to enter the wrong username and password to verify whether the validation is working properly or not.
4. Now, try to perform SQL Injection by entering ‘OR 1=1’ in the username and password input field because the condition 1=1 is always became try

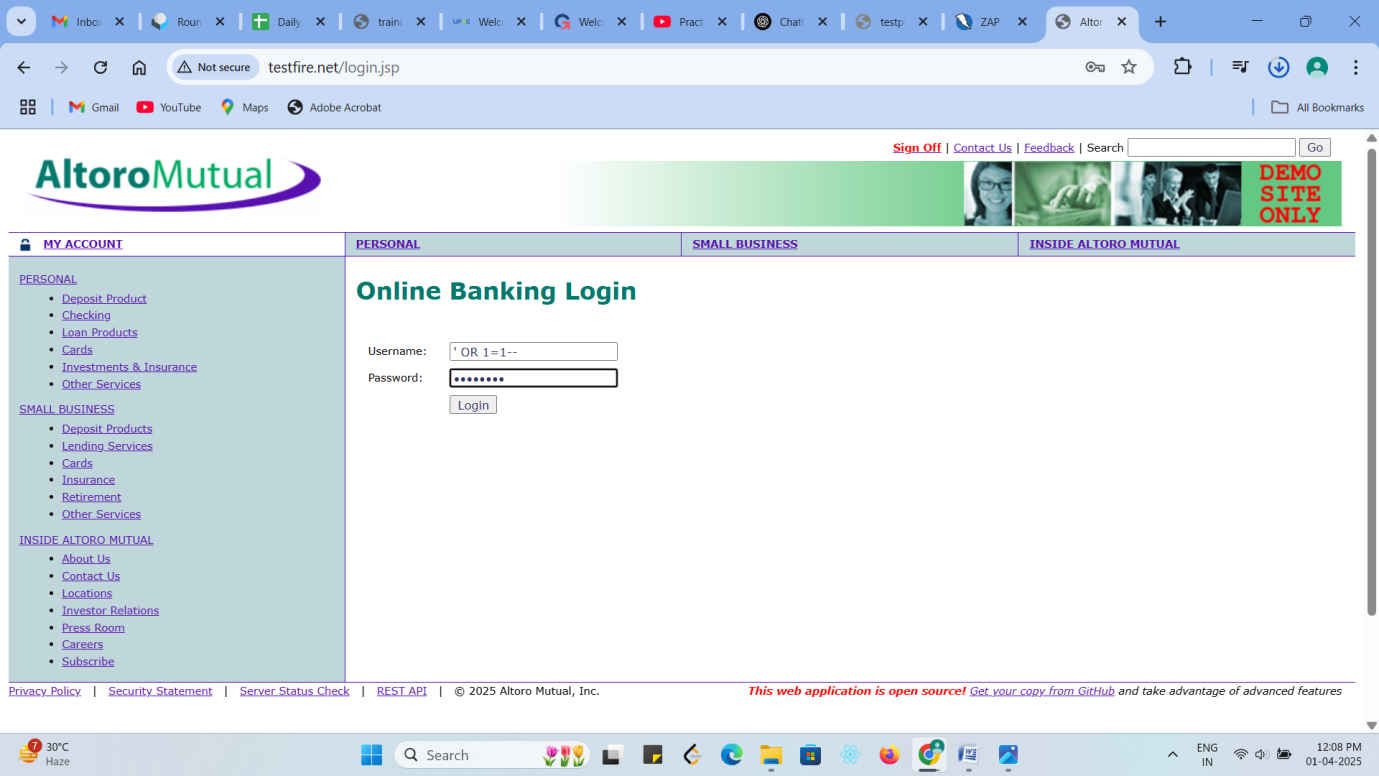
Bypass the user validation.

If the website responds with unexpected results (such as logging in without credentials), it may be **vulnerable to SQL Injection**.

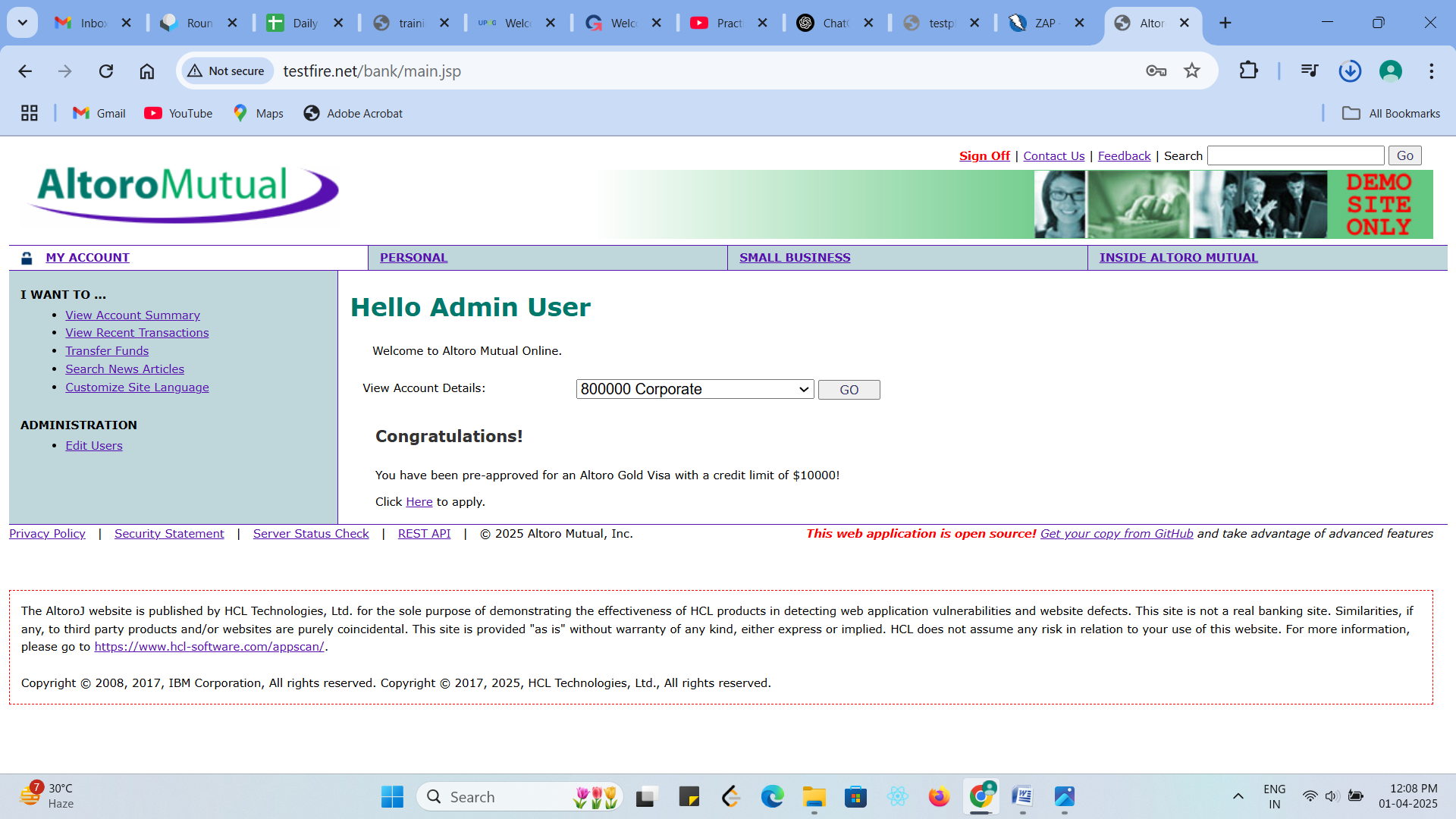
* The SQL query behind the login page might look like this:

SELECT \* FROM users WHERE username='' OR '1'='1' -- ' AND password='';

* Since '1'='1' is **always true**, the query returns all users, allowing access.



In the login page entered the SQL query “OR 1=1” to manipulate the database query, so that every vulnerable user can login to the webpage.



By passing the user authentication using SQL Injection the user can login to the webpage without the valid user credentials.

1. Cross-Site Scripting (XSS):

\*) Cross –Site scripting is a process that attacker inject malicious script into web page, which execute in user’s browser.

\*) When a victim loads a page, the injected script execute in their browser, leading to data theft, Session hijacking or website defacement.

Types of XSS :

1. Stored XSS : In stored XSS the malicious script is permenantly stored on the server.
2. Reflected XSS : In reflected XSS the malicious script is included in a URL and reflected back to the user.
3. Dom-Based XSS : In Dom-based XSS the vulnerability exist-in client-side javascript and modifies the webpage dynamically.

Steps :

1) Open the website URL : <http://testfire.net/login.jsp>

2) On the top right corner on the search input box.

3) Try to inject the malicious script to perform reflected cross site scrpitting.

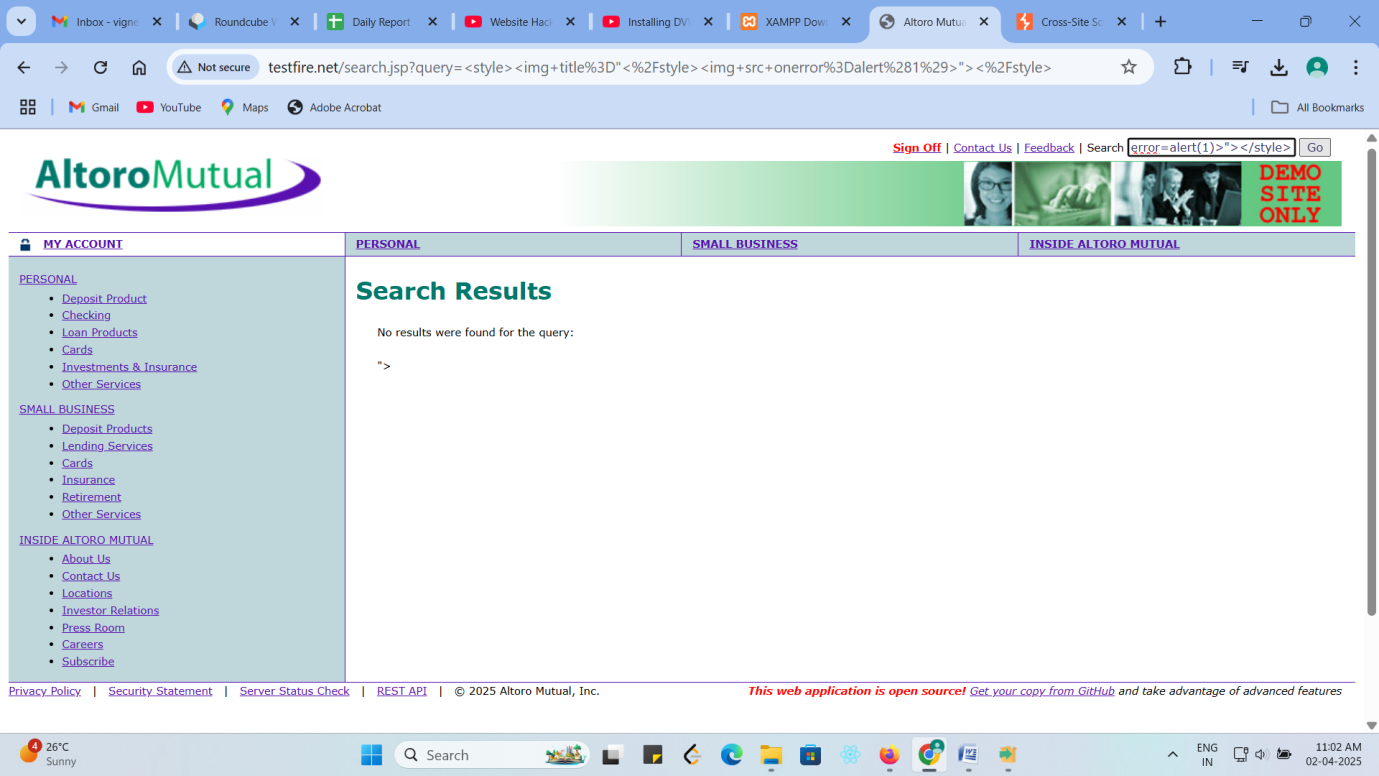
4) After entered the malicious javascript code to the search input box press the go button near to the input box to inject the code to the webpage.

If the website responds with unexpected results (such as the alert window popup), it may be **vulnerable to Cross-site Scripting (XSS).**

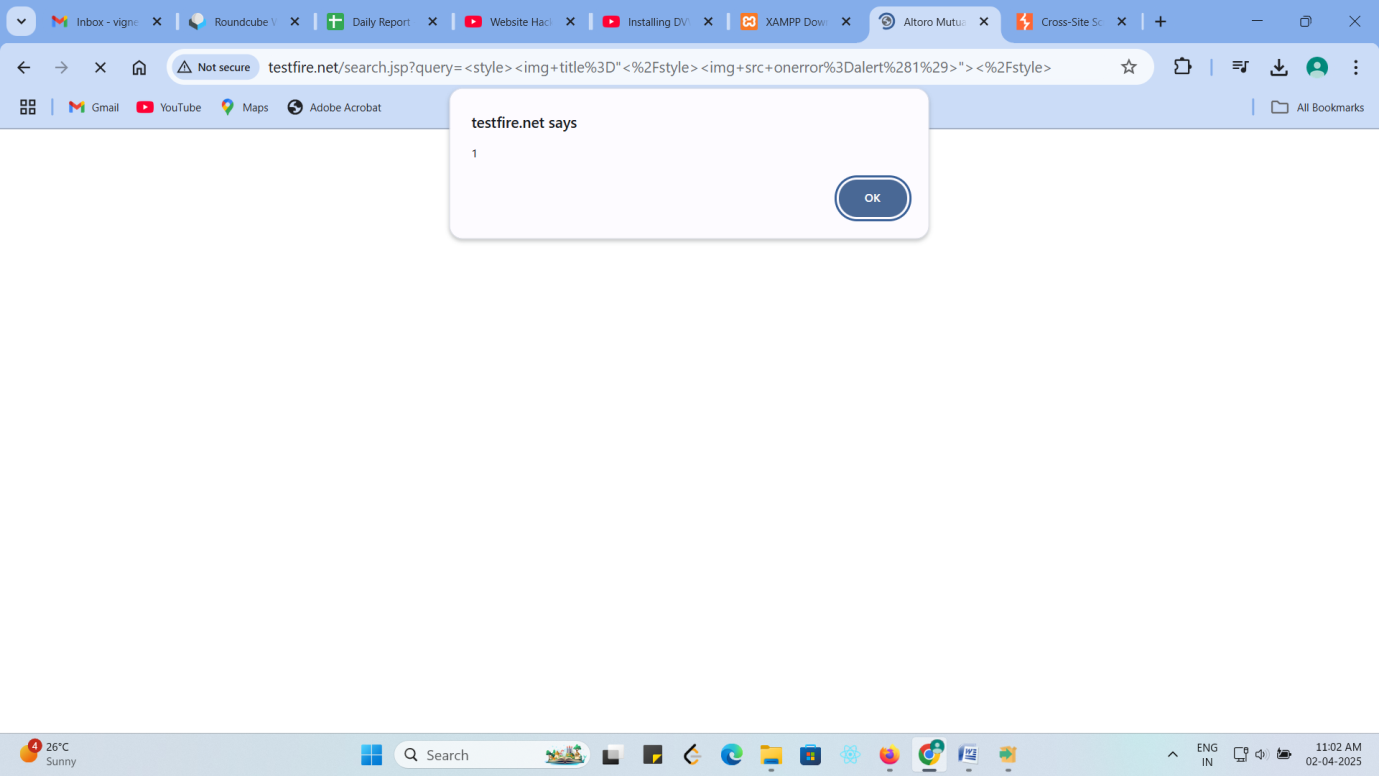
* The javaScript code injected to the webpage is :

[<script><img title="</script><img src onerror=alert(1)>"></script>](https://portswigger-labs.net/xss/xss.php?x=%3Cscript%3E%3Cimg%20title%3D%22%3C%2Fscript%3E%3Cimg%20src%20onerror%3Dalert%281%29%3E%22%3E%3C%2Fscript%3E&context=html)

* When the above javascript code is injected to the webpage the alert message with value “1” is displayed in the browser window, so the webpage is vulnerable to Cross-site scripting.



The malicious javascript code in injected to the search input box in the webpage to perform Reflected cross site scripting.



When the malicious script is executed the alert window with value “1” is displayed in the browser window, so the website is vulnerable to Cross-site scripting(XSS).

3)Cross Site Request Forgery (CSRF):

\*) Cross-site request forgery forces an authenticated user to execute unwanted action on the a trusted without their consent.

\*) CSRF attack exploit the trust a web application has in a user’s browser by sending unauthorized command from the victim authenticated session.

\*) By using CSRF token we can able to prevent the web application from cross site request forgery.

Steps :

1) Open the website URL : <http://testfire.net/login.jsp>

2) Click on the Online banking Login on the top left corner of the webpage.

3) Enter the valid login credentials and authenticate the login page and keep the user active on the same webpage.

4) After login successfully click on Transfer funds which is third in the list.

5) Keep the page remain without any changes as a active user.

6) Make a clone a particular webpage and run the webpage in separate tab on the same browser.

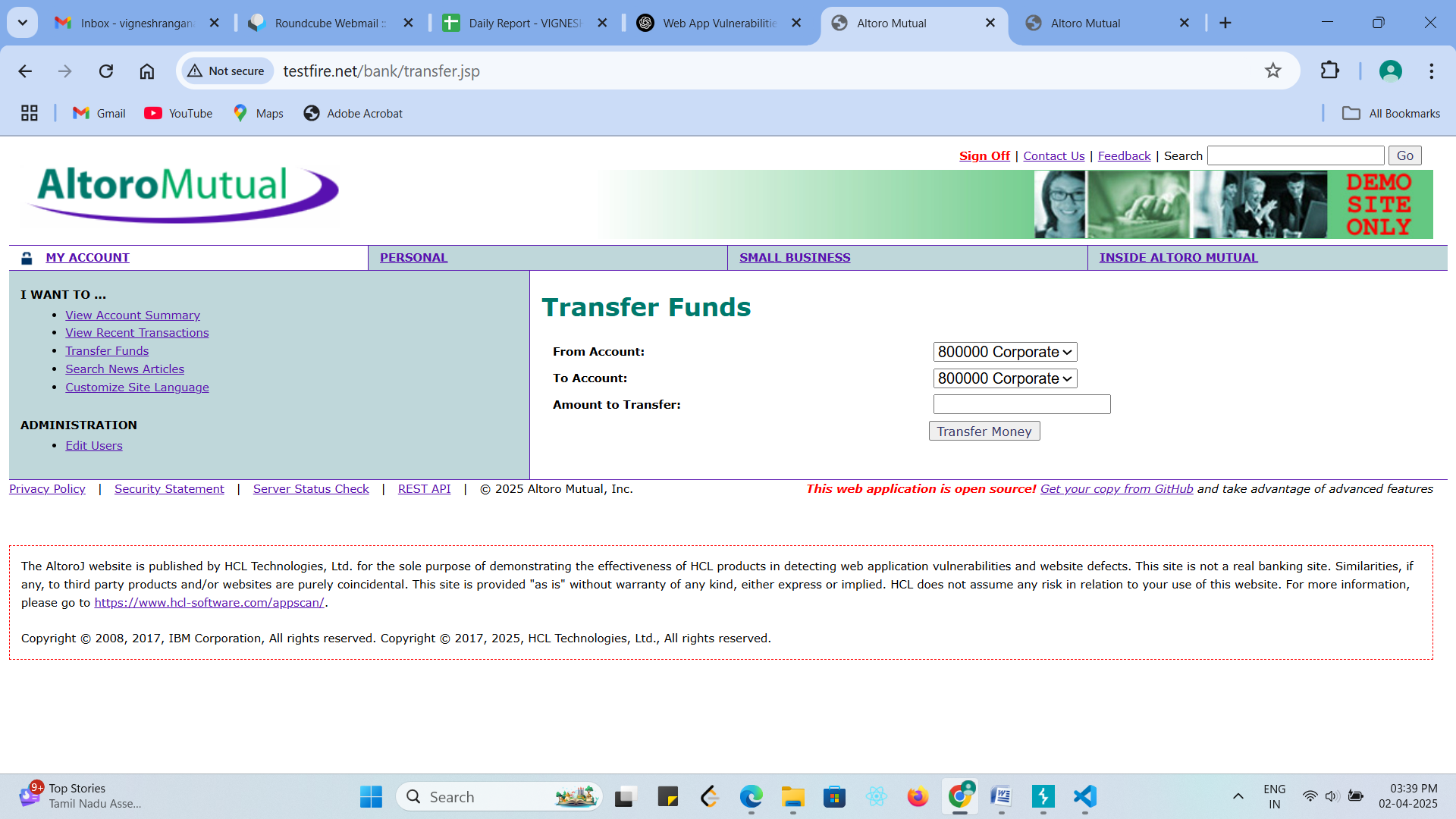
7) Choose the from account and to account and enter some valid amount in the amount transfer input box and click on transfer money.

8) Open the original tab where the user in active and review the changes.

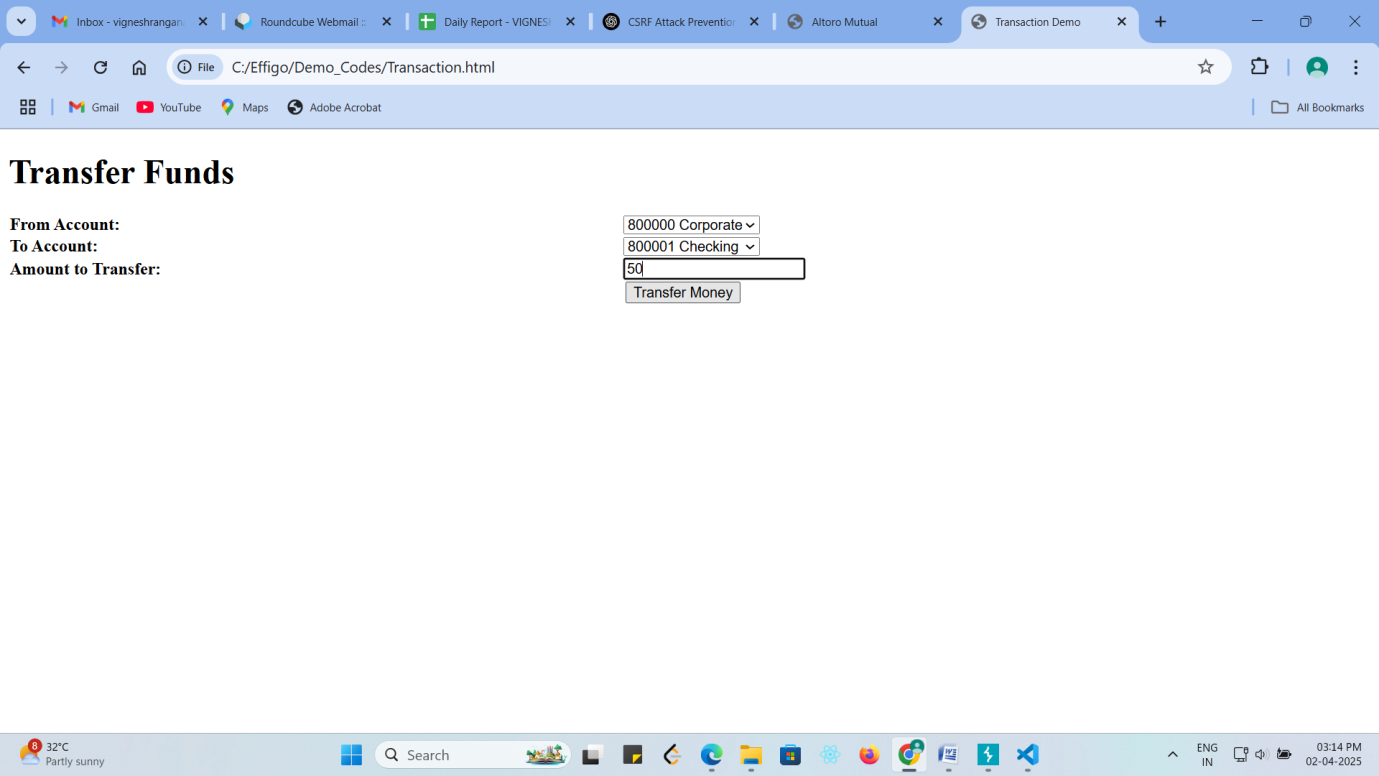
If the website responds with unexpected results (such as confirm message like amount transfer successfully from account to the particular to account), it may be **vulnerable to Cross Site Request Forgery(CSRF).**

Prevention Strategies :

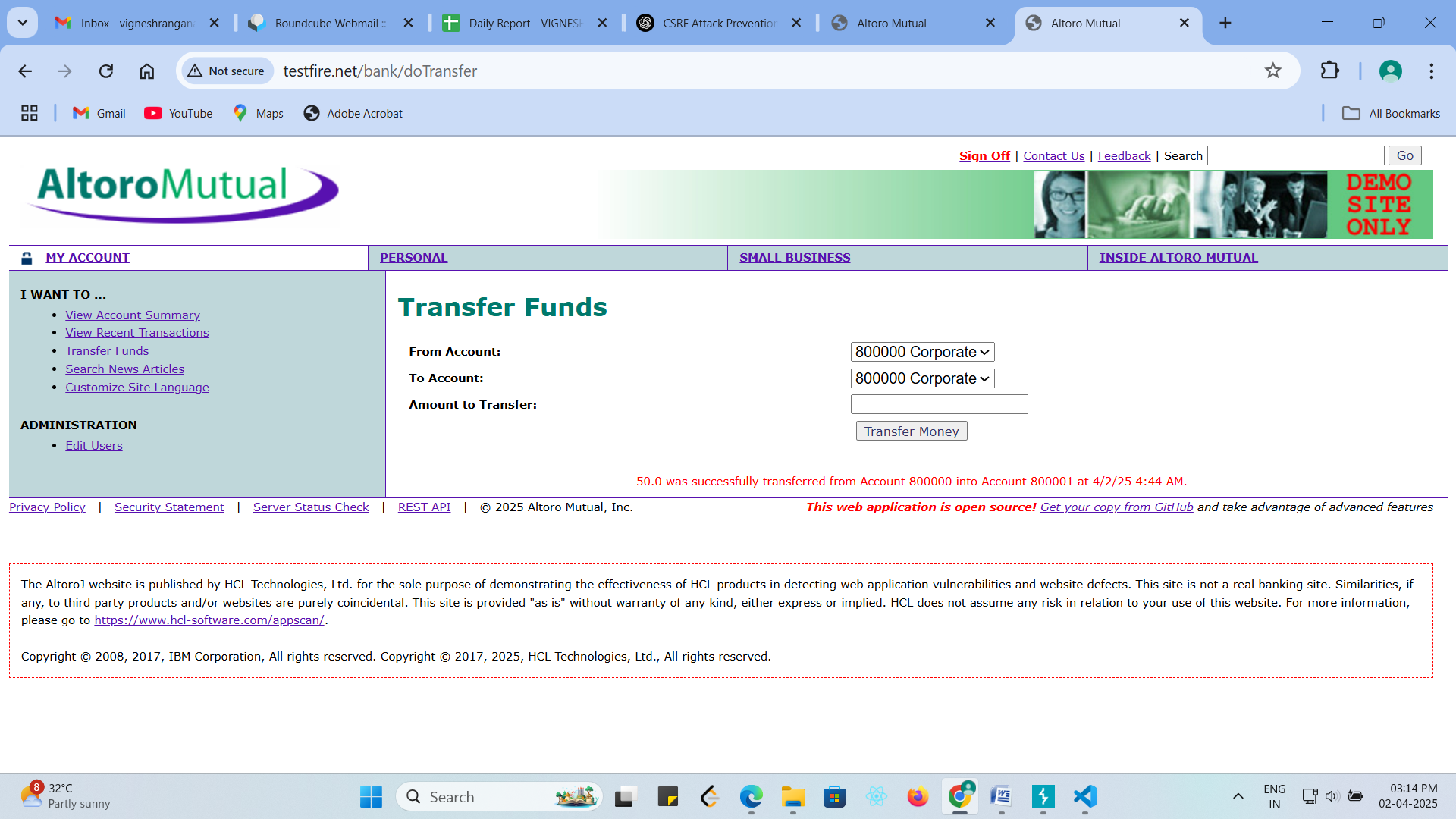
\*) Use CSRF token to validate authenticated user and prevent the web application from cross site request forgery.

****

In the original webpage authenticated the login page by using valid login credential and make the user active.



In the cloned webpage of the separate tab of the transfer funds page choosen from account and to account and entered a valid amount and clicked on transfer amount button.



The action performed on cloned webpage is reflected in the original webpage, which shows that the webpage contains a Cross site request forgery vulnerability.

4) Broken Authentication :

\*) Broken authentication is a security vulnerability that occur when a web application’s authentication mechanism are improperly implemented, allowing attacker to gain unauthorized access to user account.

\*) The broken authentication vulnerability can lead to credential theft, session hijacking, or account takeover.

Steps :

1) Open the website URL : <http://testfire.net/login.jsp>

2) Click on the Online banking Login on the top left corner of the webpage.

3) Enter the valid login credentials and authenticate the login page and keep the user active on the same webpage.

4) By right clicking the webpage we can able to inspect the page and in application section go to cookies.

5) Keep the page remain without any changes as a active user.

6) In incognito mode open the same webpage URL : <http://testfire.net/login.jsp> without login inspect and open the same cookies tab.

7) Just copy the cookies name and value of the cookies in original page and copy it in the incognito cookies section.

8) After successfully copy the cookie refresh the page and check whether the user of the cookies id is logged in that incognito mode.

If the user has login without using any username and password, only by hijacking the session cookies, it may be **vulnerable to Broken authentication.**

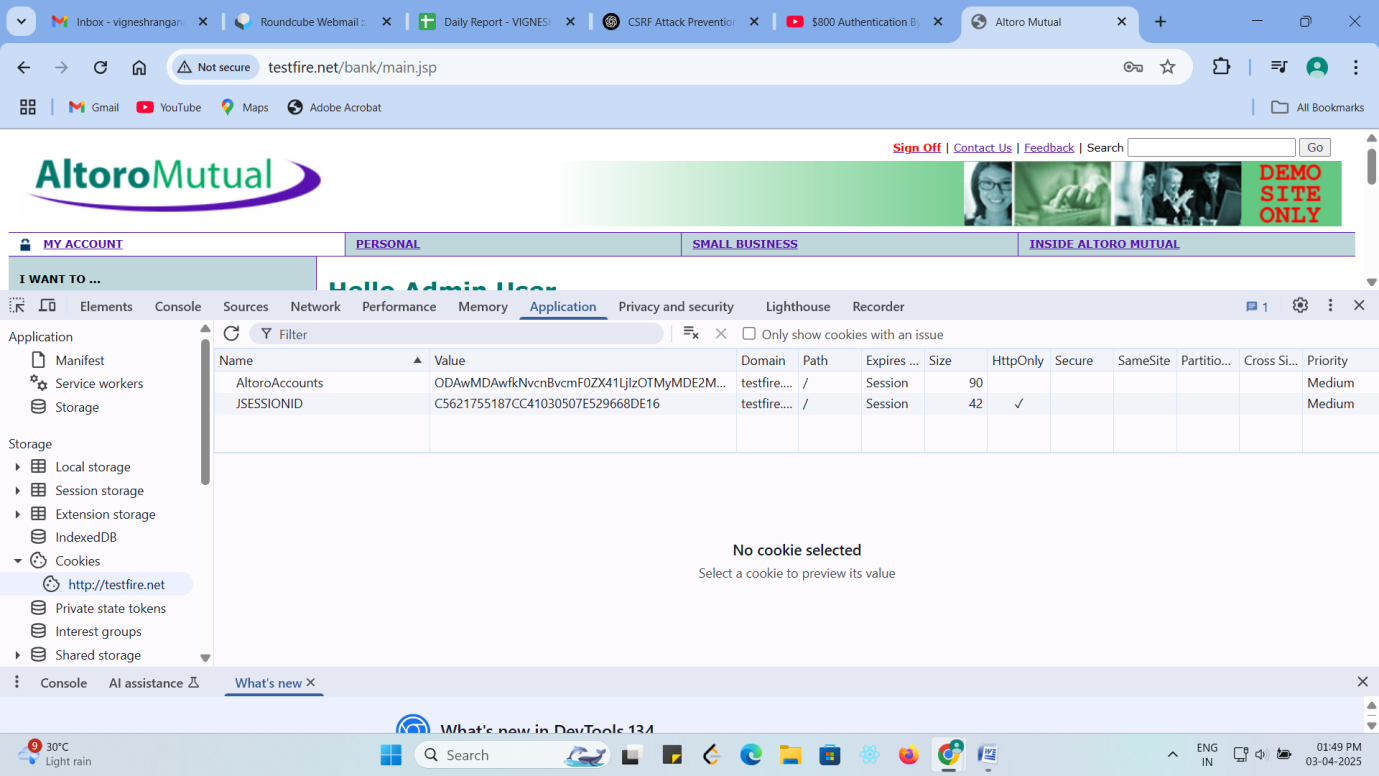
Prevention Strategies :

\*) Enforce strong password policies.

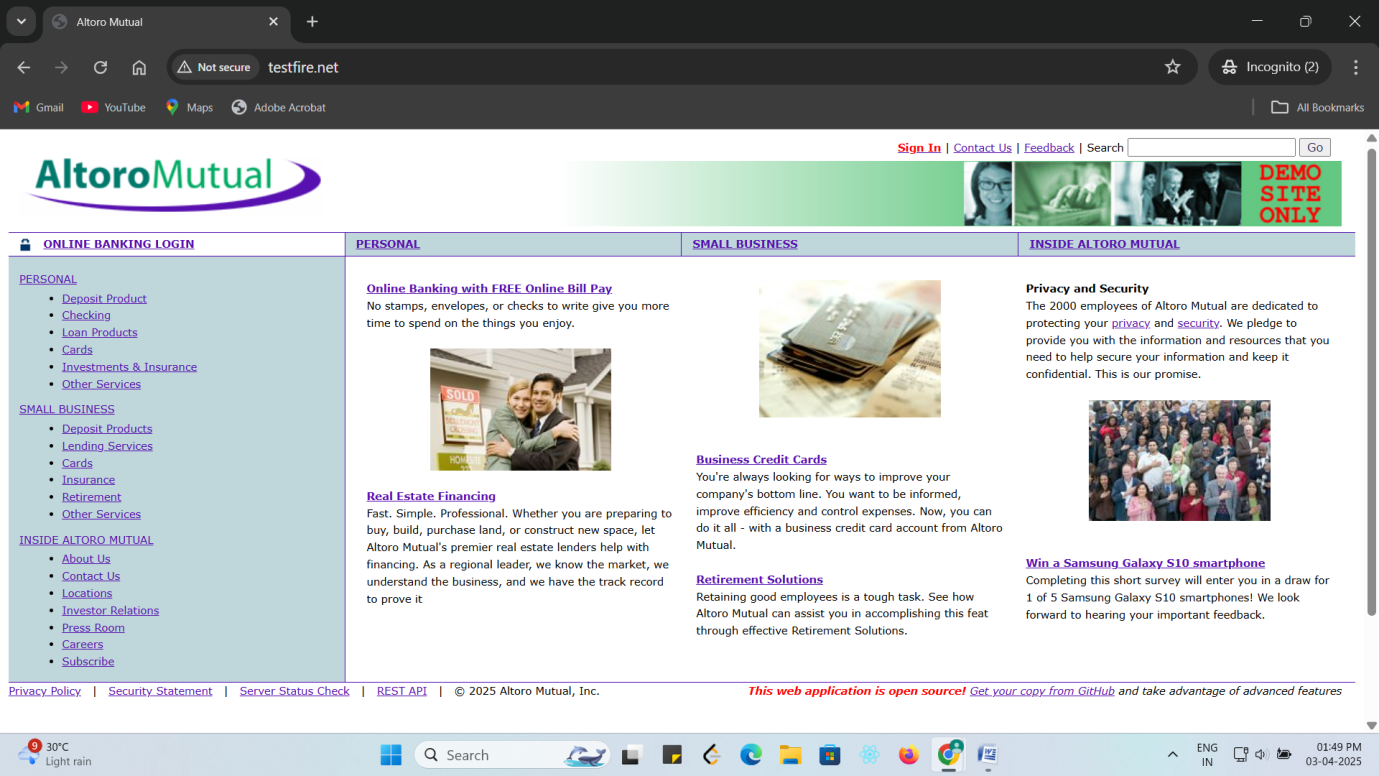
\*) Enable multi-factor authentication.

\*) Implement secure session management.

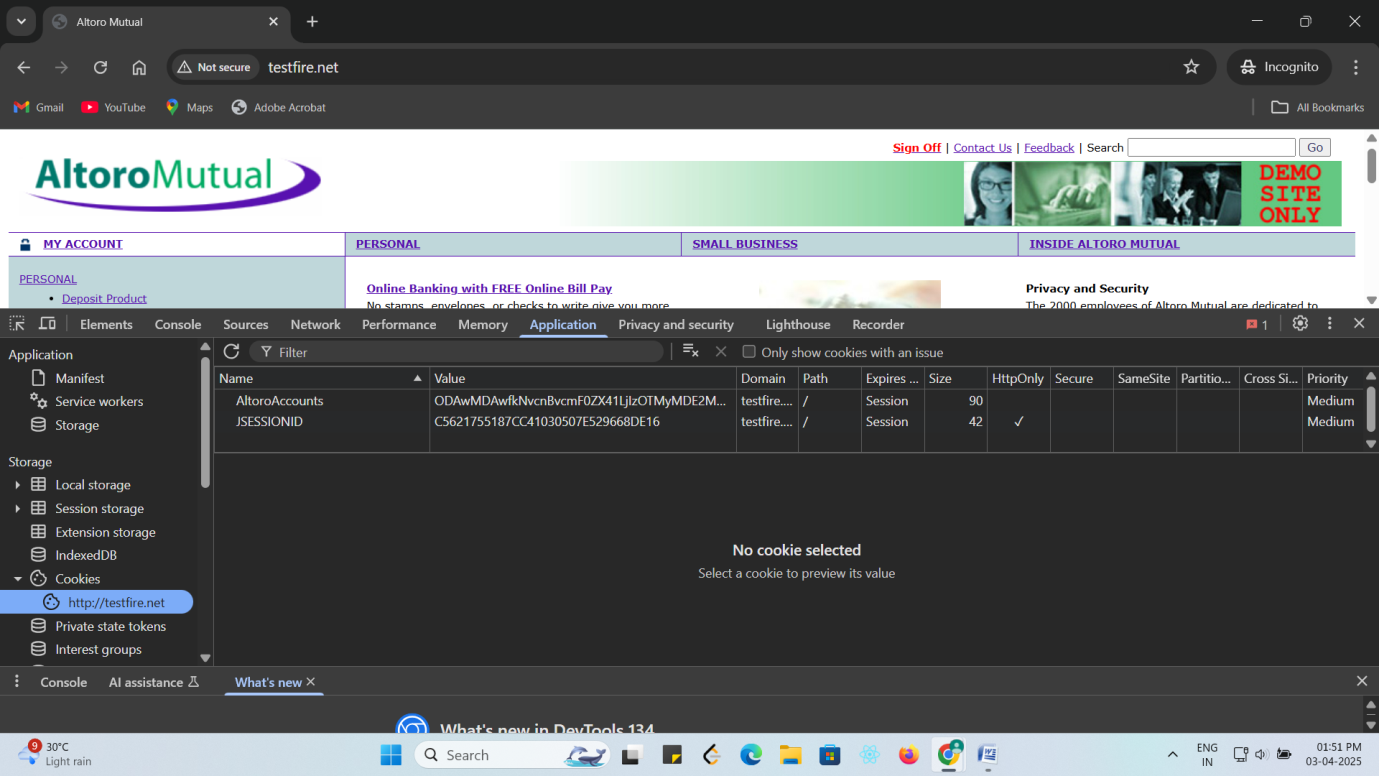
\*) Use Secure storage for credentials.



The user has logged in on the webpage using a valid login credential(Username,password) and by inspecting the webpage the session cookies of the particular active user is captured.



In the incognito mode opened the same webpage you can see that user did not login still now.



After replicating the session cookies of active logged in user in the inactive user cookies page, we can see that the user become active, it shows that the webpage is vulnerable to broken authentication.

5)Sensitive Data Exposure:

\*) Sensitive data exposure is web security vulnerability where application fail to adequately protect sensitive information such as password, credit card detail, personal details, and API keys, making them accessible to attackers.

\*) The sensitive data exposure is one of the top 10 OWASP vulnerability and can lead to theft, fraud, financial loss, and legal consequences.

How Sensitive data Exposure happen in web application :

1) Storing sensitive data in plaintext.

2) Insecure data transmission.

3) Weak (or) No Encryption.

4) Data leakage via logs or Error message.

5) Exposed API keys or Token.

6) Publicly accessible cloud storage or database.

Prevention Strategies:

1) Encrypt Sensitive data (At Rest and In Transit).

2) Hash and salt passwords.

3) Enforce secure communication.

4) Secure API key and sensitive configuration.

5) Prevent data leakage in logs and Errors.

6) Secure cloud storage and database.

6)Security Misconfiguration:

\*) Security misconfiguration occur when web application, server, database, or cloud service are improperly configured, exposing sensitive data and making them vulnerable to attack, data breaches, and unauthorized access.

\*) The security misconfiguration can have severe consequences, affective data security, user privacy, system availability, and business reputation.

Common causes of Security Misconfiguration:

1) Default credential and insecure settings.

2) Exposed error message and debugging mode enabled.

3) Unrestricted access to sensitive files and directories.

4) Improper security header.

5) Misconfigured cloud storage and database.

Prevention Strategies:

1) Remove default account and default credentials.

2) Disable debug mode and detailed error message.

3) Secure file and directory permissions.

4) Implement security headers.

5) Secure cloud storage and database.

7)Server-side Request Forgergy (SSRF):

\*) Server side request forgery (SSRF) is web Security vulnerability where an attacker manipulates a web application to make unauthorized HTTP request to internal or external resources.

Server-side Request Forgery (SSRF) often allow attacker to :

1) Access internal system that are not directly accessible from the internet.

2) Performing port scanning on the internal networking.

3) Fetch sensitive data from cloud metadata services.

4) Exploit internal API to execute Remote Code Execution (RCE).

Prevention Strategies:

1) Block request to internal IPS.

2) Use allow list instead of block list.

3) Disable unnecessary HTTP methods.

4) Use web application firewall(WAF).

5) Restrict cloud metadata access.

8)Insecure Design:

\*) Insecure design refers to fundamental flaws in an application architecture, logic, or security control, making it vulnerable to attck.

\*) Insecure design means the application lacks proper security measures from the start.

\*)Example : A banking application allowing passwordless fund transfer without proper verification is an insecure design flaw.

How Insecure Design Occur :

1) Lack of security-first mindset.

2) Missing threat modeling.

3) Weak access control and authorization.

4) Business logic flaws.

5) No rate limiting or security control.

Prevention Strategies:

1) Implement Security design pattern.

2) Perform thread modeling.

3) Enforce secure authentication and authorization.

4) Implement business logic validation.

5) Apply rate limiting and API security measures.