"Web Application Security Assessment – Practical Findings"

Detailed Findings:

1 HTML Injection:

Affected URL: https://demo.testfire.net/search.jsp?q=TEST

Description:

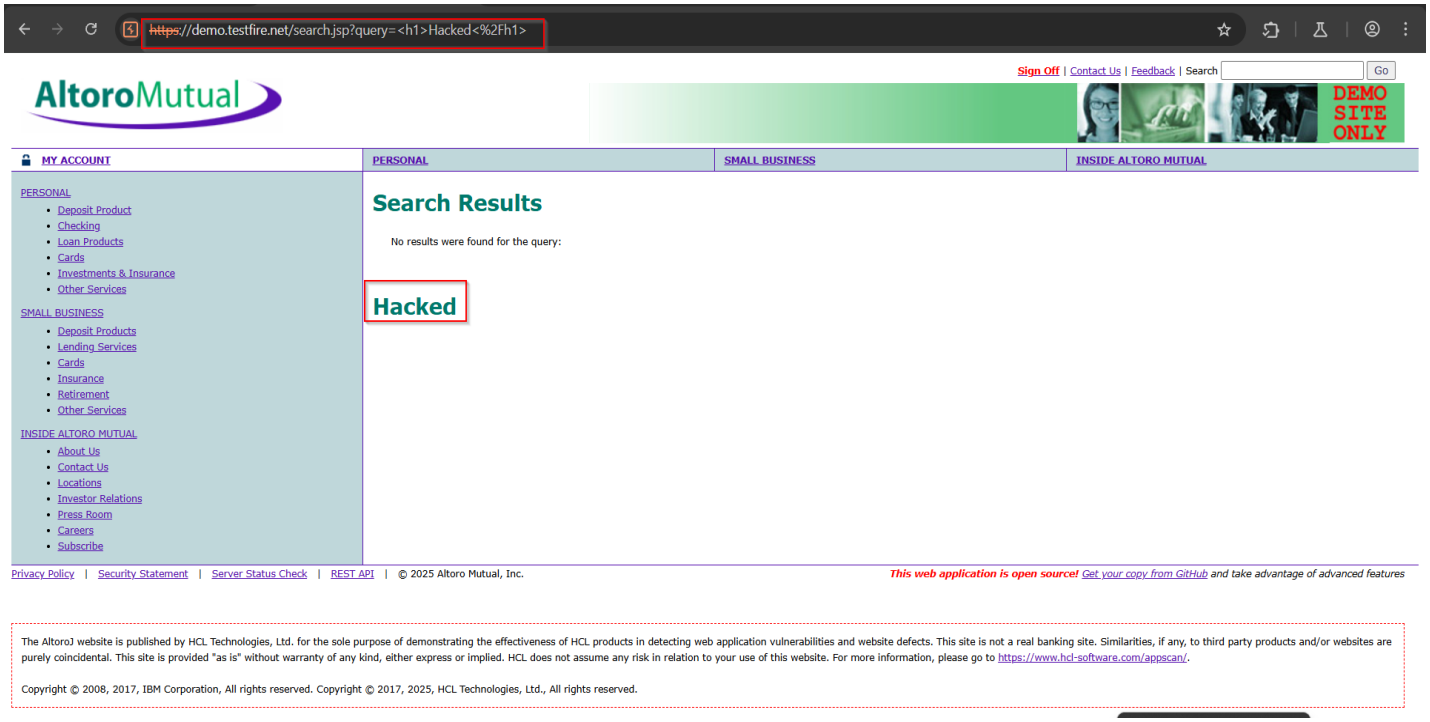
HTML Injection is a web security vulnerability where an attacker is able to inject HTML code into a web page due to improper input handling.

This usually happens when:

- User input is not properly sanitized.
- Output is not encoded before displaying it in the browser.

As a result, attackers can modify how the web page behaves or appears to users.

Evidence:



The above screenshot shows how the HTML code is rendered directly on the web page, proving that HTML Injection is possible

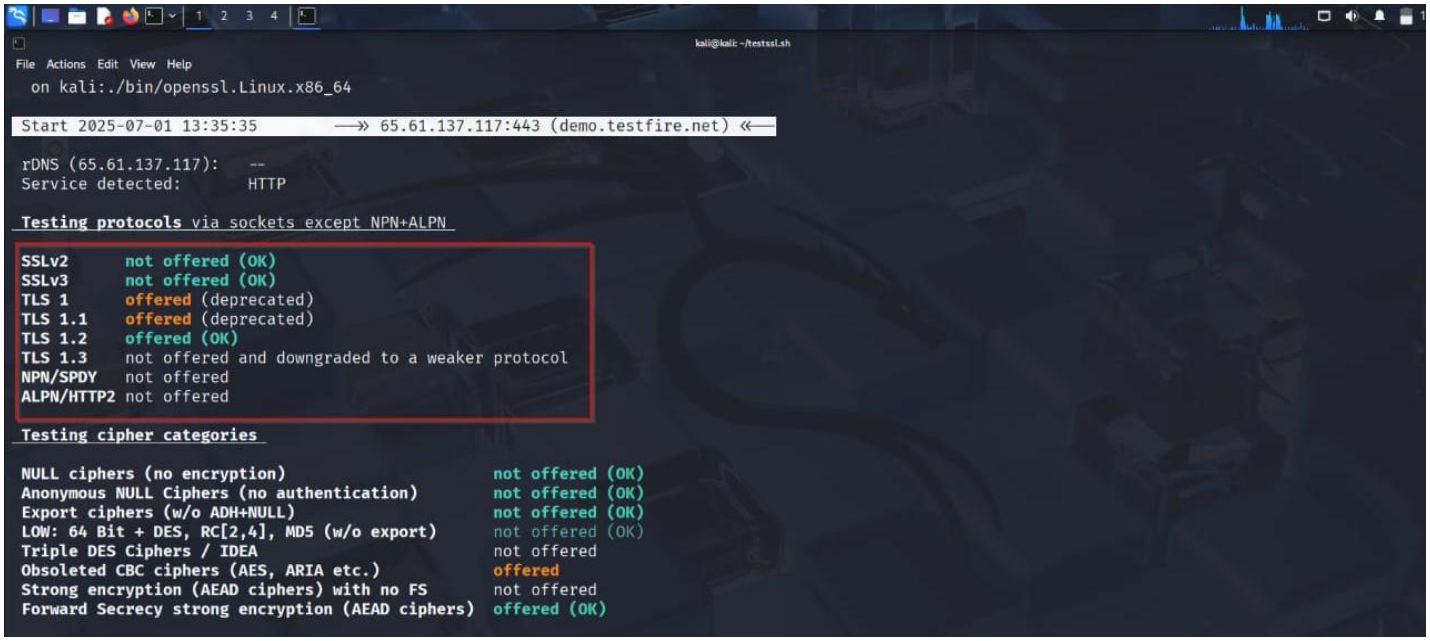
2)Deprecated Protocols Enabled:

Affected URL: https://demo.testfire.net

Description:

Deprecated protocols are old versions of secure communication protocols that are no longer considered safe. They have known vulnerabilities, weak encryption, and are not recommended for use.

Evidence:



The above screenshot shows that the server supports deprecated protocols, which are considered insecure.

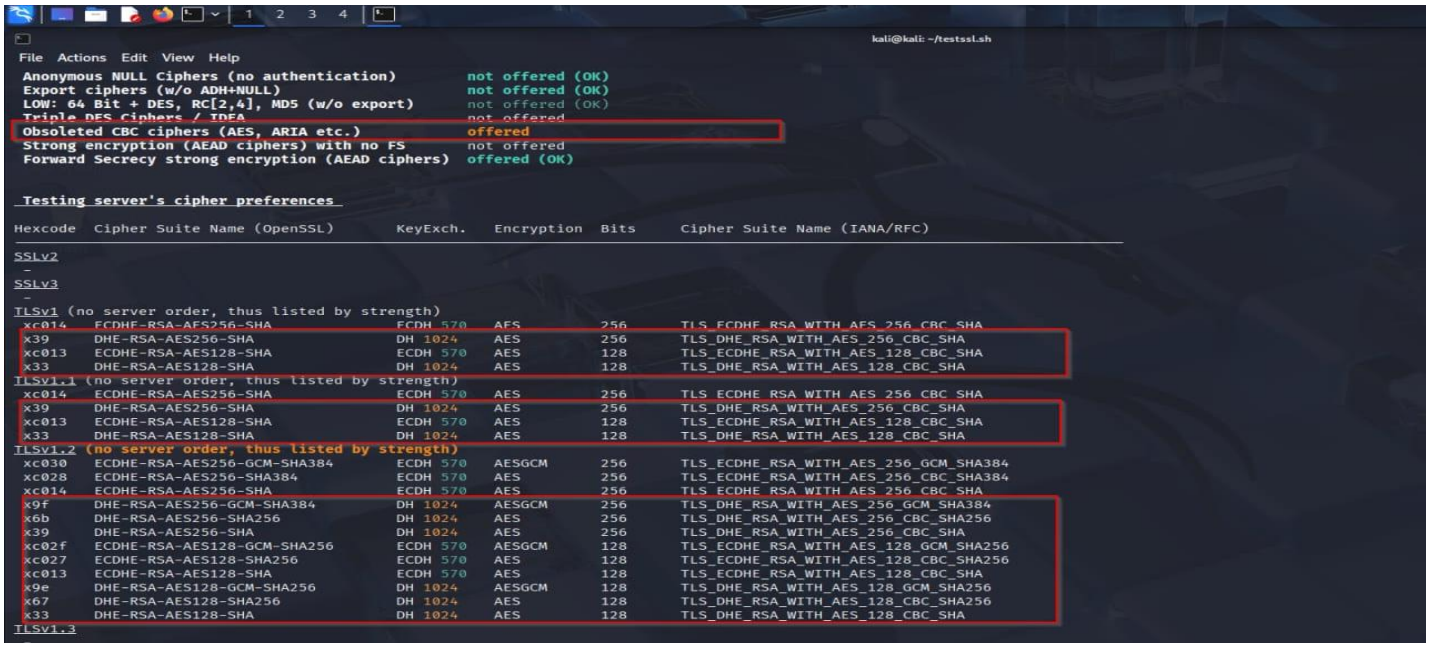
3)Weak Ciphers:

Affected URL: https://demo.testfire.net

Description:

Weak ciphers are outdated or insecure encryption algorithms used in SSL/TLS communication. They make the connection between the server and client vulnerable to attacks like brute-force, downgrade, and Man-in-the-Middle (MITM).

Evidence:



The above screenshot shows that the server supports weak SSL/TLS ciphers, which can be exploited by attackers.

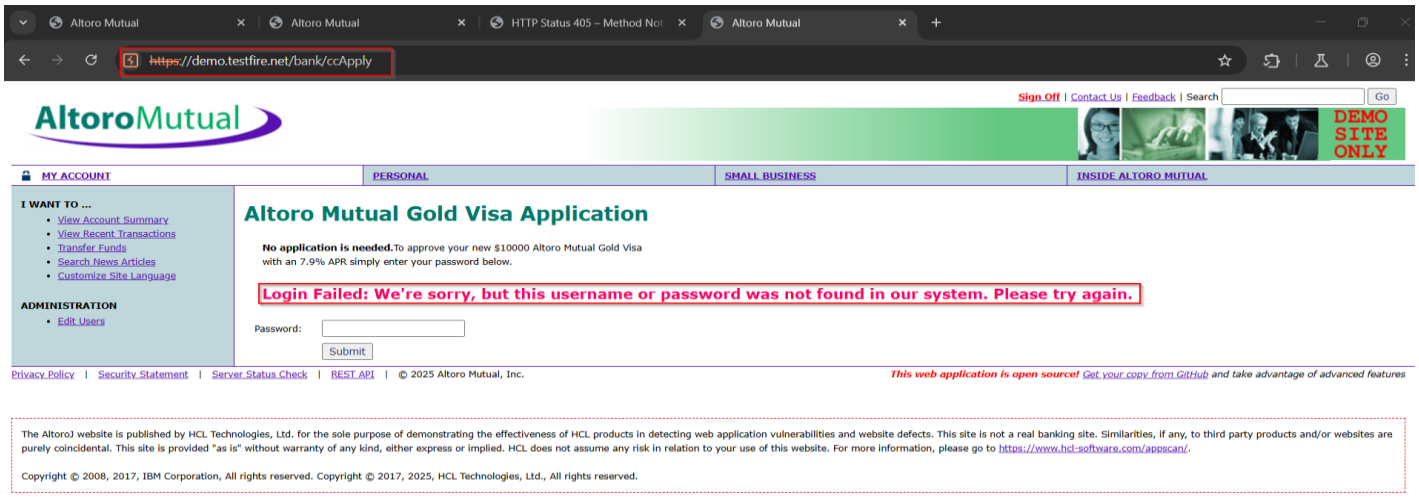
4)Information Disclosure through Error Messages

Affected URL: https://demo.testfire.net/search.jsp

Description:

Detailed error messages can help attackers find security flaws, guess file paths, identify database technologies, or prepare for further attacks like SQL injection or path traversal.

Evidence:



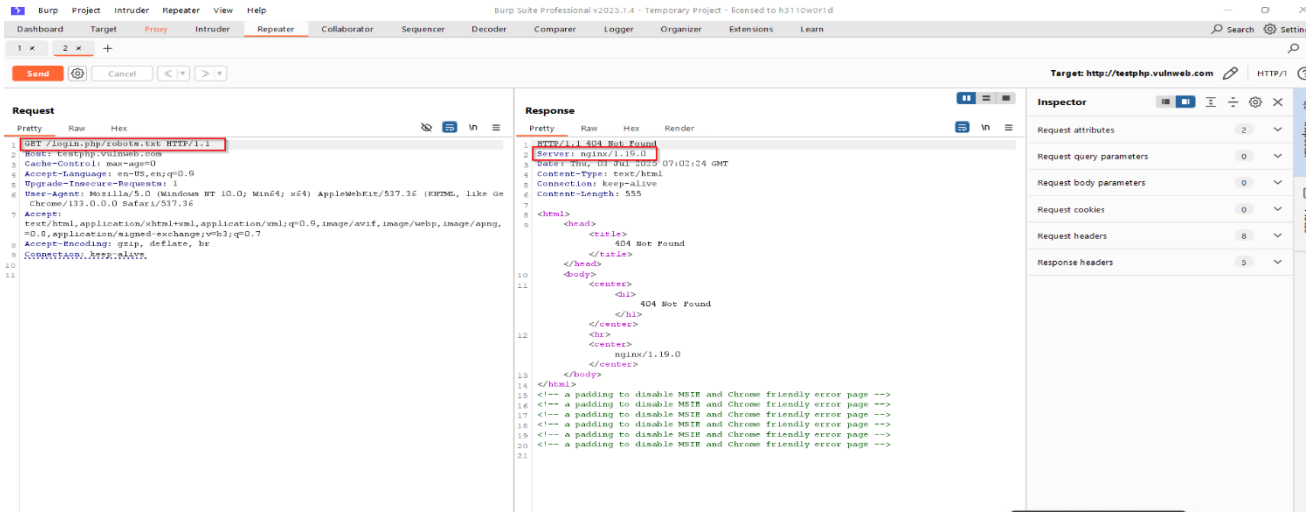
The above screenshot shows an error message revealing backend details, indicating information disclosure.

5)Robots.txt:(Forced browsing information disclosure)

Affected URL: https://demo.testfire.net/robots.txt

Description:

The robots.txt file is used to guide search engine crawlers, but sometimes it can accidentally reveal sensitive information. In this case, it exposed server-related details such as the server version or server type, which can be useful to attackers.



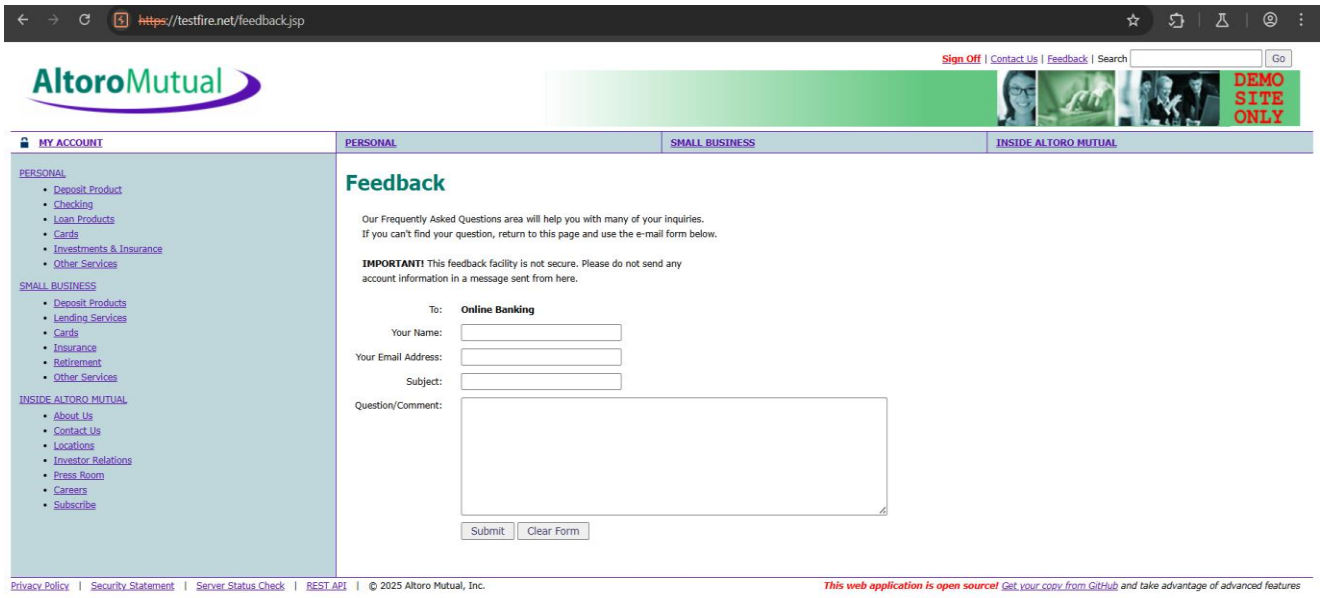
The above screenshot shows that the robots.txt file reveals server information like version or server type.

6) Data Semantics (Weak input validation):

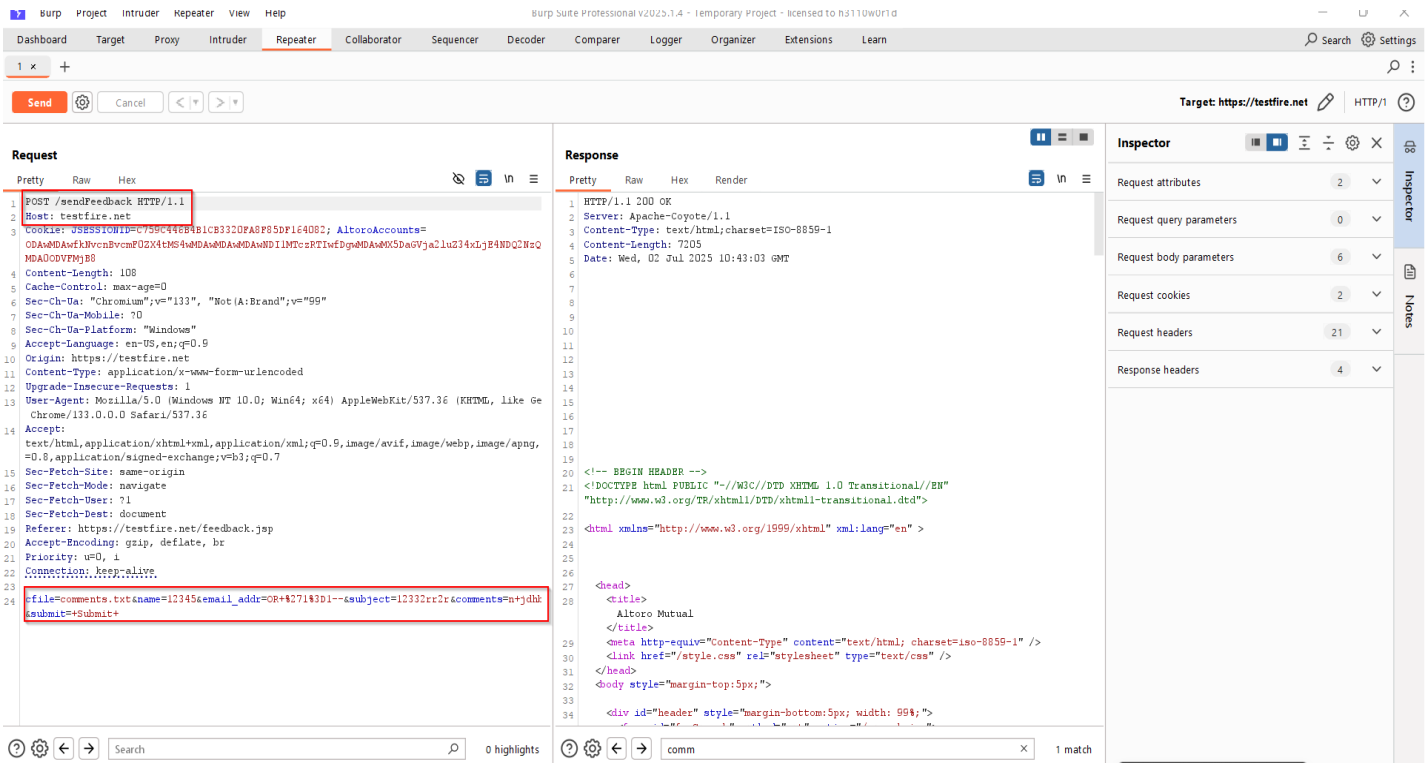
Description:

Data semantics testing checks whether the application properly validates user inputs — like email, phone number, or password — to ensure they follow the correct format and type. If weak validation is in place, attackers may bypass security and inject malicious inputs.

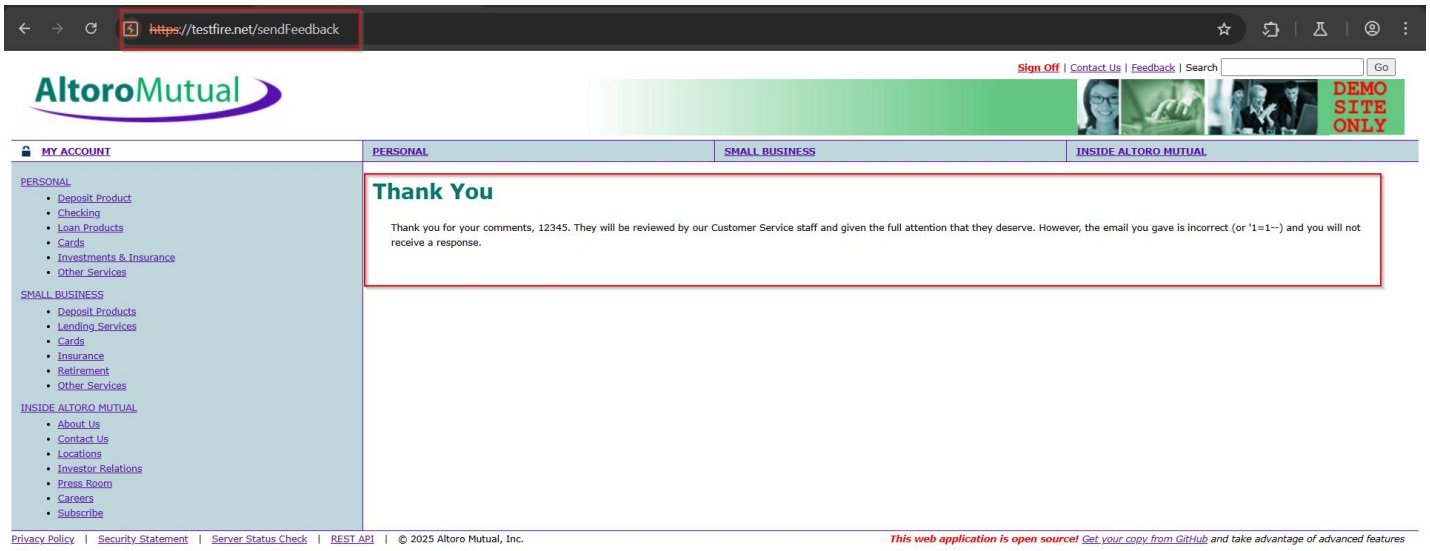
Evidence:



The below feedback shows how the application accepts invalid or unexpected input without proper validation.



The above screenshot shows that the application failed to properly validate input formats, allowing unexpected or potentially harmful data.



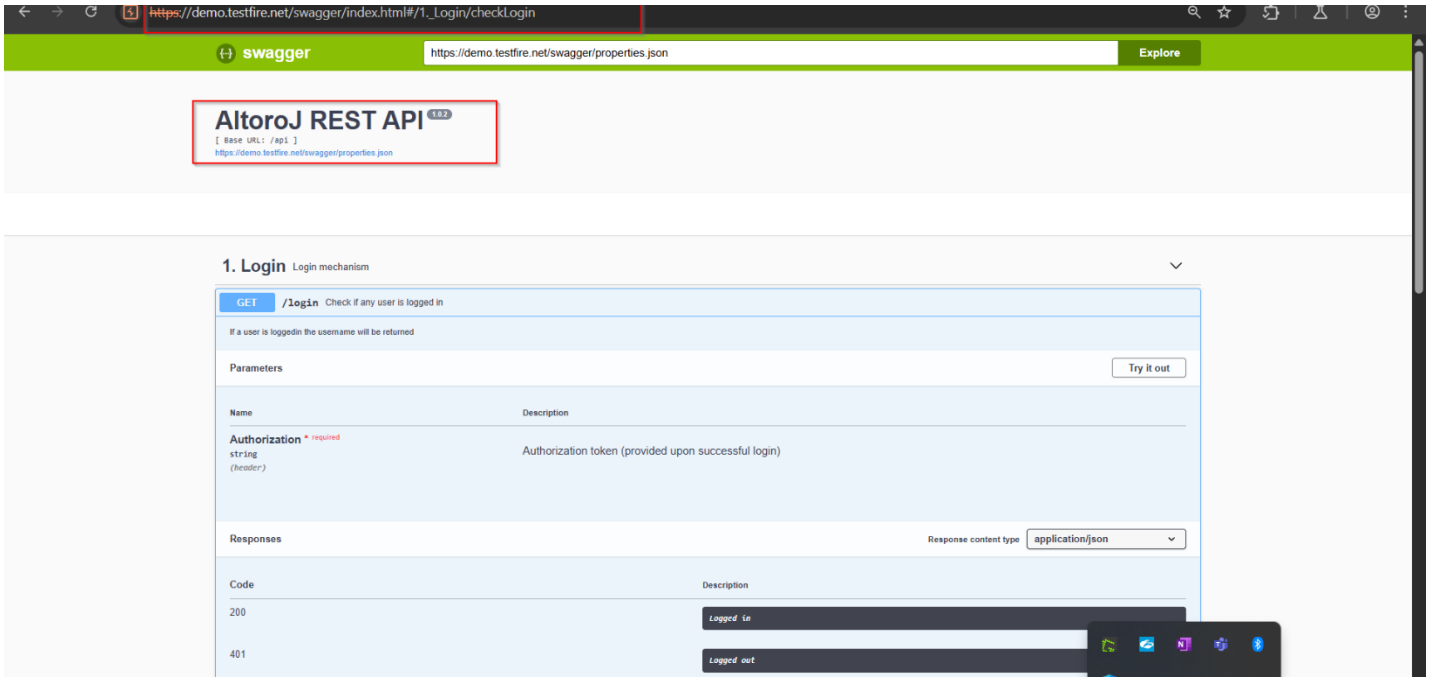
The above screenshot shows that the application failed to properly validate input formats, allowing unexpected or potentially harmful data.

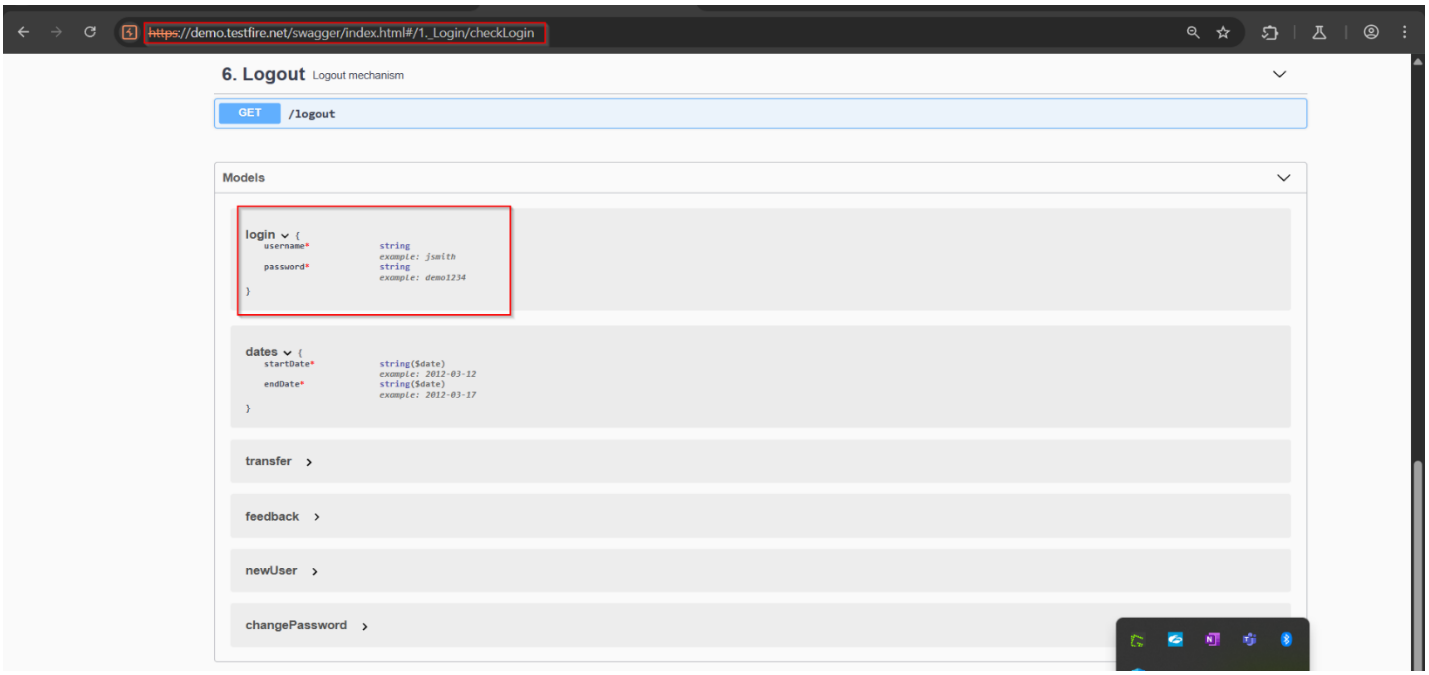
7)Sensitive Information Disclosure (Exposed Swagger API Documentation):

Description:

Swagger UI is publicly accessible at https://demo.testfire.net/swagger/, revealing full internal API documentation. This includes sensitive endpoints like /login, accepted parameters, expected responses, and headers — which can help attackers understand the API structure and potentially craft malicious requests.

Evidence:





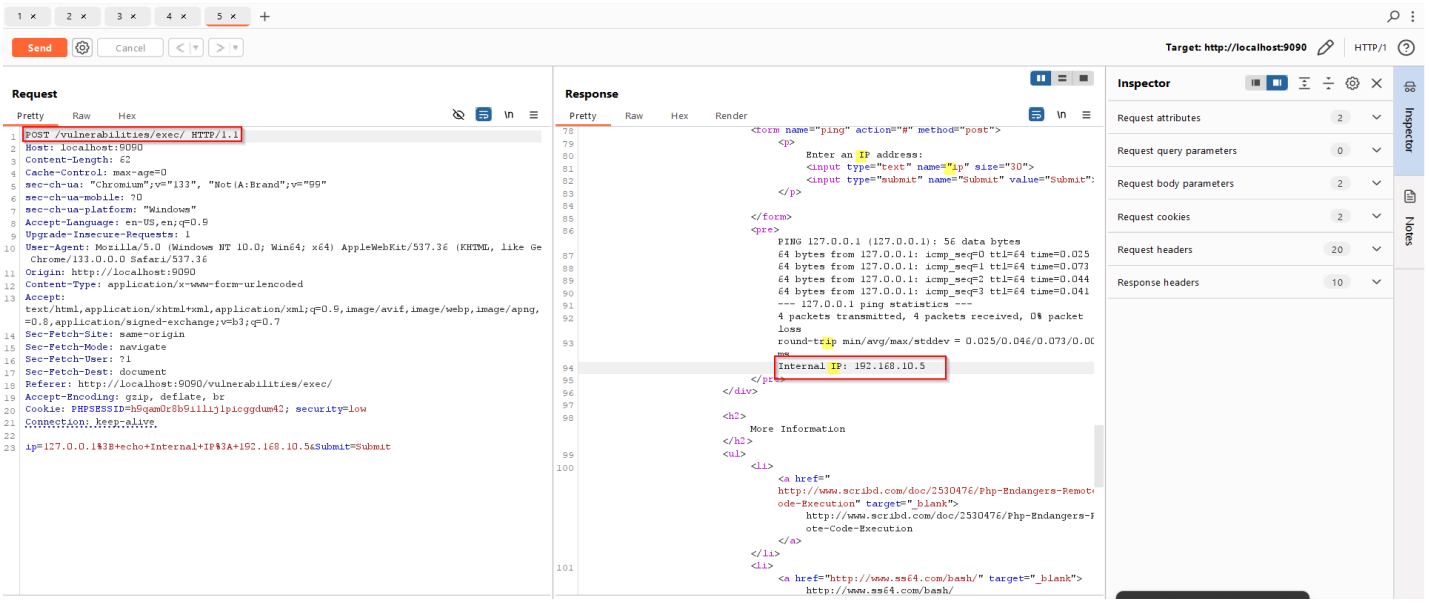
The screenshot below shows the publicly accessible Swagger UI disclosing the full AltoroJ REST API structure, including login functionality and authentication details.

8) Internal IP Disclosure Severity:

Description:

Internal IP addresses like 192.168.x.x, 10.x.x.x, or 172.16.x.x were found in the application's response or source code. These are meant to stay private within internal networks but were unintentionally exposed.

Evidence:



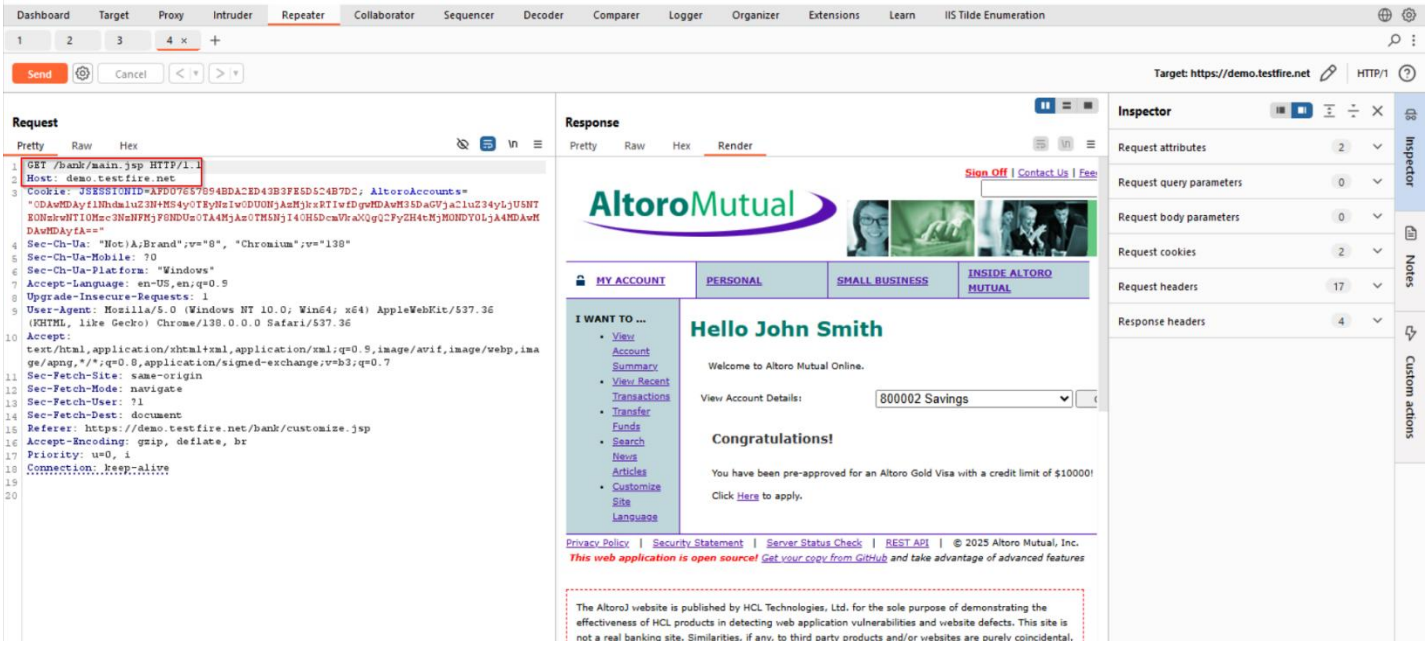
The screenshot below shows that an internal IP address leaked in the response, which should not be visible to users.

9)Privilege Escalation:

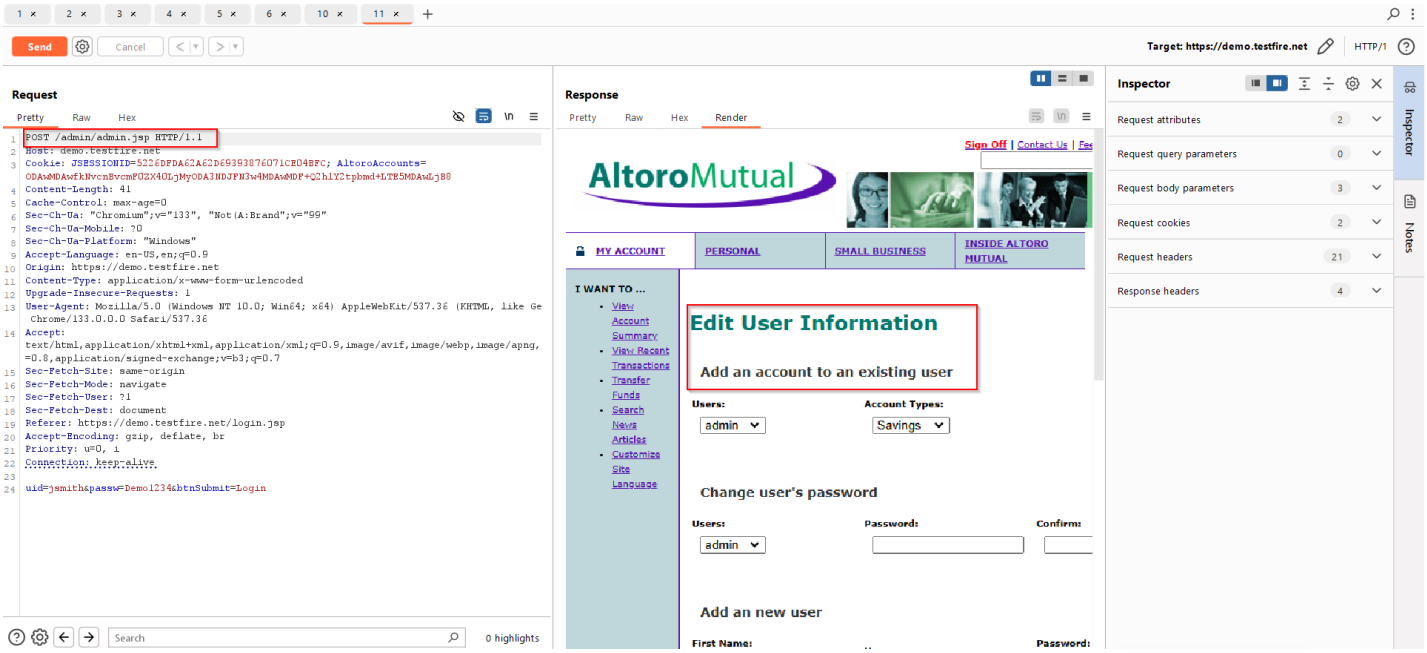
Description:

The application allows a regular user to access admin-level features by directly accessing or submitting a request to an admin-only endpoint (/admin/admin.jsp). No server-side validation or access control mechanism is preventing this role change. This is a critical misconfiguration that enables privilege escalation.

Evidence:



The screenshot shows a normal user logged in.



The screenshots show that a regular user successfully accessed admin functionalities by directly navigating to /admin/admin.jsp without being restricted.

10) Clickjacking:

Description:

Clickjacking is a type of attack where a malicious website tricks the user into clicking on something different from what they perceive — usually by loading the target site in a transparent <iframe> over a fake page. This vulnerability exists when a website doesn't restrict its content from being embedded in a frame.

Evidence:

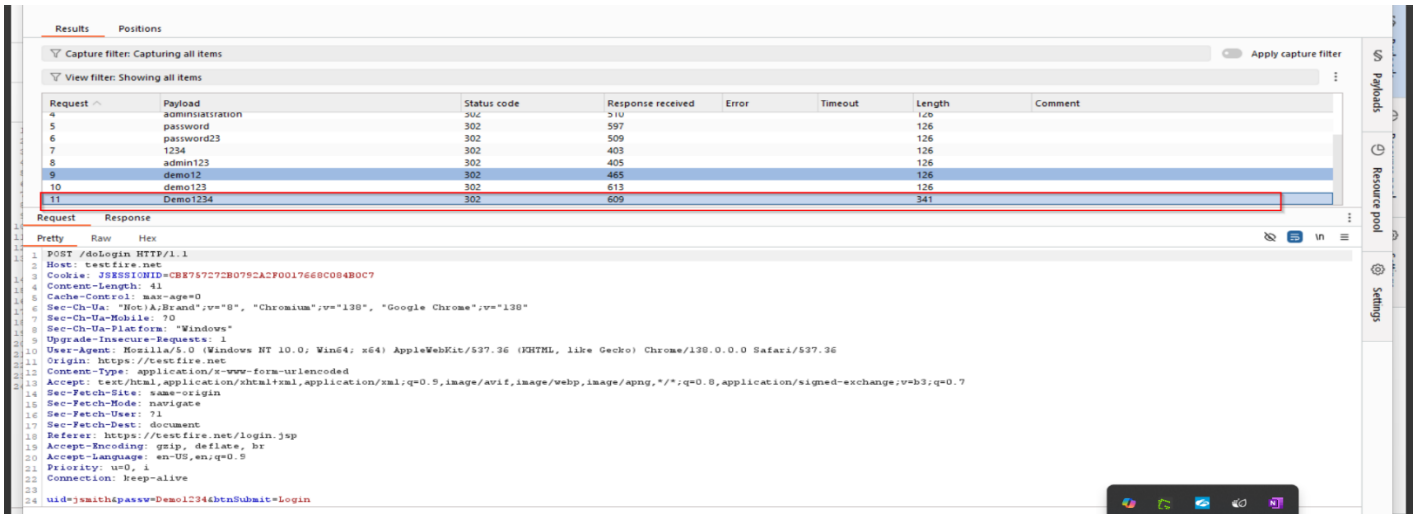
The screenshot demonstrates the target website being embedded within an iframe on another page, confirming that it is vulnerable to clickjacking attacks due to the absence of proper frame protection headers.

11)Account Takeover by Brute Force

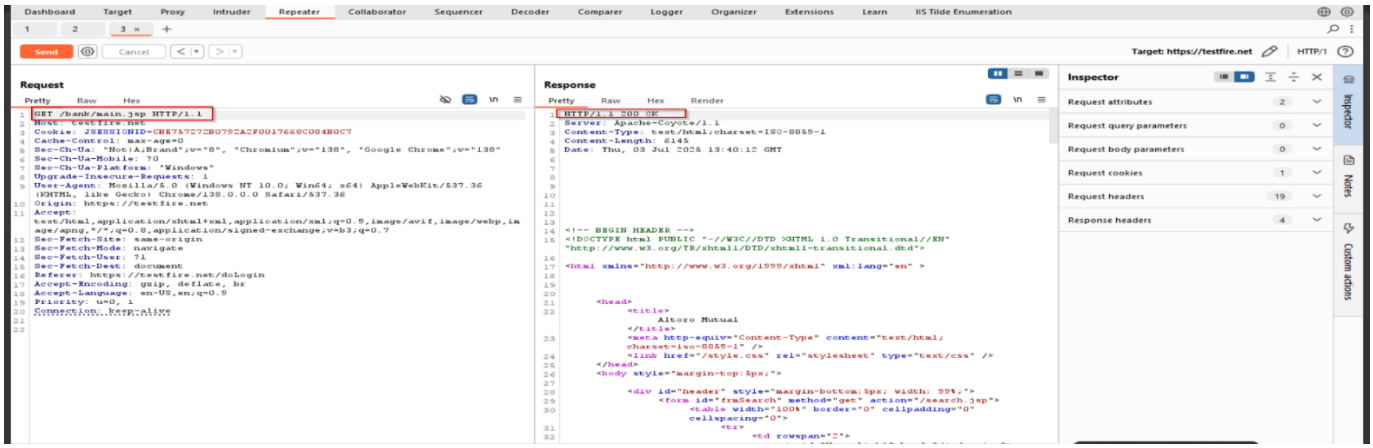
Description:

Brute force attack is when an attacker tries to guess a valid username and password by submitting many login attempts using different combinations. If the application does not block or limit repeated login attempts, it allows attackers to guess weak passwords and take over user accounts.

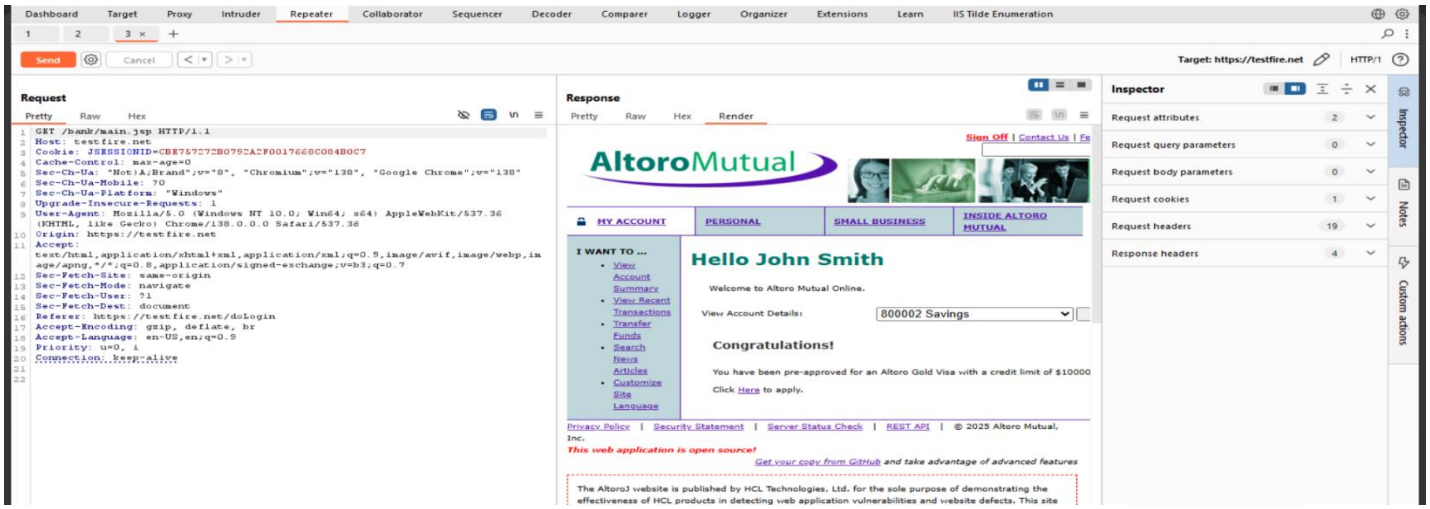
Evidence:



The screenshot shows multiple rapid login attempts made without any lockout or CAPTCHA, indicating that brute force protection is missing



Shows a valid password being guessed and accepted after several attempts.



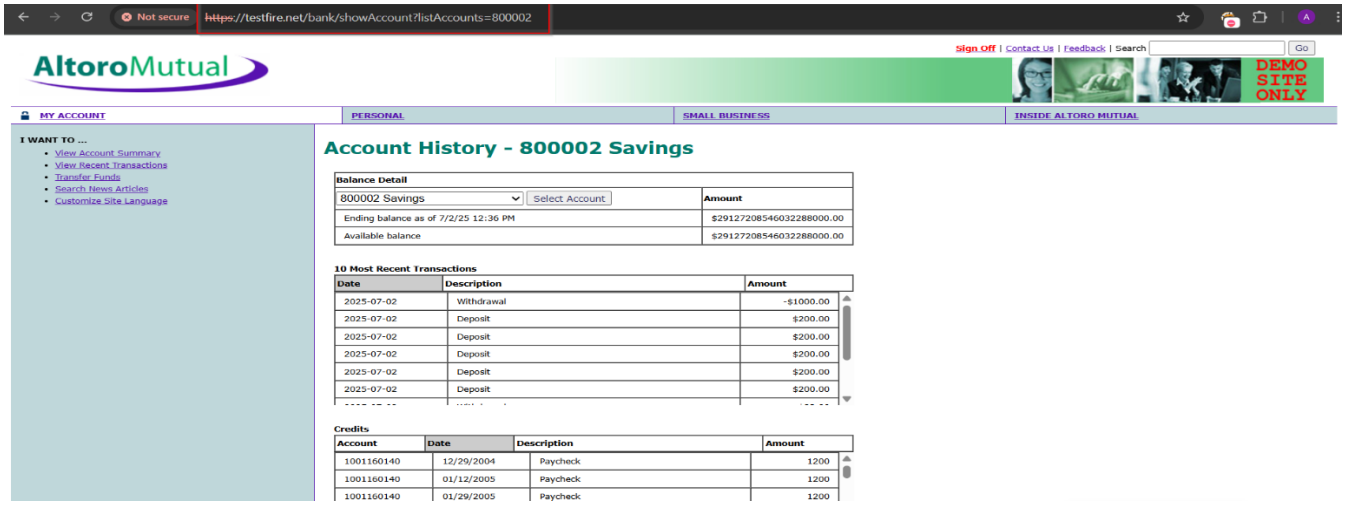
Confirms successful login into the user's account, proving account takeover through brute force.

12 Insecure Direct Object Reference (IDOR)

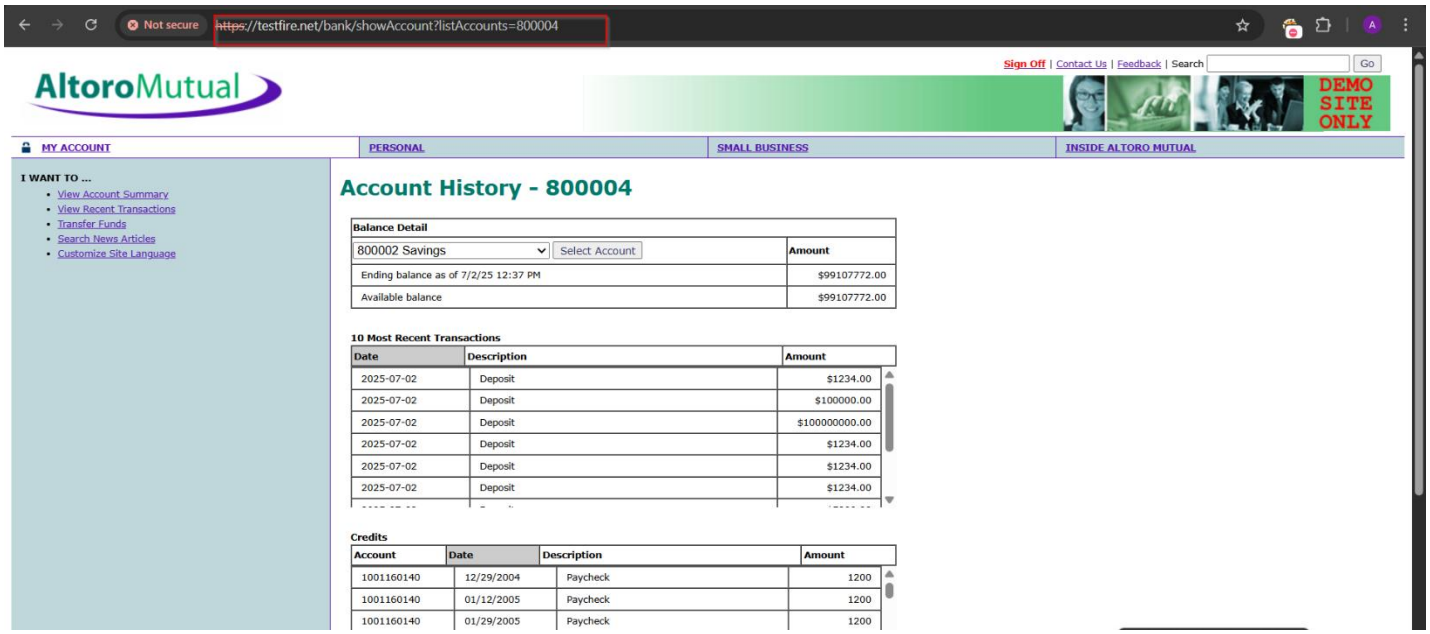
Description:

IDOR occurs when a user can access or modify data by simply changing a reference value (like a user ID, account number, or file name) in a request — without proper authorization checks. If the server does not verify that the user owns the requested resource, it may allow access to other users' sensitive data.

Evidence:



The user logged in using his own account ID.

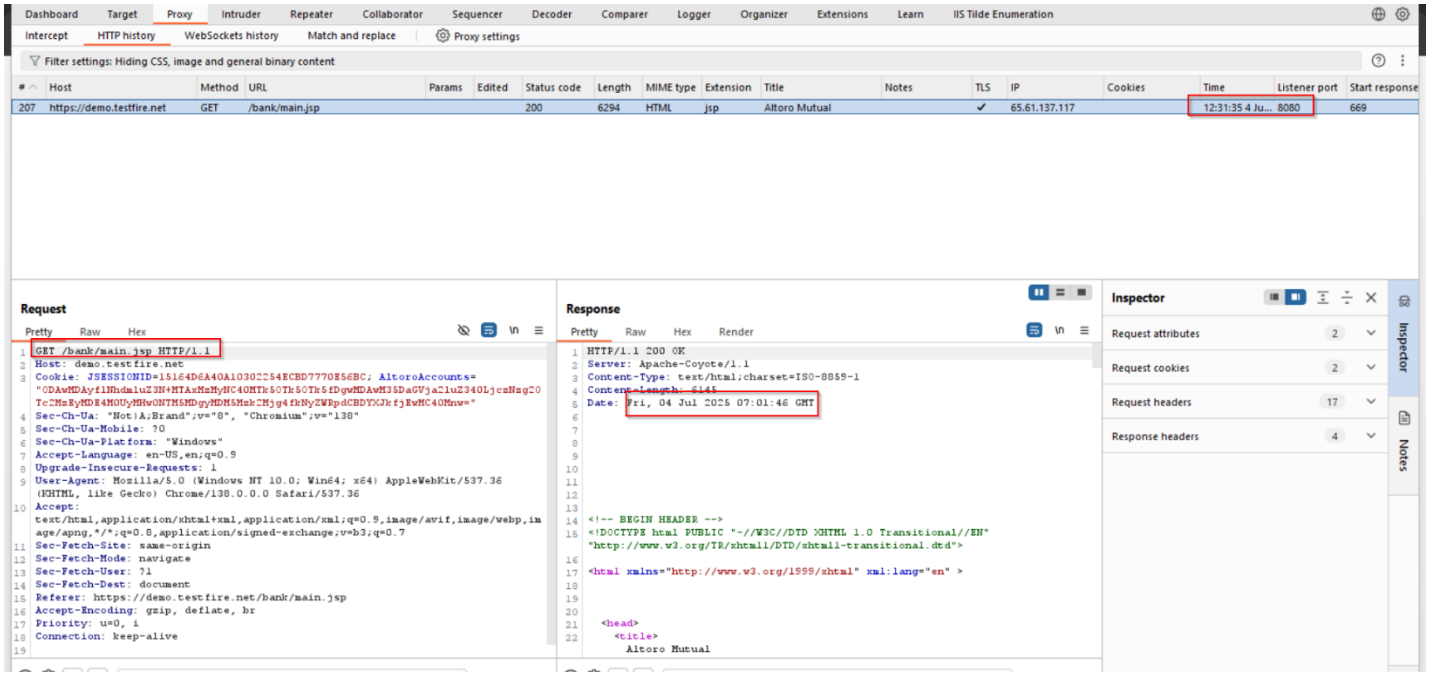


The screenshot shows that by changing the accountId parameter in the URL/request, the user was able to access another user's data, confirming the IDOR vulnerability.

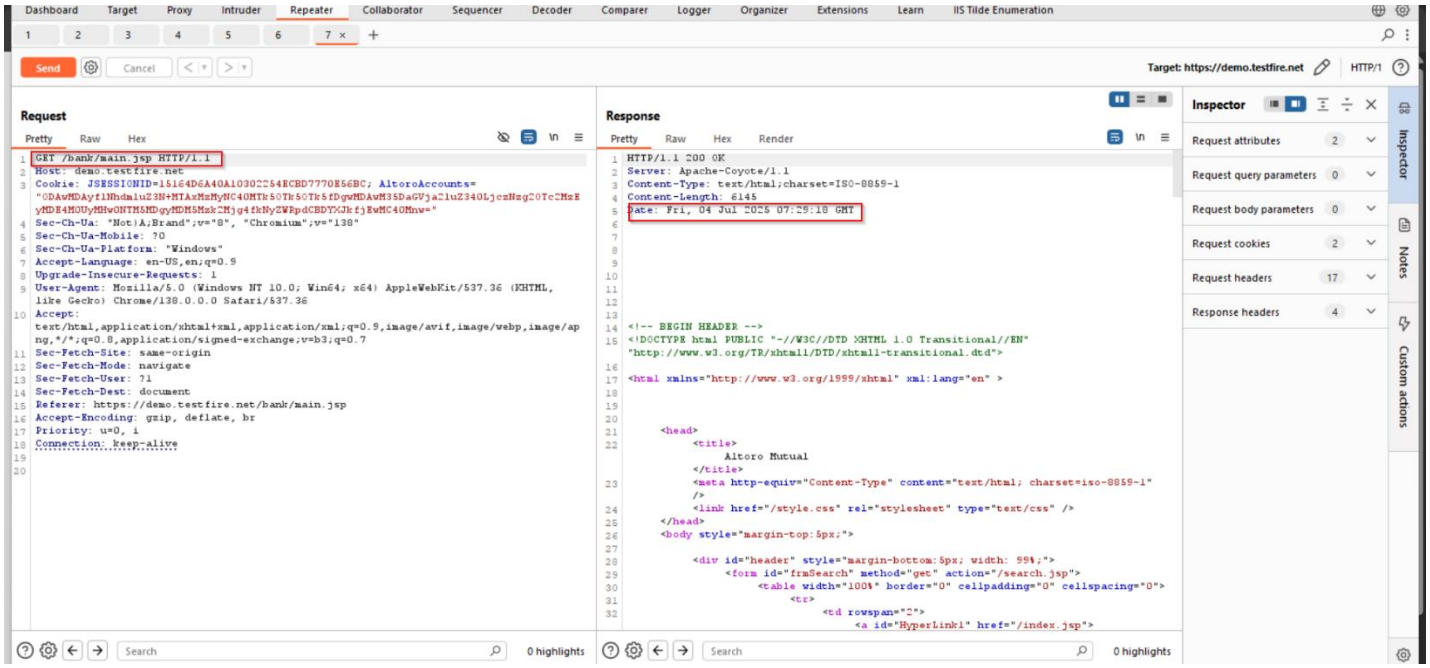
13) Excessive Session Timeout:

Description:

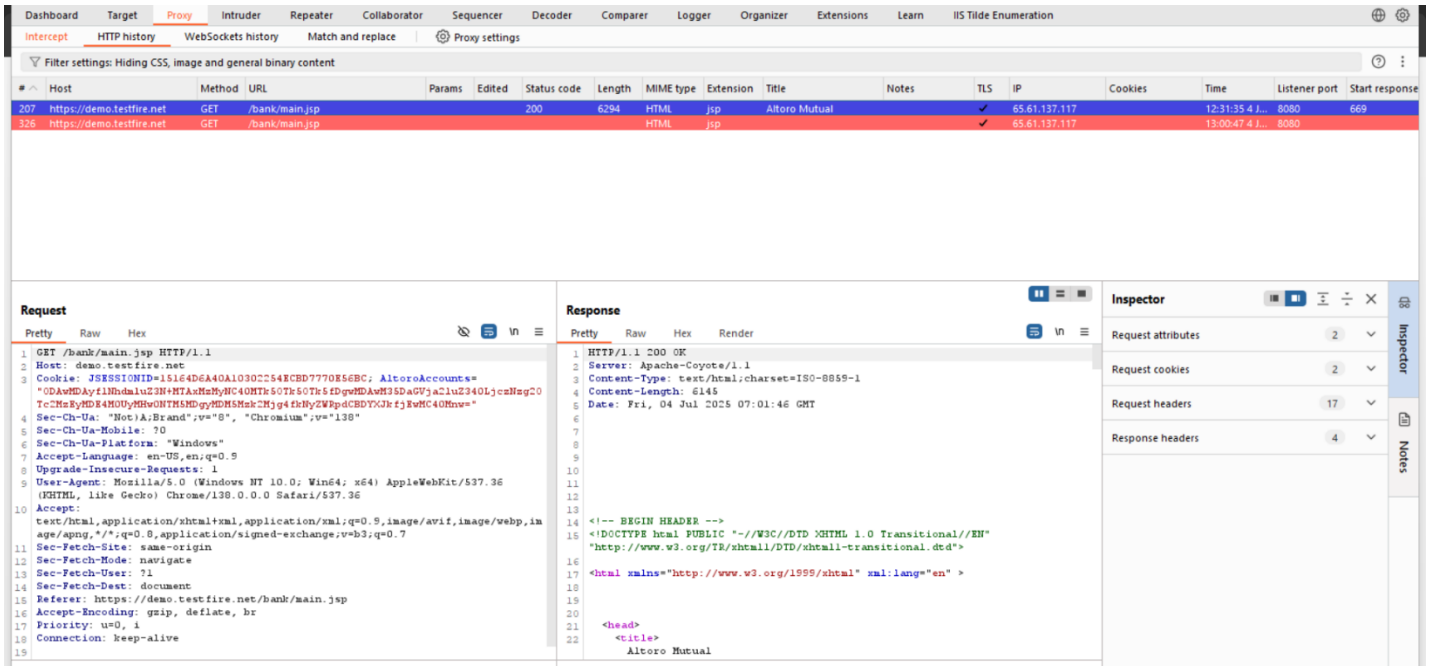
The application does not enforce a proper session timeout policy. During testing, a user logged in and remained inactive for over 30 minutes. Despite this inactivity, the session remained active, and the user was able to access authenticated pages without being prompted to log in again. This confirms that the session timeout mechanism is either missing or improperly configured.



Shows a user successfully logged in at the beginning of the test.



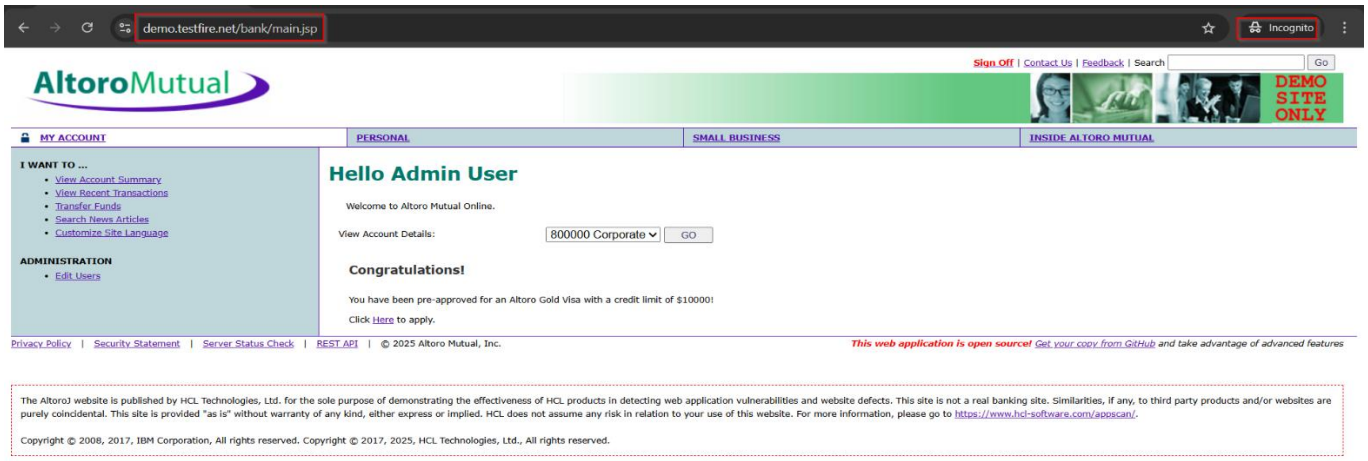
Shows the same session still active even after 30 minutes of inactivity — confirming excessive session timeout.

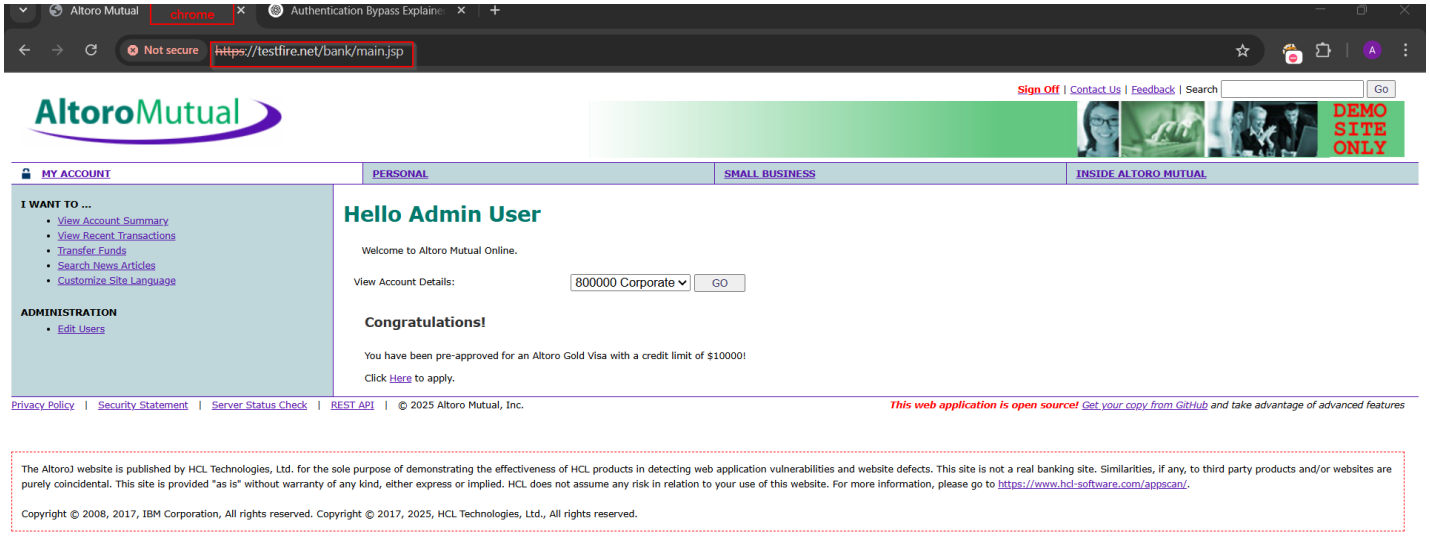


14) Concurrent login:

Description:

The application allows the same user account to be logged in from multiple browsers or devices at the same time without any restrictions. During testing, the same credentials were used to log in simultaneously from different browsers (e.g., Firefox and Chrome), and both sessions remained active without invalidating the other. This confirms that the application does not enforce any concurrent session limits for user accounts.



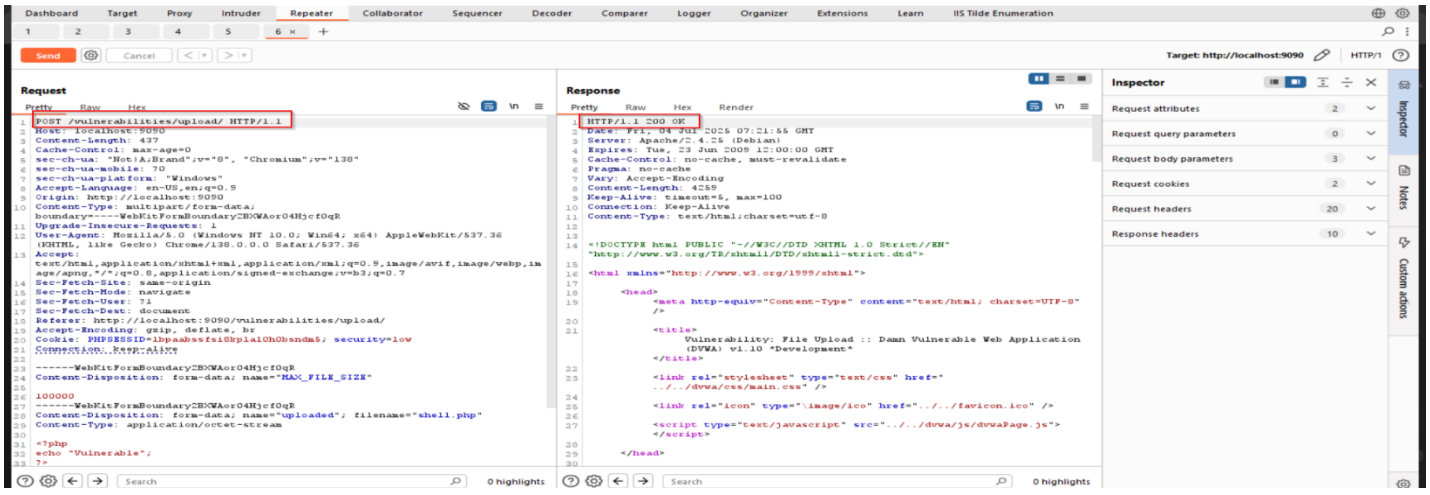


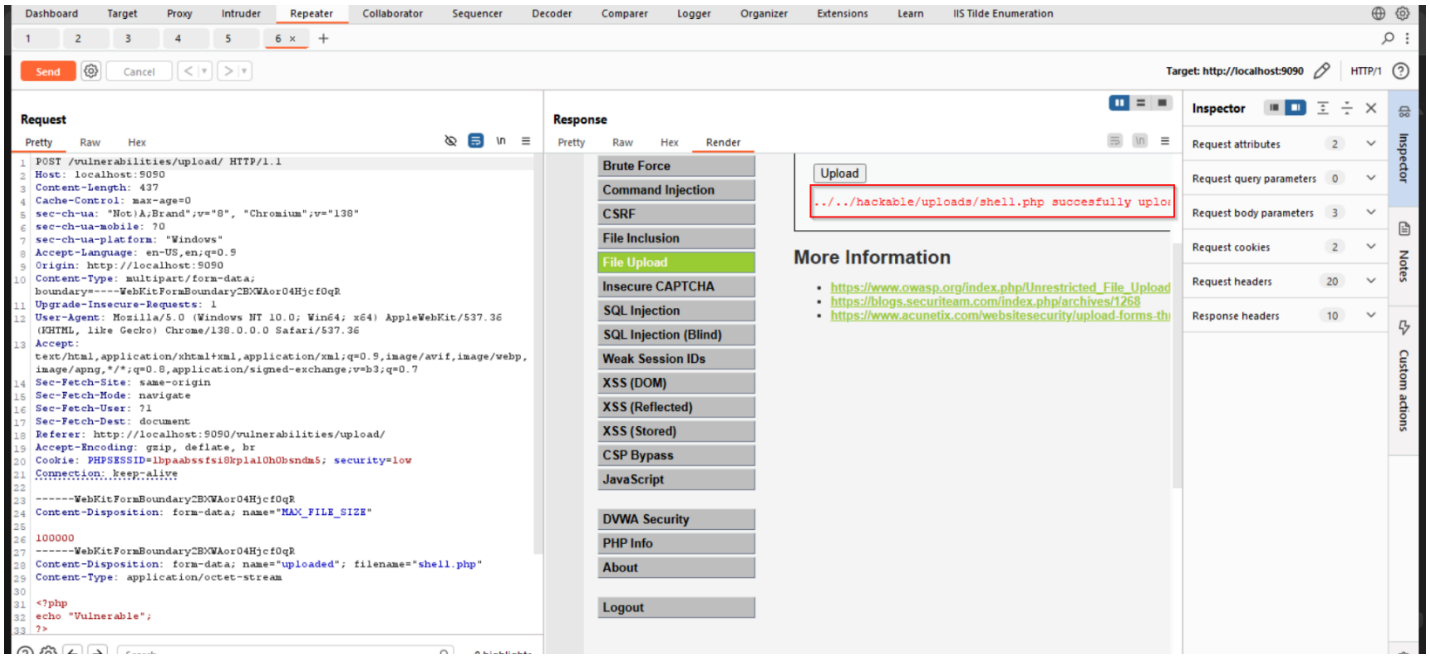
The screenshots demonstrate that the same user account was able to maintain active sessions on two different browsers simultaneously, confirming that the application allows concurrent logins without restriction.

15 File upload vulnerability:

Description:

The application permits file uploads but does not properly validate the uploaded file's extension, content, or MIME type. During testing, a .php file containing harmless PHP code was uploaded successfully, indicating that server-side validation is missing or weak. This behavior suggests that malicious files could be uploaded and potentially executed on the server.





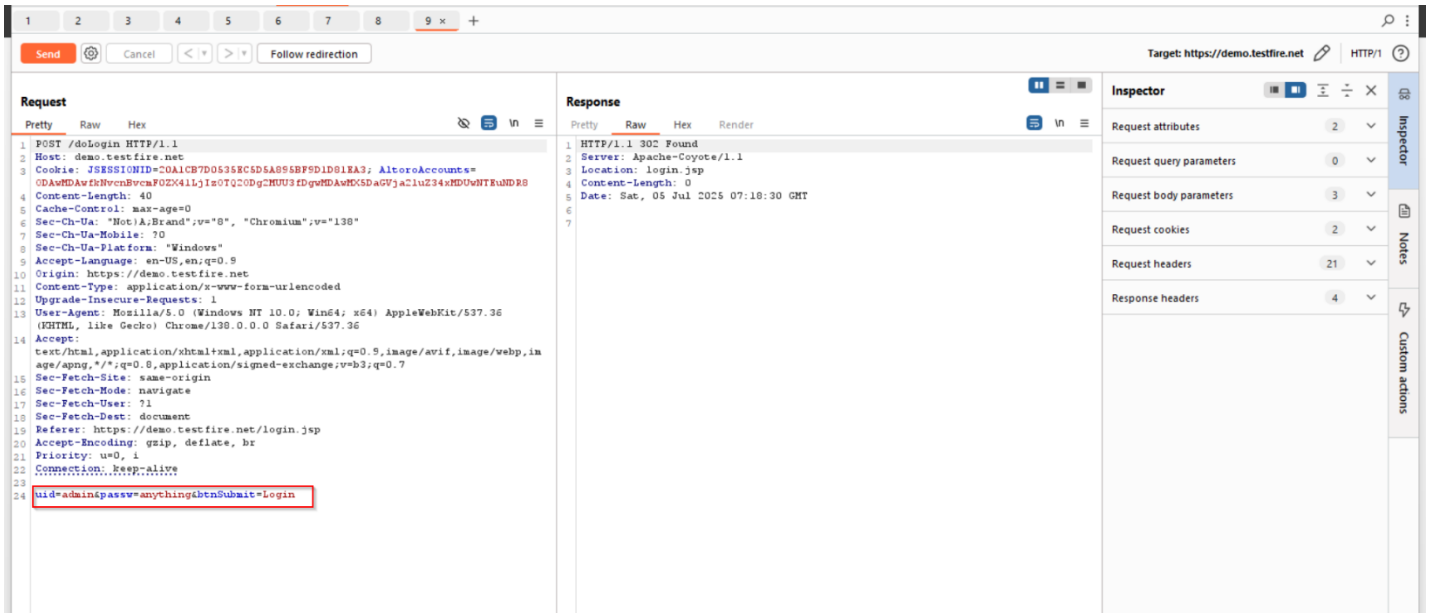
The screenshot shows that a PHP file was successfully uploaded through the file upload feature, confirming that the application accepts dangerous file types without proper validation.

16)Authentication bypass:

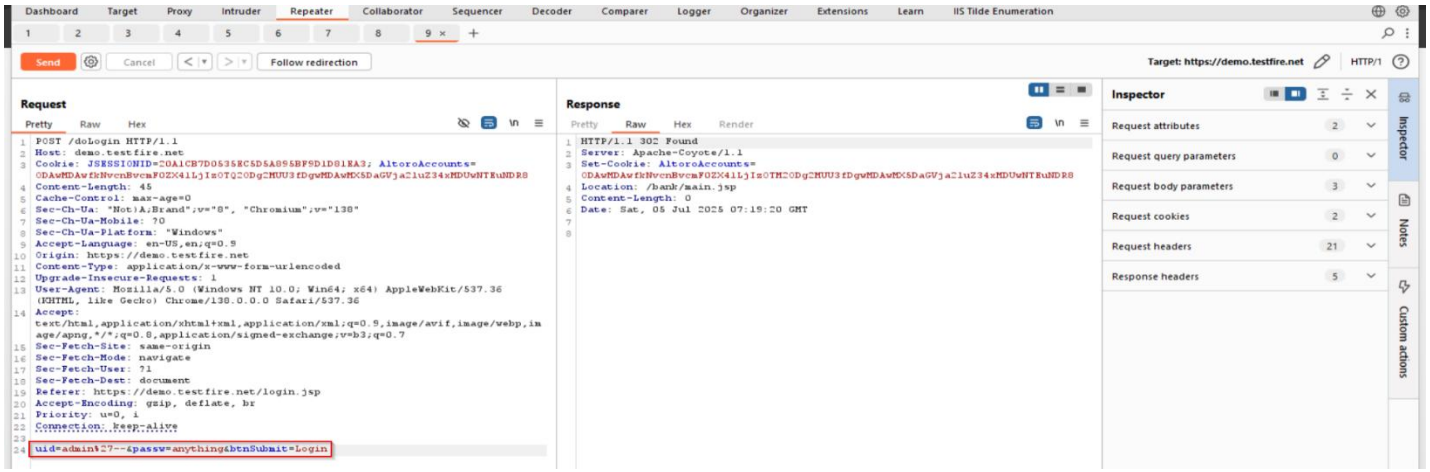
Description:

If the application allows unlimited login attempts, attackers can exploit this to guess credentials and gain unauthorized access to user accounts. This leads to full account takeover, sensitive data exposure, identity misuse, or privilege escalation. If admin accounts are affected, complete system compromise may occur.

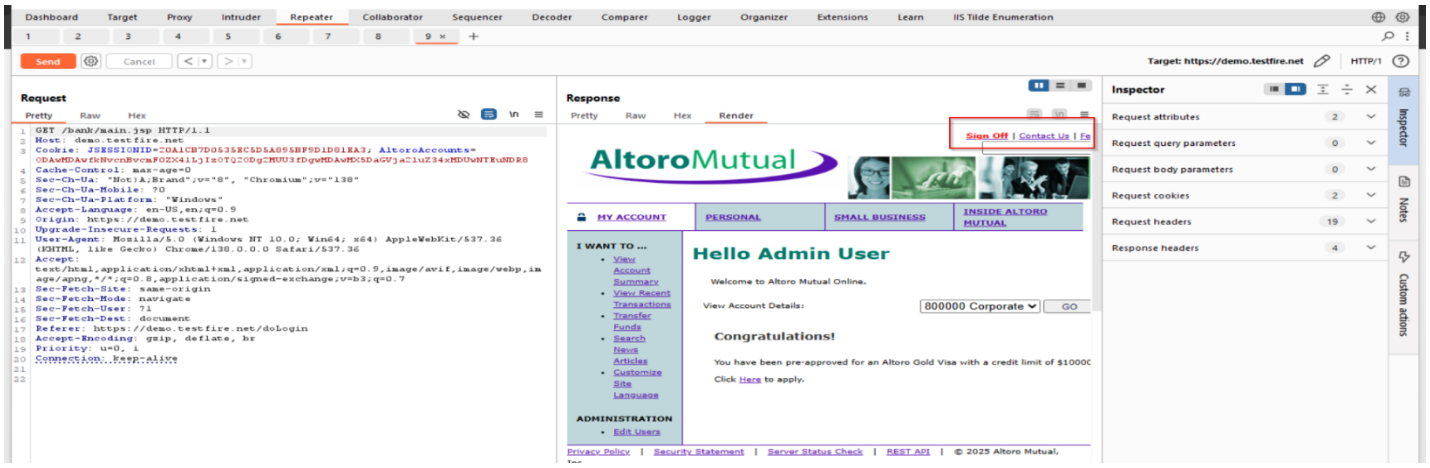
Evidence:



Here logged in as normal user



Added some injection payloads in username and tries to login



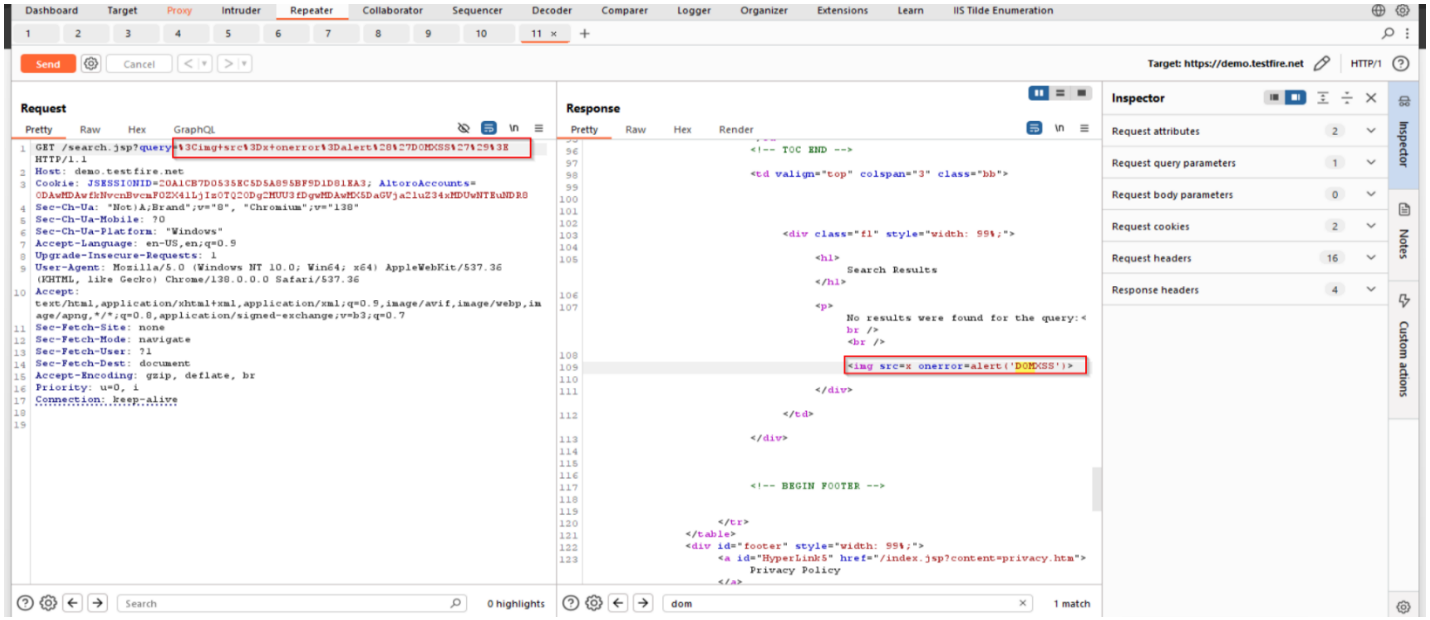
Burp Suite output showing repeated login attempts and successful access to a valid user account without any rate-limiting or lockout in place.

17) DOM XSS:

Description:

DOM-based XSS happens when JavaScript reads untrusted data from the DOM (like a URL fragment) and inserts it directly into the webpage without validation.

Evidence:



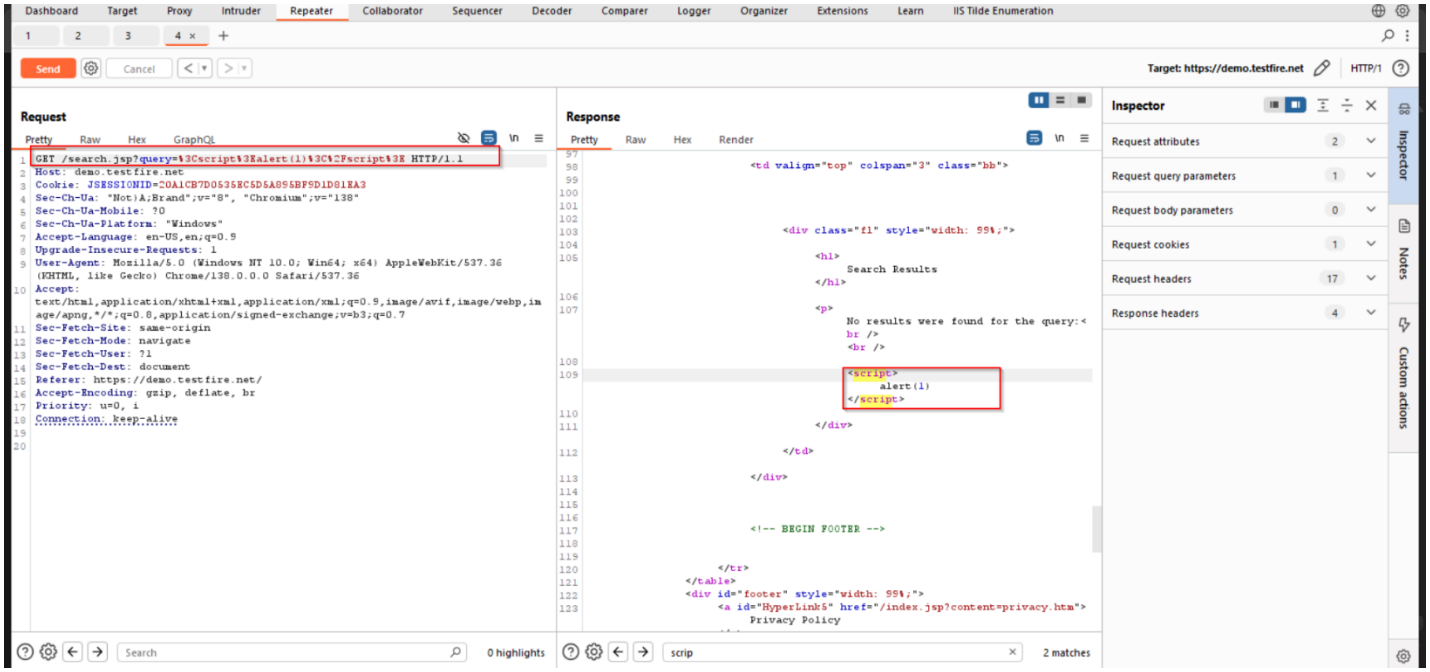
DOM evidence showing script execution from URL fragment.

18) Reflected XSS:

Description:

The application reflects unsanitized user input directly in the browser without proper output encoding or validation. This enables attackers to inject malicious scripts that execute in the victim's browser

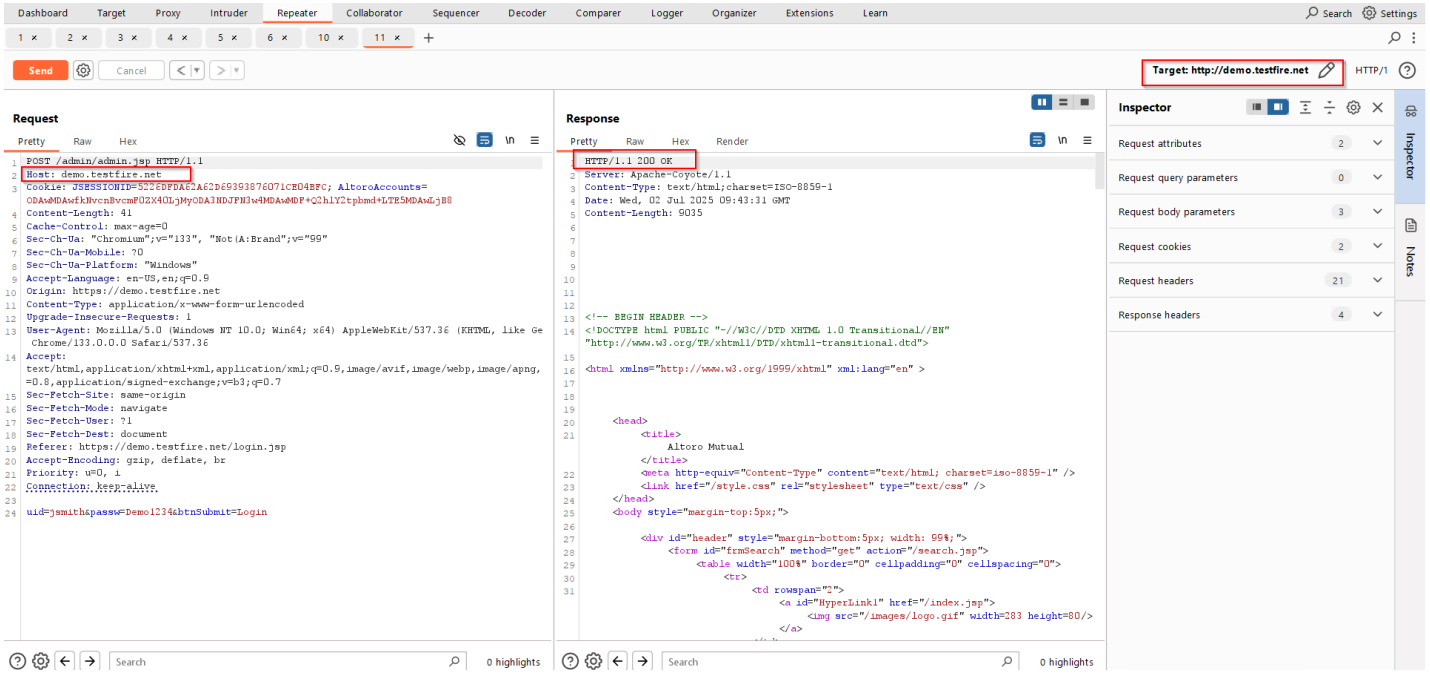
Evidence:



Burp Suite request/response and browser view showing script execution

19) Insecure HTTP:

Evidence:



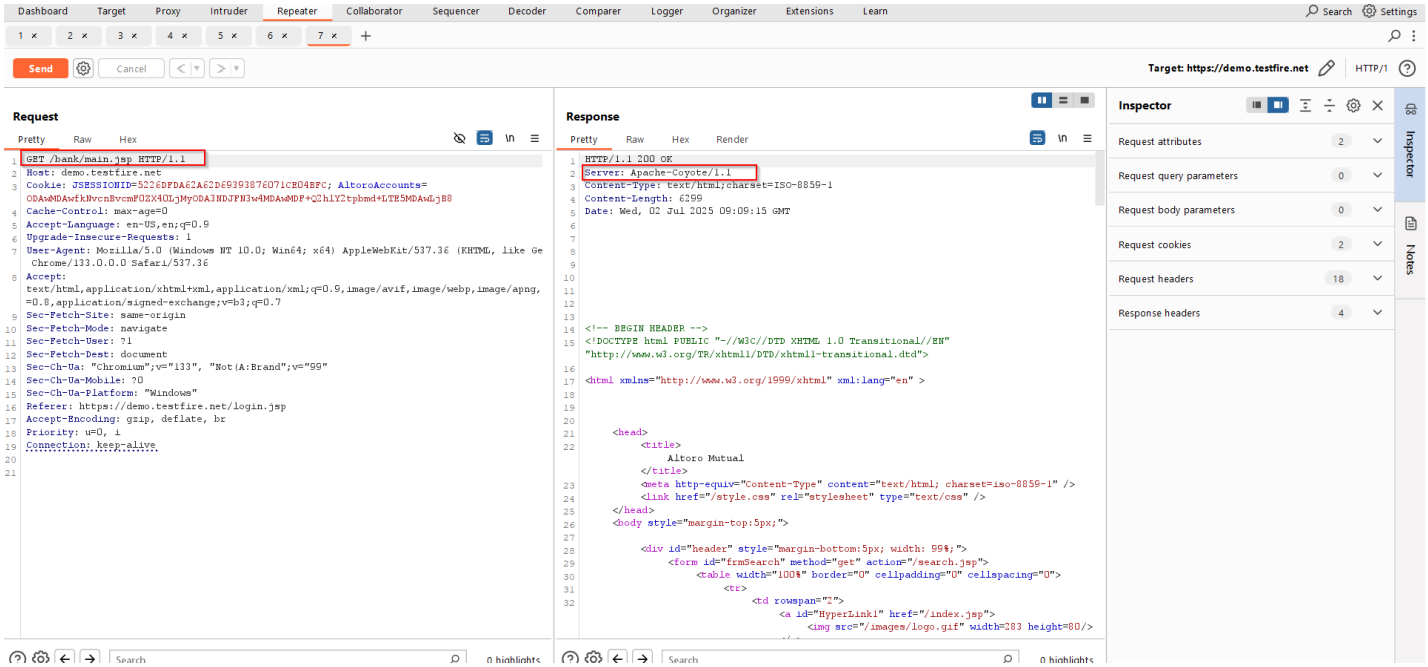
The request is going through insecure http

20)Verbose server banner disclosure:

Description:

The server reveals the version and software name (Apache-Coyote/1.1) in its HTTP headers. Such disclosures aid attackers in identifying the underlying application stack and matching it against known vulnerabilities.

Evidence:



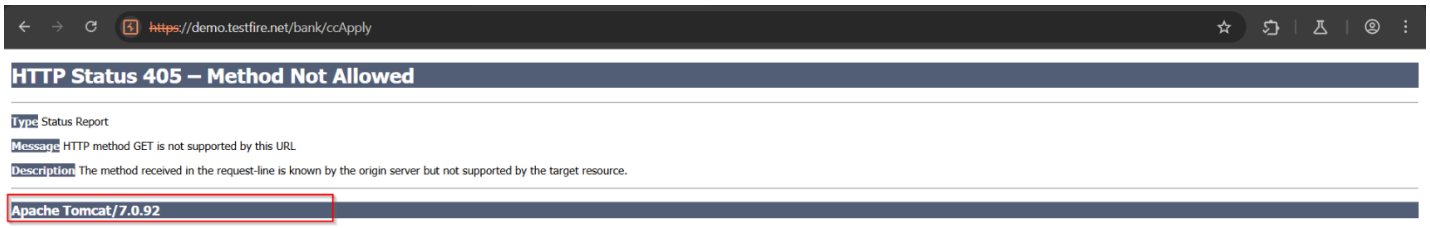
HTTP response headers showing Server: Apache-Coyote/1.1 which is disclosure of server

21)Server Version Disclosure via Directory Enumeration

Description:

A directory enumeration attack revealed default folders or pages that exposed the exact version of Apache Tomcat (7.0.92). Default examples and documentation pages often disclose sensitive server metadata.

Evidence:



Burp Suite response or browser view showing Tomcat default page with version.

22)Missing security headers:

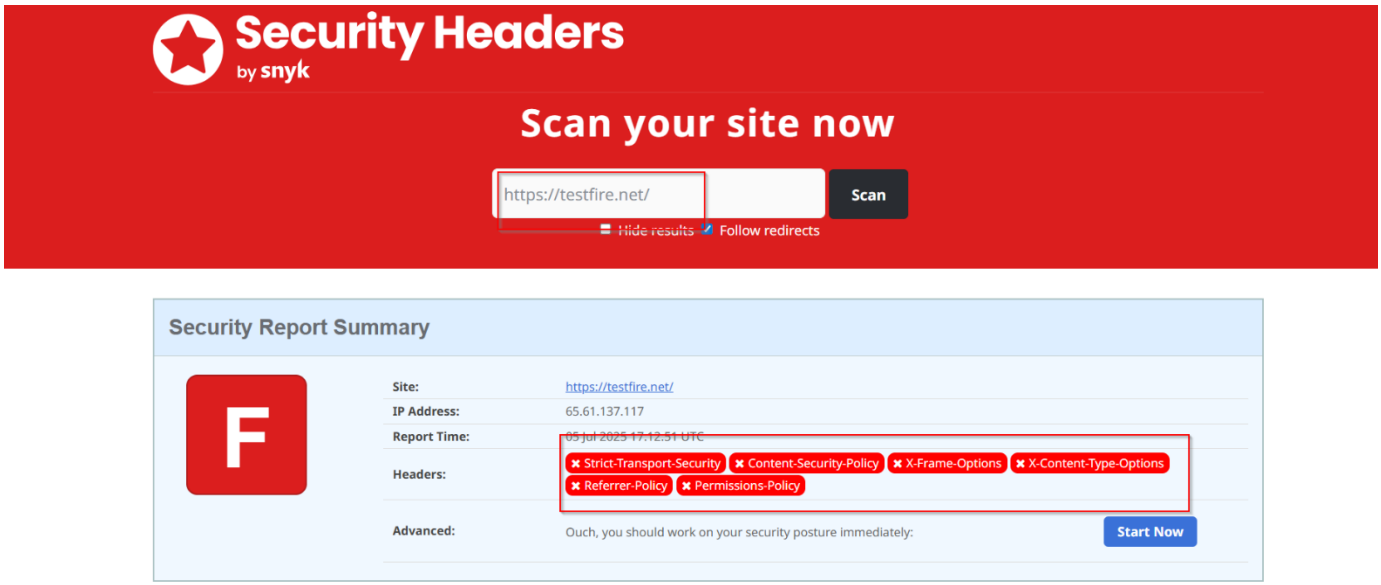
Description:

The application fails to implement critical HTTP response headers that enhance browser security. Missing headers include:

- Content-Security-Policy
- X-Frame-Options

- X-Content-Type-Options
- Referrer-Policy
- Strict-Transport-Security

These headers provide safeguards against XSS, clickjacking, and information disclosure.

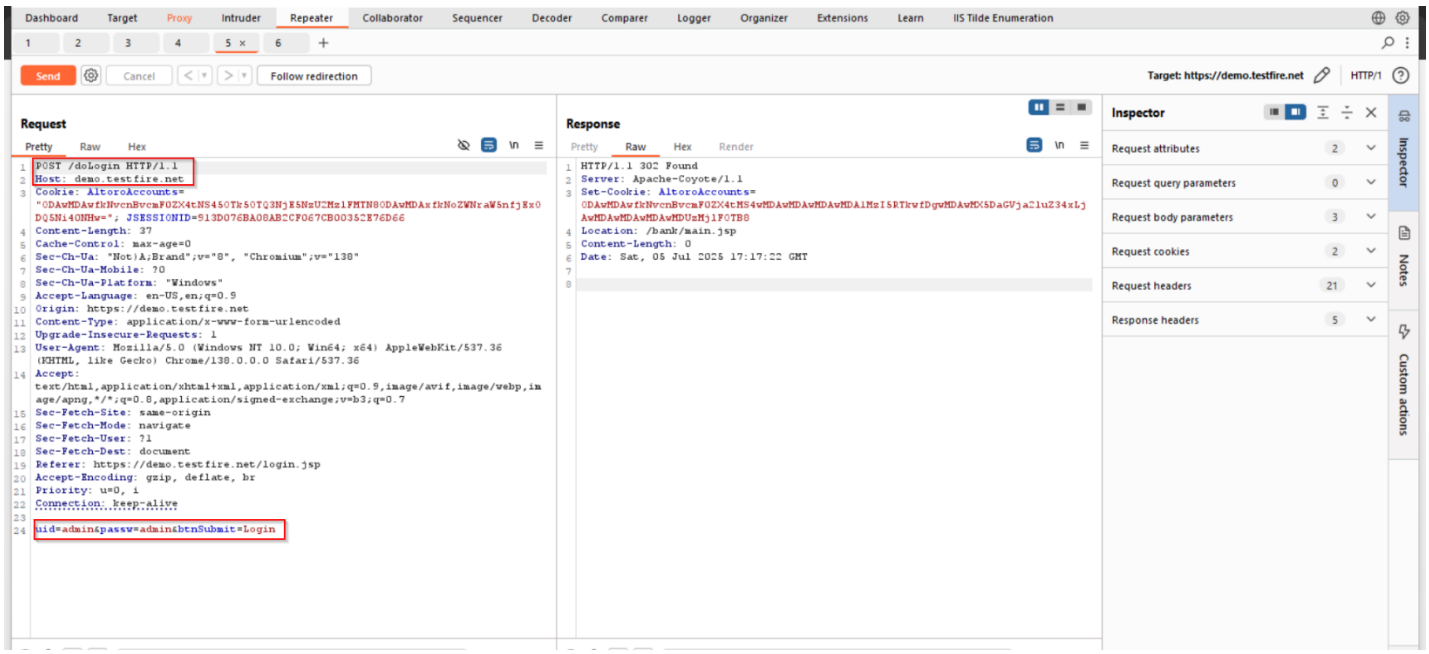


The above screenshot shows the missing security headers.

23) Weak password policy

Description:

The application permits weak or easily guessable passwords such as 123456 or password. There is no enforced policy for password strength, complexity, or reuse.

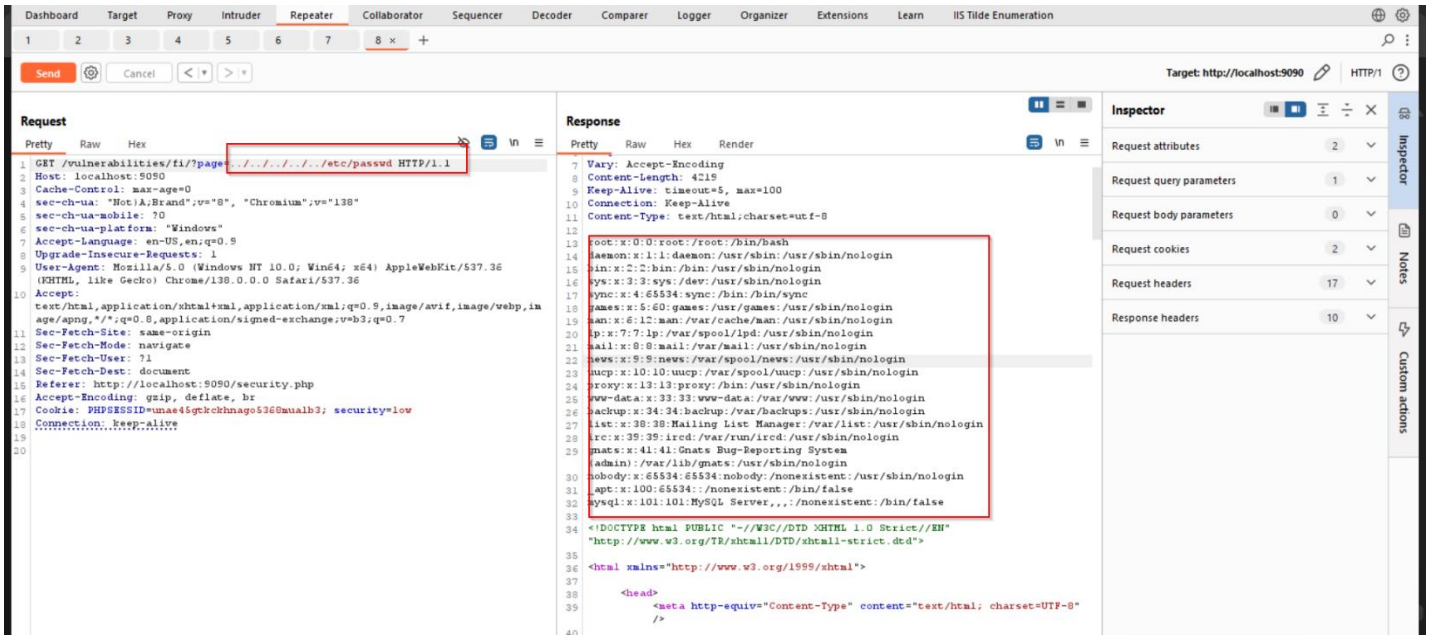


The application is accepting weak passwords.

24) Local file inclusion:

Description:

The application dynamically includes a file based on a user-supplied parameter without proper validation. This allows traversal and inclusion of arbitrary local files on the server.

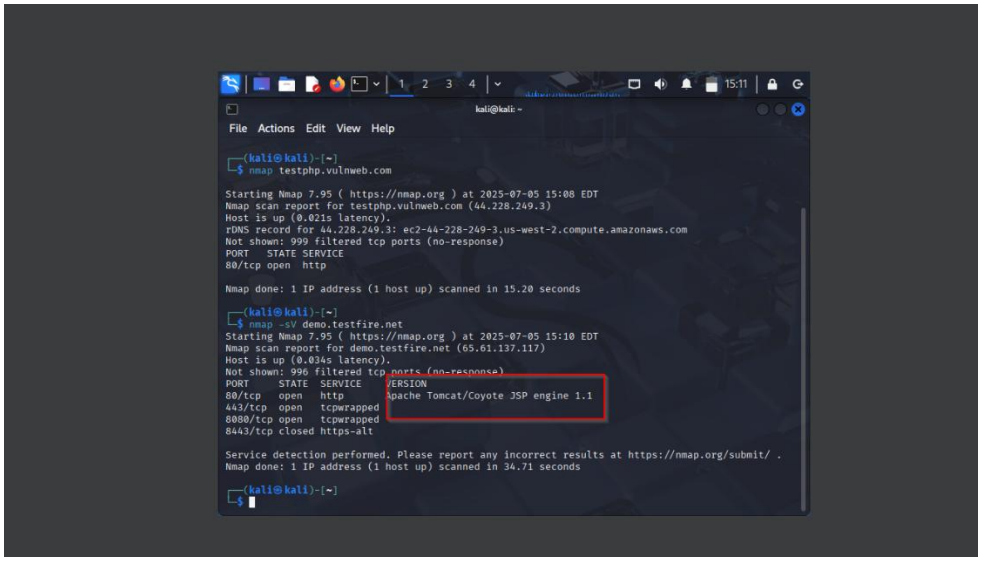


Burp Suite response showing part of the /etc/passwd file or error logs revealing sensitive file contents.

25)Vulnerable software:

Description:

The server is running an outdated version of JSP Engine (v11.1), which may be vulnerable to known exploits including Denial of Service (DoS) or code disclosure vulnerabilities**.**

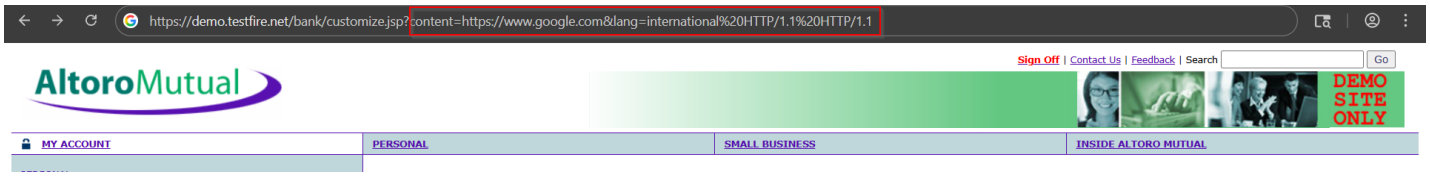


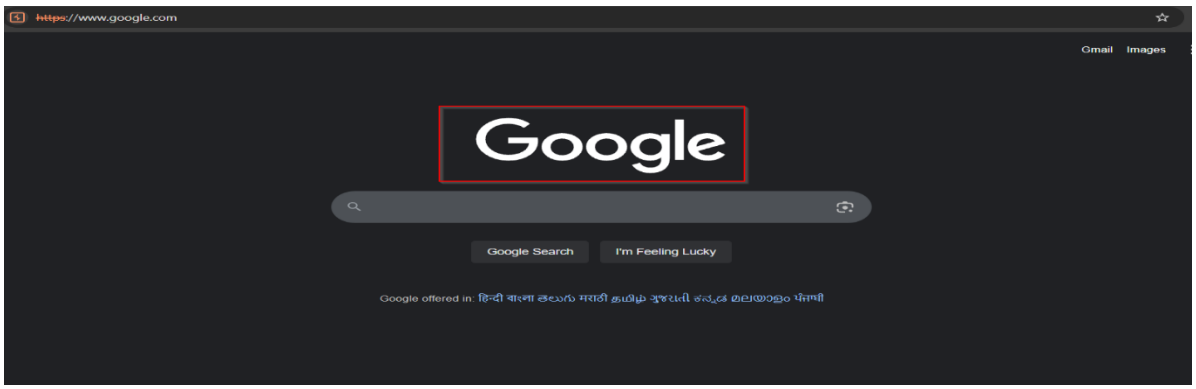
The screenshot indicates the use of vulnerable JSP Engine 11.1.

26)URL redirection:

Description:

The application redirects users to external URLs without validation, enabling open redirection attacks that may lead to phishing or malware redirection



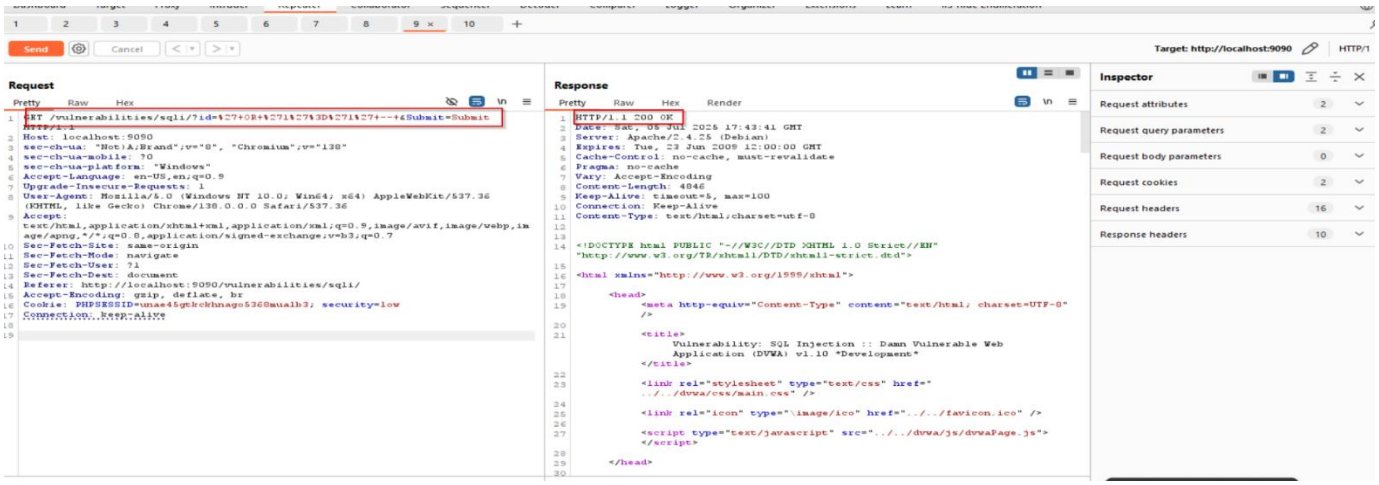


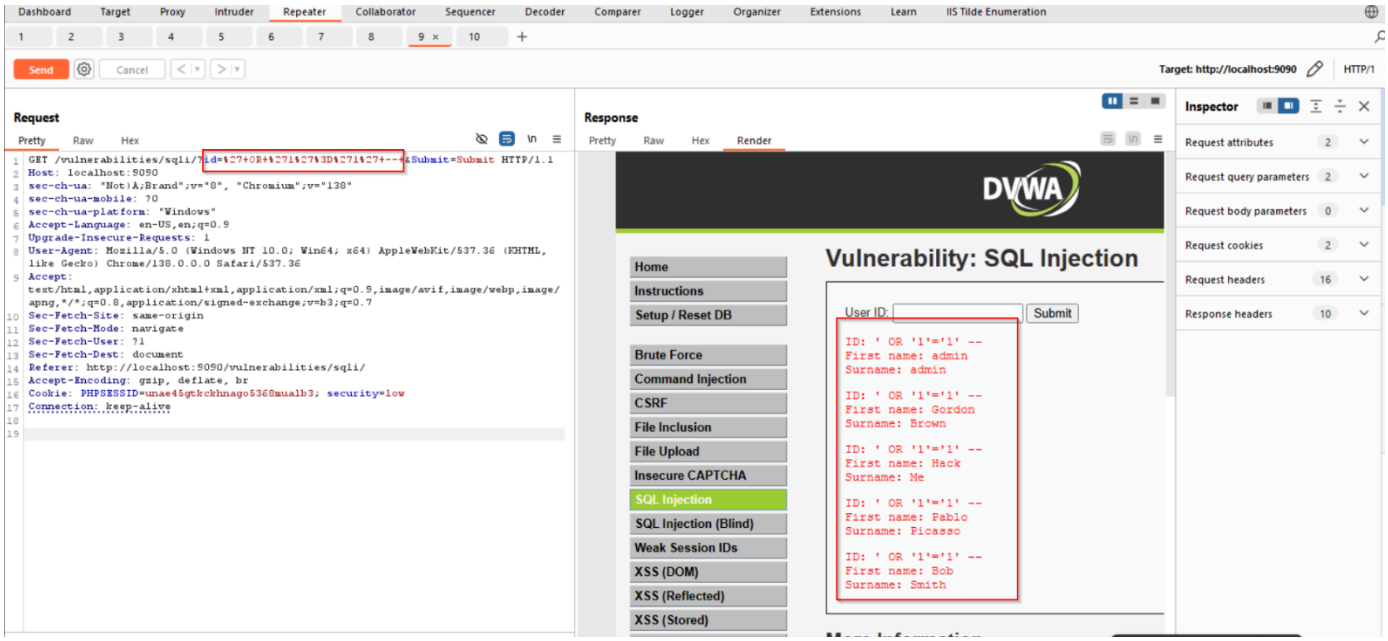
The screenshot displays a crafted URL which redirects the user to a third-party domain.

27)SQL injection attack:

Description:

SQL injection was successful, allowing attackers to manipulate backend queries, potentially accessing sensitive data or bypassing authentication.

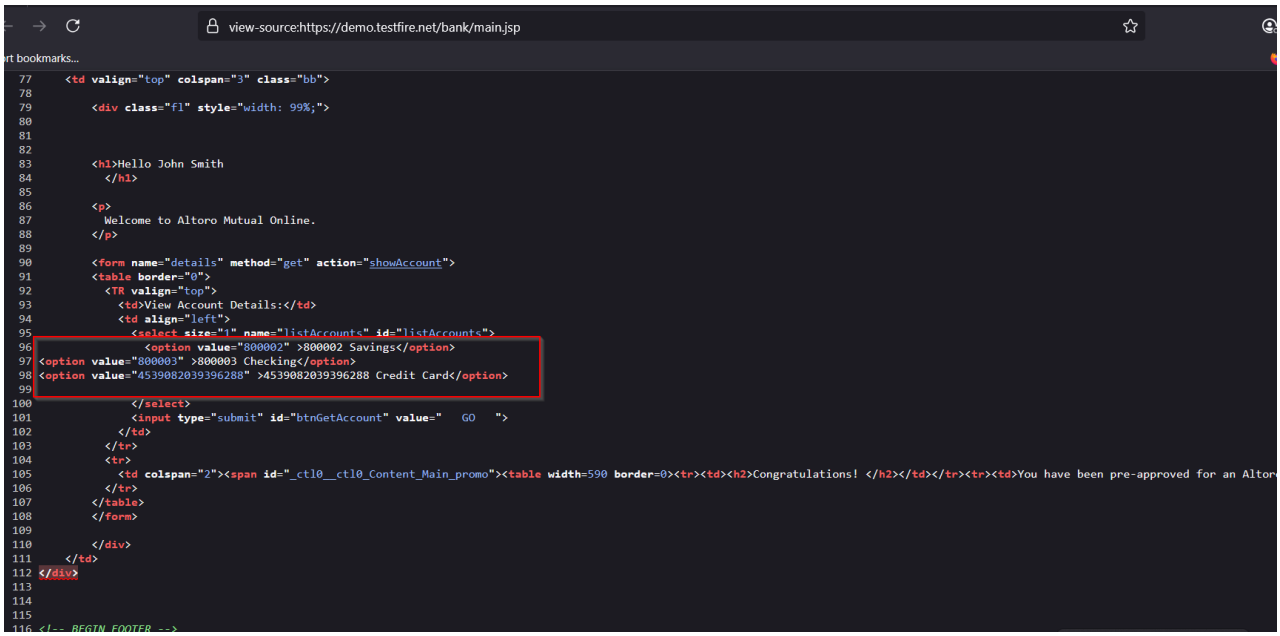




28)Source code disclosure:

Description:

Source code disclosure exposes confidential logic and sensitive information due to improper file access or misconfiguration.

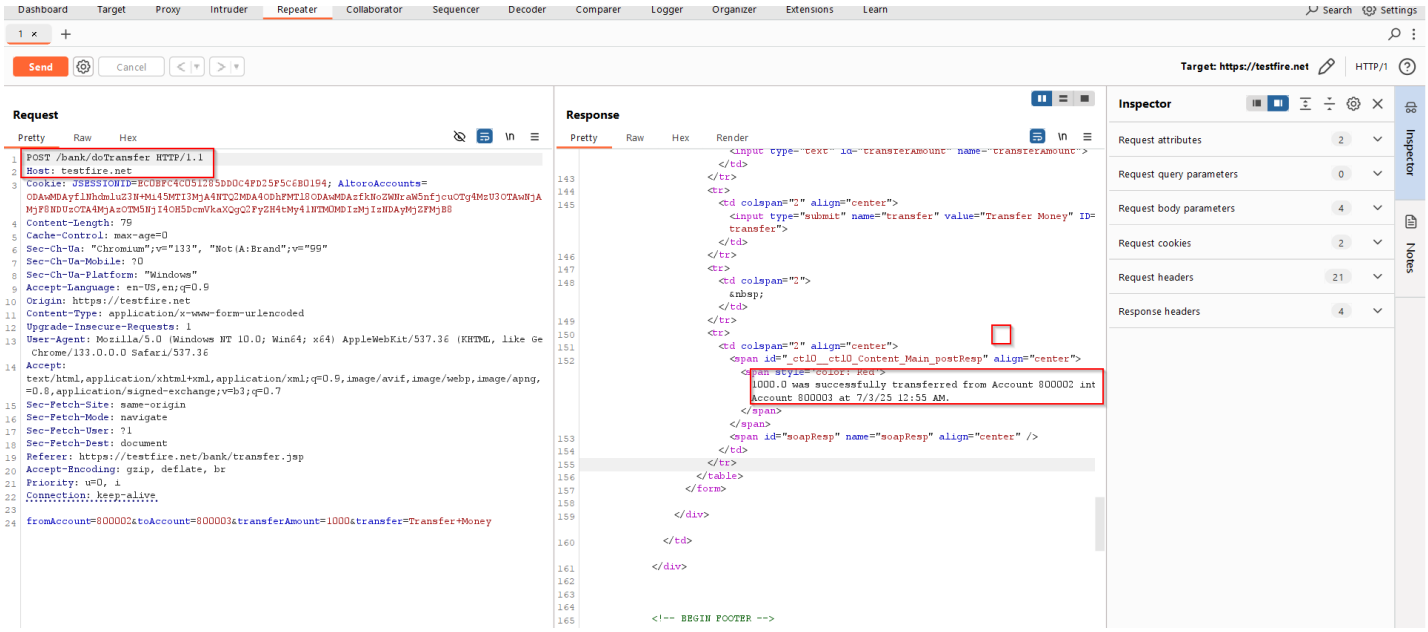


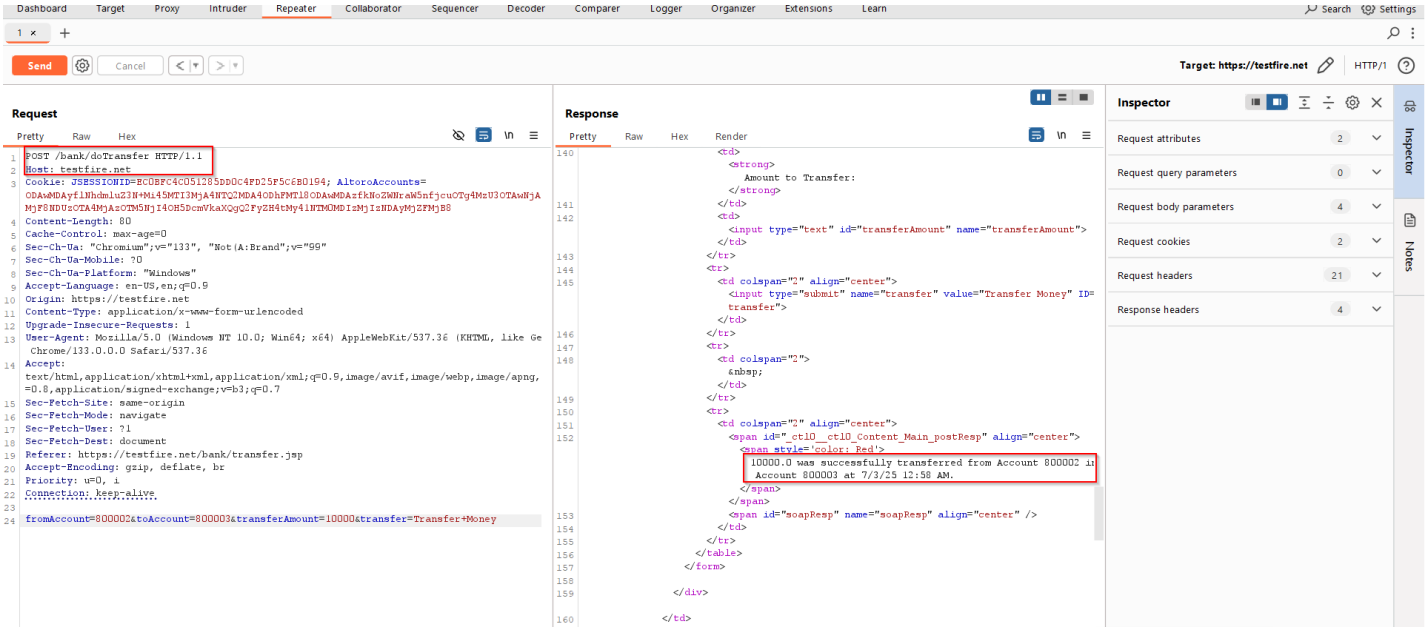
The screenshot reveals source code with sensitive data such as credit card numbers and account details visible in the page response.

29)Parameter tampering:

Description:

Client-side parameters were altered to manipulate business logic, such as changing the amount in a transaction — indicating parameter tampering.



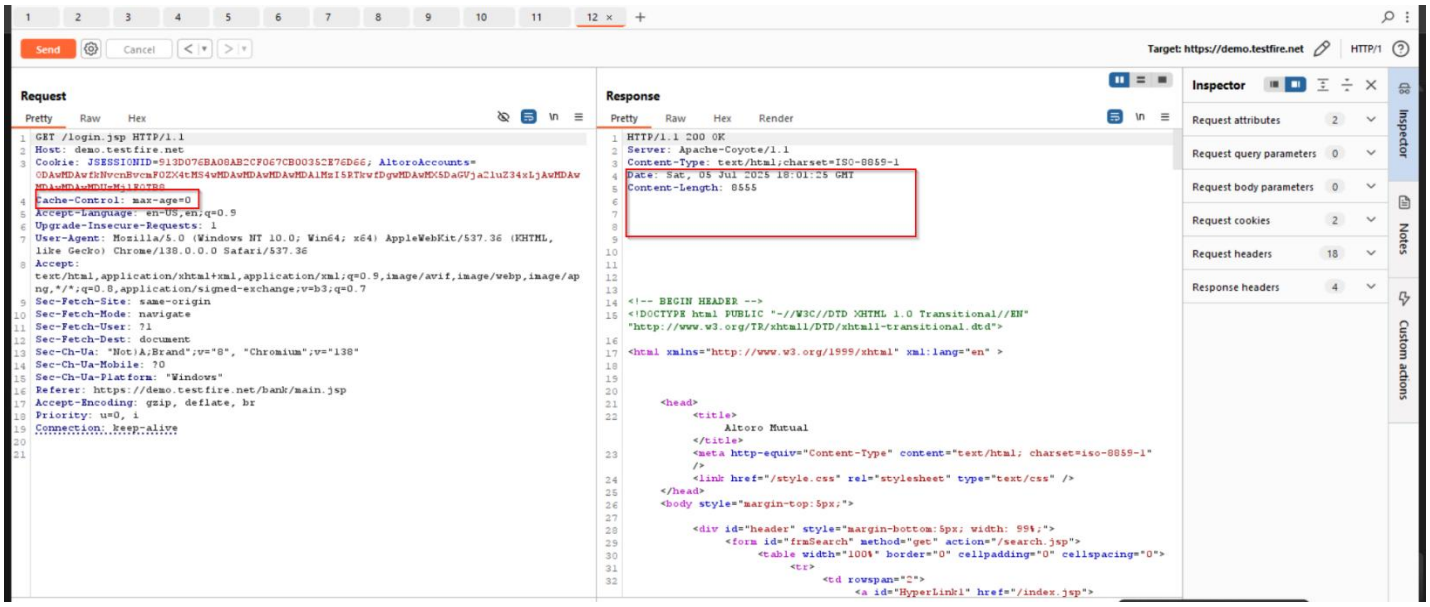


The screenshot shows a modified amount parameter in the transaction request resulting in unauthorized payment processing.

30)Cache control is missing:

Description:

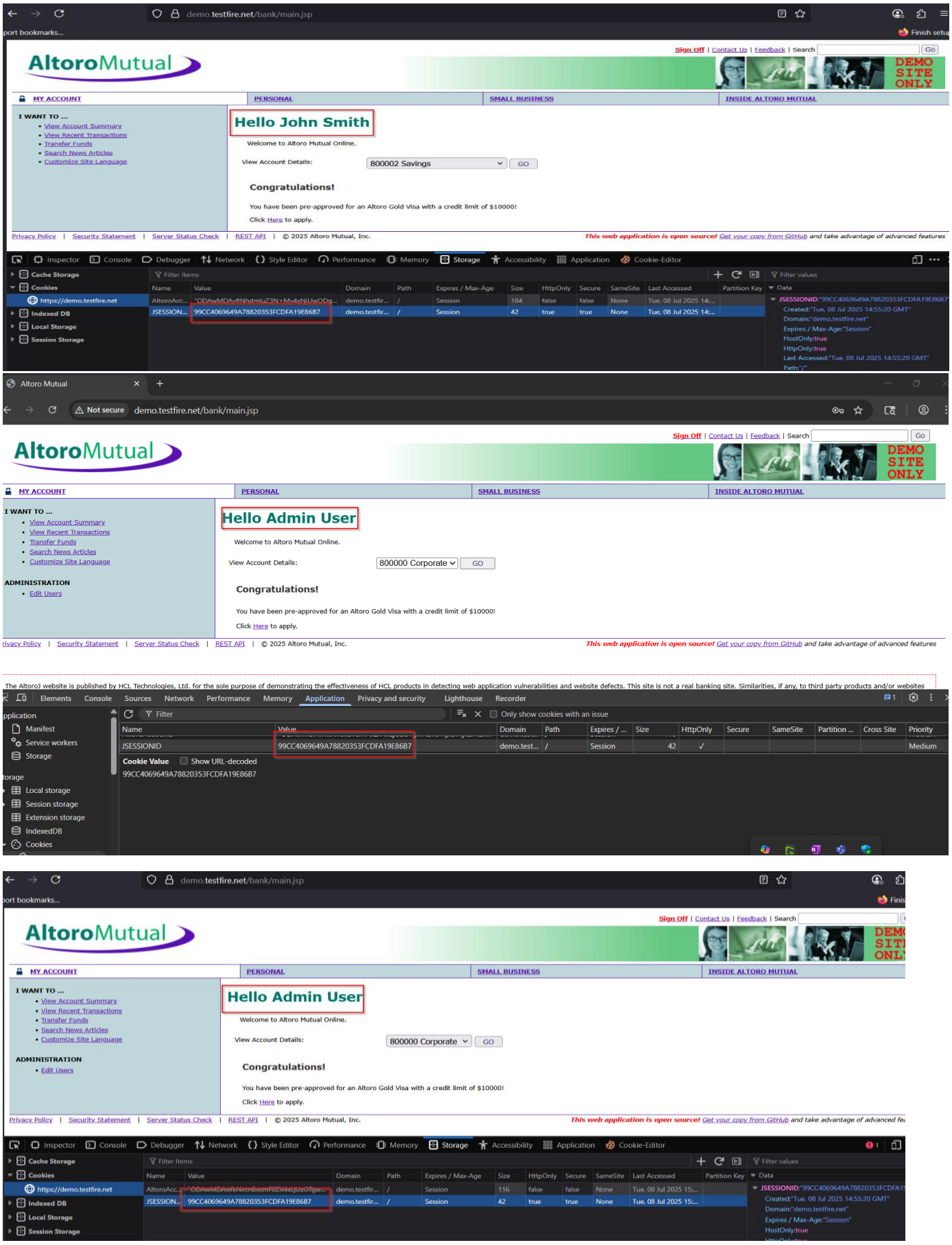
Absence of proper cache control headers can cause sensitive pages (like logout or profile) to be stored in the browser cache, leading to potential data leakage.



31)Session hijacking:

Description:

The session ID was intercepted or reused, allowing attackers to impersonate legitimate users and access their accounts.



The screenshot shows session cookie reuse or token exposure leading to successful hijacking of another user's session.