Application Name:	Altro Mutual
Version Number:	
Assessment Type	Dynamic Application Security Assessment
Report Published Date:	13/4/2026
Scheduling ID	

1 INTRODUCTION AND BACKGROUND 2

2 OBJECTIVE 2

3 APPROACH 2

4 EXECUTIVE REPORT 3

5 TECHNICAL TESTING RESULTS 5

1. Broken Access Control Allowing Unauthorized Account Data Access 5
2. SQL Injection in Login Endpoint Allowing Authentication Bypass 7
3. Plaintext Transmission of Password in Authentication Request 8
4. Unrestricted Login Attempts (Brute Force Vulnerability) 10
5. Business Logic Flaw in Fund Transfer (Unauthorized Parameter Manipulation & Duplicate Transaction) 11
6. Reflected Cross-Site Scripting (XSS) 13
7. Reflected Cross-Site Scripting (XSS) via HTML Injection in content Parameter 14
8. Sensitive Information Disclosure via Verbose Error Message in content Parameter 15
9. Use of Outdated Apache Tomcat (Apache-Coyote/1.1) with Known CVE Vulnerabilities 16
10. Insufficient Logging and Monitoring of Authentication Attempts 18
11. Multiple Sensitive Open Ports Exposing Internal Services 19
12. Weak Session Management – Session ID Not Invalidated After Logout 20
13. Clickjacking Vulnerability due to Missing Frame Protection 21
14. Open Redirect via URL Parameter 22
15. Cross-Site Request Forgery (CSRF) in Login Functionality 23
16. Missing Security Headers Leading to Increased Attack Surface 24

6 NEXT STEPS 26

7 VULNERABILITY REMEDIATION TIMELINE 26

8 CONTACTS AND DEMOGRAPHICS 27

Assessment Contacts 27

9 REPORT PROPERTIES 27

CLASSIFICATION 27

1 Introduction and Background

Application security assessment was performed on Altoro Mutual Online Banking Application. The assessment period was from 6 Apr'26 to 13 Apr'26.

The Altoro Mutual application is a publicly accessible demo online banking platform used for security testing and training purposes. The application simulates real-world banking functionalities such as account management, fund transfers, user authentication, and customer interactions. The system processes user-related information such as account details, transaction data, login credentials, and feedback information.

2 Objective

- Verify the implementation of security controls within the application in alignment with OWASP Top 10 (2025) and industry best practices.
- Identify, prioritize, and document security vulnerabilities discovered during testing.
- Provide recommendations for remediation to improve the overall security posture of the application.

3 Approach

- Performed application reconnaissance and mapped available endpoints and functionalities.
- Conducted Dynamic Application Security Testing (DAST) on Altoro Mutual hosted at https://altoro.testfire.net.
- Utilized Burp Suite Community Edition for intercepting, modifying, and analyzing HTTP requests and responses.
- Performed manual vulnerability testing aligned with OWASP Top 10 (2025) categories.
- Tested for vulnerabilities such as Injection (XSS), Broken Access Control, CSRF, Security Misconfiguration, Business Logic Flaws, and Session Management issues.
- Analyzed findings based on severity and potential impact.
- Documented vulnerabilities along with proof of concept and remediation recommendations.

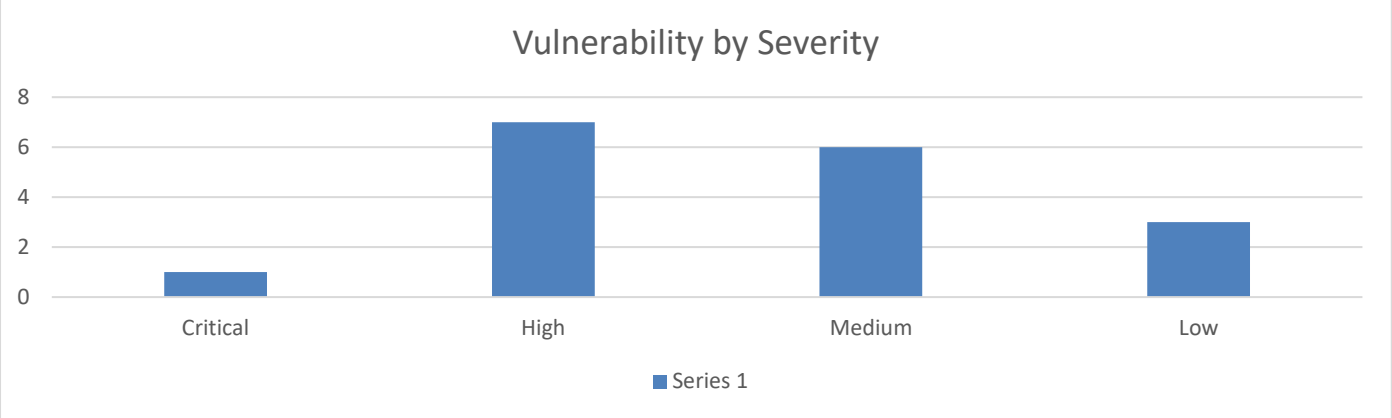
Constraints:

- Testing was limited to Burp Suite Community Edition capabilities.
- No automated authenticated scanning tools (e.g., Burp Pro, WebInspect) were used.
- Testing was performed only on the publicly accessible demo environment.

4 Executive Report

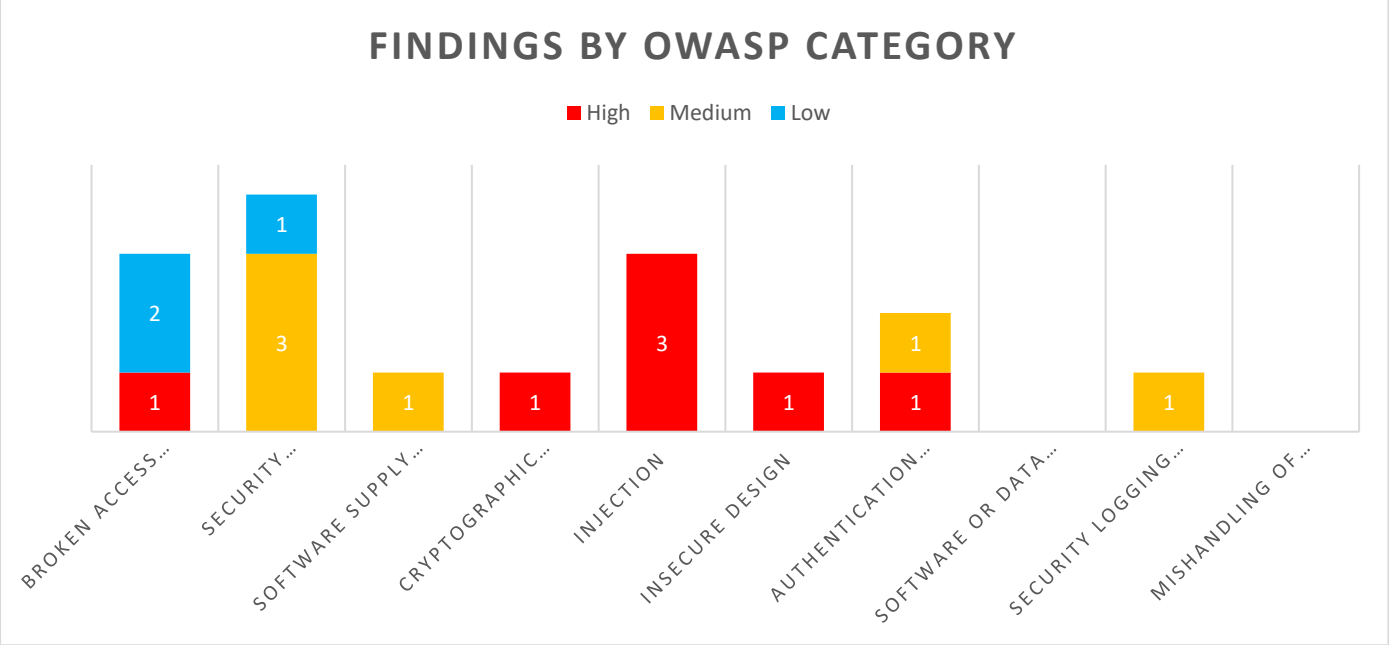
Executive summary of vulnerabilities identified during Dynamic Application Assessment of **ALTRO MUTUAL**

Findings by Vulnerability Severity



Absence of a bar chart represents there are no vulnerabilities for that severity level

Findings by OWASP Category



List of Vulnerabilities

The full list of findings can be found in the Technical Testing Results section below.

Sl. No.	eGRC Finding ID	Finding	Severity	# instances identified	Finding status	Target date for remediation
1		Broken Access Control	High	Web App		
2		Injection	Critical	Web App		
3		Cryptographic Failure	High	Web App		
4		Authentication Failures	High	Web App		
5		Insecure Design	High	Web App		
6		Injection	High	Web App		
7		Injection	High	Web App		
8		Security Misconfiguration	Medium	Web App		
9		Software Supply Chain Failures	Medium	Web App		
10		Security Logging and Monitoring Failures	Medium	Web App		

11		Security Misconfiguration	Medium	Web App		
12		Authentication Failures	Medium	Web App		
13		Security Misconfiguration	Medium	Web App		
14		Broken Access Control	Low	Web App		
15		Security Misconfiguration	Low	Web App		
16		Broken Access Control	Low	Web App		

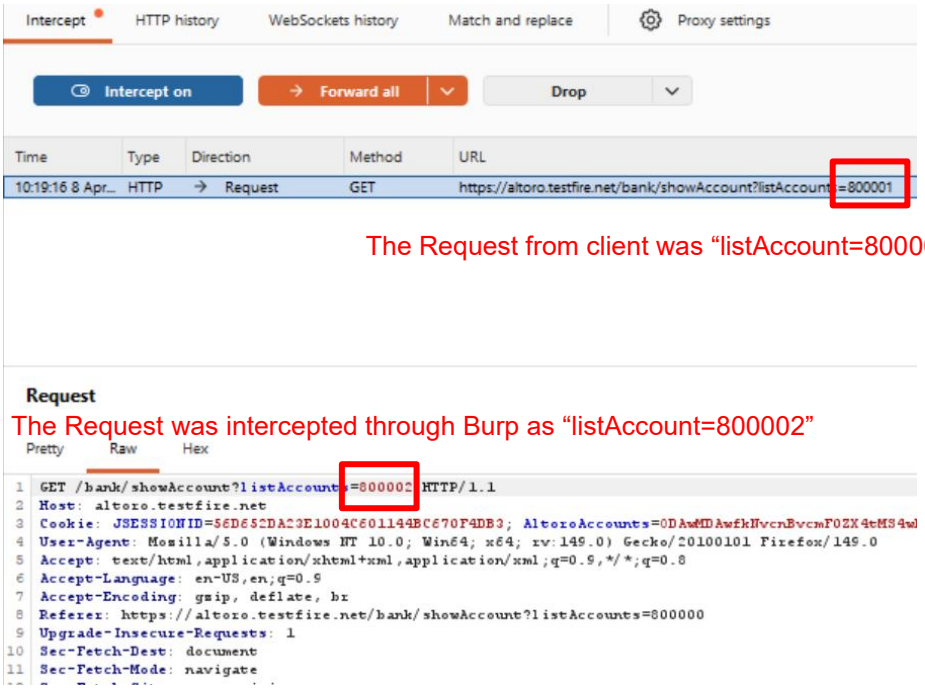
Highly Confidential

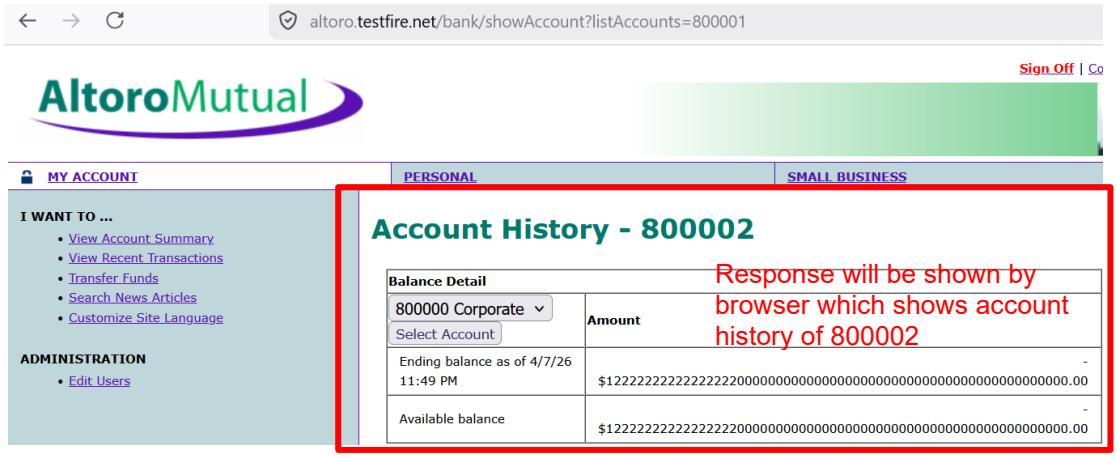
5 Technical Testing Results

1. Broken Access Control Allowing Unauthorized Account Data Access

Risk Rating	High
Category	Broken Access Control
Description	The application exposes a direct object reference via the parameter: *`/bank/showAccount?listAccounts=800001`* By modifying the value to: *`listAccounts=800002`* we were able to access another user's account data without authorization checks.
Impact	• Unauthorized access to other users' financial data • Exposure of sensitive information (balances, transactions) • Potential for fraud or account manipulation • Loss of confidentiality and integrity • Regulatory/legal consequences (especially for financial systems) In real-world scenarios, this could lead to: • Account takeover chaining • Data exfiltration at scale (enumerating account IDs)
Recommendation	1. Enforce Server-Side Authorization 2. Use Indirect References 3. Implement Access Control Checks Everywhere 4. Add Monitoring & Rate Limiting 5. Security Testing
Finding Id	
Target Date	
Affected URLs	

Evidences

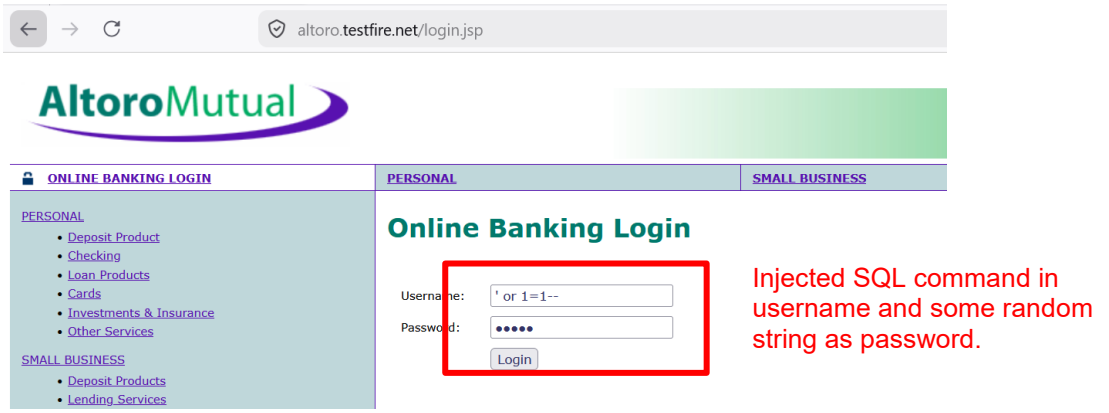




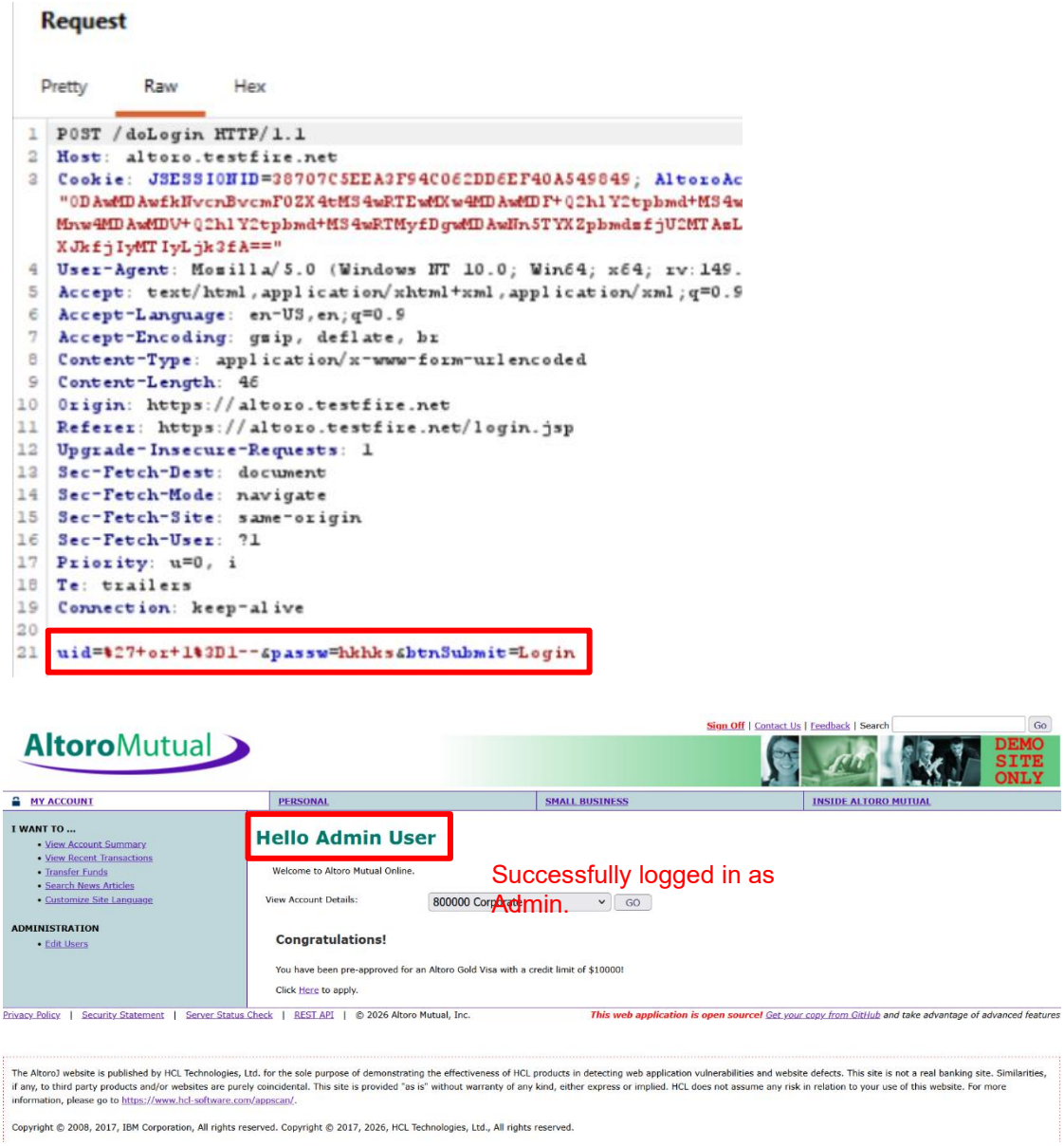
Management response	djaGDJGjfakj

Highly Confidential

2. *SQL Injection in Login Endpoint Allowing Authentication Bypass*

Risk Rating	Critical
Category	Injection
Description	The application is vulnerable to SQL Injection in the login functionality (/doLogin endpoint). By injecting SQL payload into the username field: `' or 1=1--` The backend query logic is altered, causing the authentication check to always evaluate as true. This allows login without valid credentials. This occurs because: • User input is directly embedded into SQL queries • No input validation or parameterization is enforced
Impact	• Authentication bypass (login without credentials) • Full account takeover • Unauthorized access to sensitive financial data • Potential database compromise (depending on exploitation level) • Data extraction, modification, or deletion In real-world scenarios, this can escalate to: • Dumping entire user databases • Privilege escalation (e.g., admin access)
Recommendation	1. Use Parameterized Queries (Prepared Statements) 2. Input Validation & Sanitization 3. Use ORM / Safe Database Libraries 4. Least Privilege Principle 5. Implement Web Application Firewall (WAF) 6. Error Handling
Finding Id	
Target Date	
Affected URLs	
Evidences	

Highly Confidential



Management response	djaGDJGjfakj

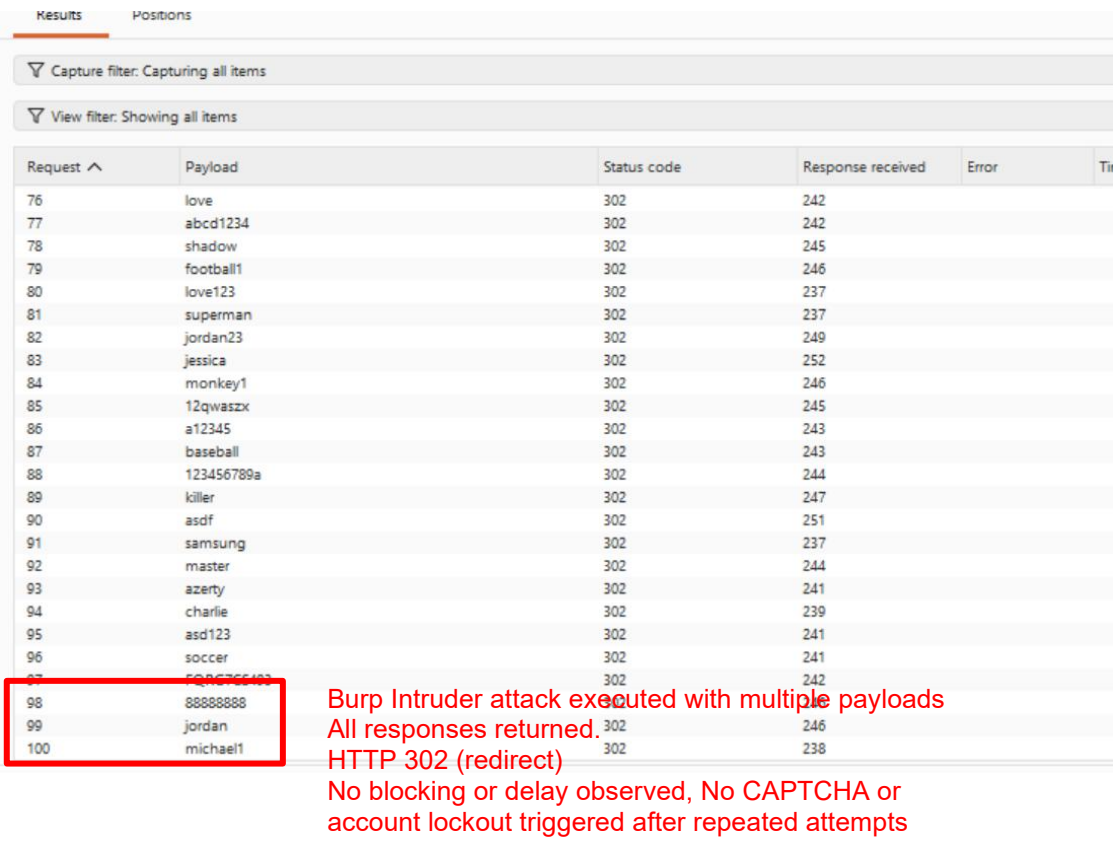
3. Plaintext Transmission of Password in Authentication Request

Risk Rating	High
Category	Cryptographic Failures (Sensitive Data Exposure)
Description	The application transmits user credentials (password) in **plaintext** within the HTTP request. From the captured request (via proxy), the password parameter is visible directly: `password=admin` This indicates that: • The password is **not encrypted or protected at the application layer** • Sensitive data is exposed in transit and potentially in logs/intermediary systems.

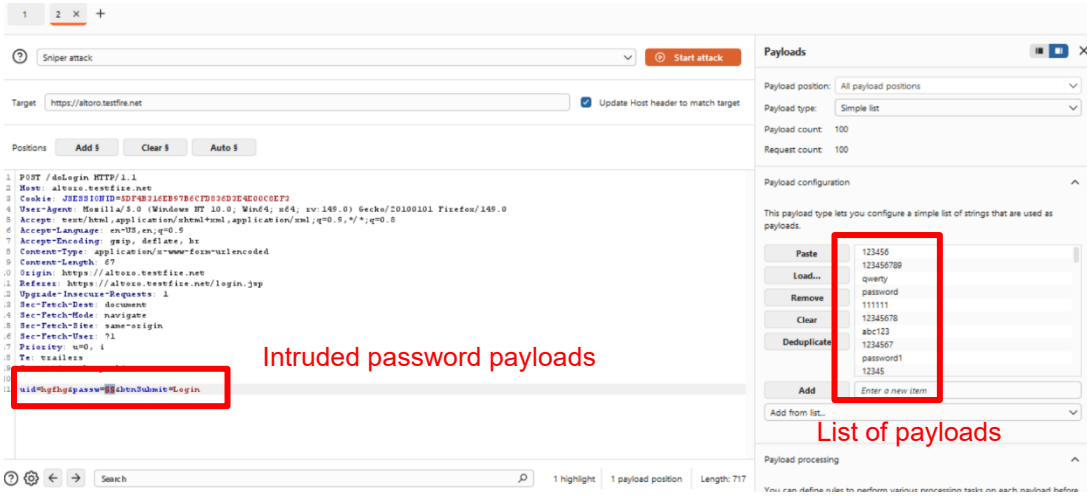
	If transmitted over an insecure channel (HTTP), this becomes trivial to intercept. Even over HTTPS, improper handling (e.g., logging plaintext passwords) still poses a serious risk.
Impact	• Exposure of user credentials to attackers • Account takeover • Credential reuse attacks on other platforms • Sensitive data leakage via logs, proxies, or monitoring tools • Compliance violations (e.g., GDPR, PCI-DSS) In real-world scenarios, this can escalate to: • Dumping entire user databases • Privilege escalation (e.g., admin access)
Recommendation	1. Use Parameterized Queries (Prepared Statements) 2. Input Validation & Sanitization 3. Use ORM / Safe Database Libraries 4. Least Privilege Principle 5. Implement Web Application Firewall (WAF) 6. Error Handling
Finding Id	
Target Date	
Affected URLs	
Evidences	**Request** Pretty Raw Hex POST /doLogin HTTP/1.1 Host: altoro.testfire.net Cookie: JSESSIONID=ACC5D09A76CE068BD16A655735ECA687; AltoroAccounts= "0DAwMDAwfkNvcnBvcmF0ZX4tMS4wRTg4fDgwMDAwMX5DaGVja2luZ34xMzMx0TI3LjklfDgwMDAwMn5TYXZpbmdzfi0xLjBF MzA4fDgwMDAwM35DaGVja2luZ34xLjBFMzA4fDgwMDAwNH5TYXZpbmdzfi05MDQzMzQuMHw4MDAwMDV+Q2hlY2tpbmd+MjQuM Hw4MDAwMDZ+U2F2aW5nc34t0S450Tk0MDg5N0U4fDgwMDAwN35DaGVja2luZ34xNTAuMHw0NTM5MDgyMDM5Mmk2Mjg4fkNyZW RpdCBDYXJkfjIuMjQyNDQyMjQyNDQ0NDgs0DRFMTh8NDQ4NTk4MzM1NjI0MjIxN35DcmVkaXQgQ2FyZH4xMDAwMC45N3w=" User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:149.0) Gecko/20100101 Firefox/149.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.9 Accept-Encoding: gzip, deflate, br Content-Type: application/x-www-form-urlencoded Content-Length: 37 Origin: https://altoro.testfire.net Referer: https://altoro.testfire.net/login.jsp Upgrade-Insecure-Requests: 1 Sec-Fetch-Dest: document Sec-Fetch-Mode: navigate Sec-Fetch-Site: same-origin Sec-Fetch-User: ?1 Priority: u=0, i Te: trailers Connection: keep-alive uid=admin&passw=admin&btnSubmit=Login Password is not encrypted and sent plain text through the POST request.
Management response	djaGDJGjfakj

Highly Confidential

4. Unrestricted Login Attempts (Brute Force Vulnerability)

Risk Rating	**High**
Category	Authentication Failures
Description	The application does not implement any rate limiting or account lockout mechanism on the login functionality. An attacker can perform unlimited login attempts using automated tools such as Burp Suite Intruder. This allows attackers to brute-force user credentials and gain unauthorized access.
Impact	• Account compromise • Unauthorized access to sensitive user data • Potential financial fraud (banking app context) • Credential stuffing attacks possible
Recommendation	1. Implement **account lockout** after 5 failed attempts 2. Add **CAPTCHA** after repeated failures 3. Enforce **rate limiting** (e.g., max 5 requests/minute) 4. Implement **multi-factor authentication (MFA)** 5. Monitor and alert on suspicious login activity
Finding Id	
Target Date	
Affected URLs	
Evidences	

Highly Confidential

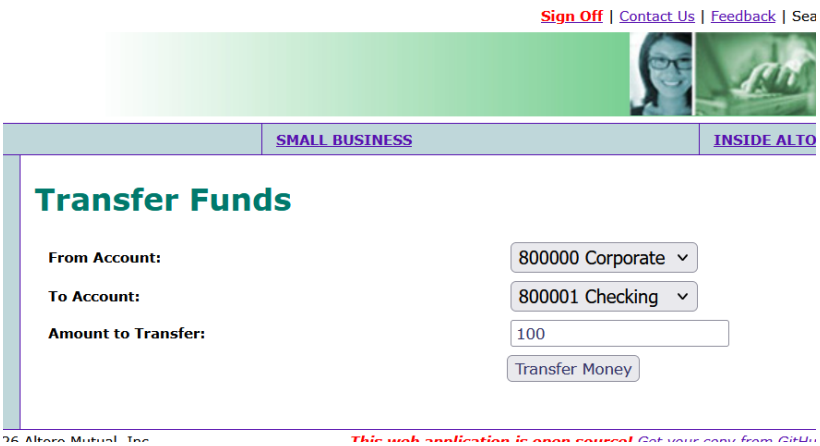


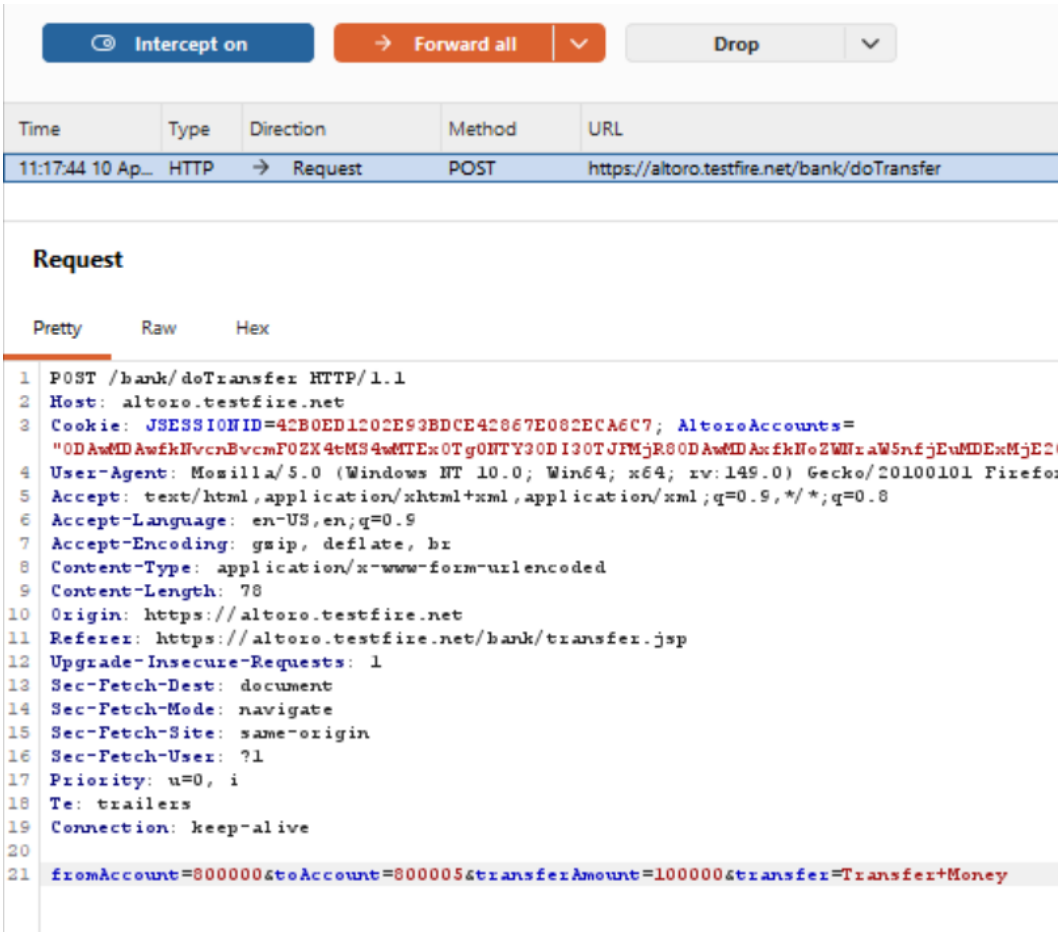
Management response	djaGDJGjfakj

5. Business Logic Flaw in Fund Transfer (Unauthorized Parameter Manipulation & Duplicate Transaction)

Risk Rating	High
Category	Insecure Design
Description	The fund transfer functionality does not properly validate user input or enforce business rules. By intercepting and modifying the request, an attacker can change critical parameters such as fromAccount, toAccount, and transferAmount. Additionally, the same transaction request can be replayed multiple times, resulting in duplicate processing of the transfer. This indicates a lack of server-side validation and absence of controls to prevent transaction replay or duplicate submissions.
Impact	• Unauthorized transfer of funds between accounts • Manipulation of transaction amounts • Duplicate transactions leading to financial loss • Compromise of data integrity and trust in the system
Recommendation	1. Implement strict server-side validation for all transaction parameters 2. Ensure users can only transfer funds from accounts they own 3. Add transaction integrity controls (e.g., unique transaction IDs, anti-replay tokens) 4. Prevent duplicate submissions using mechanisms like one-time tokens or request validation 5. Enforce proper business rules (e.g., valid account pairing, positive amount checks)
Finding Id	
Target Date	
Affected URLs	

Evidences

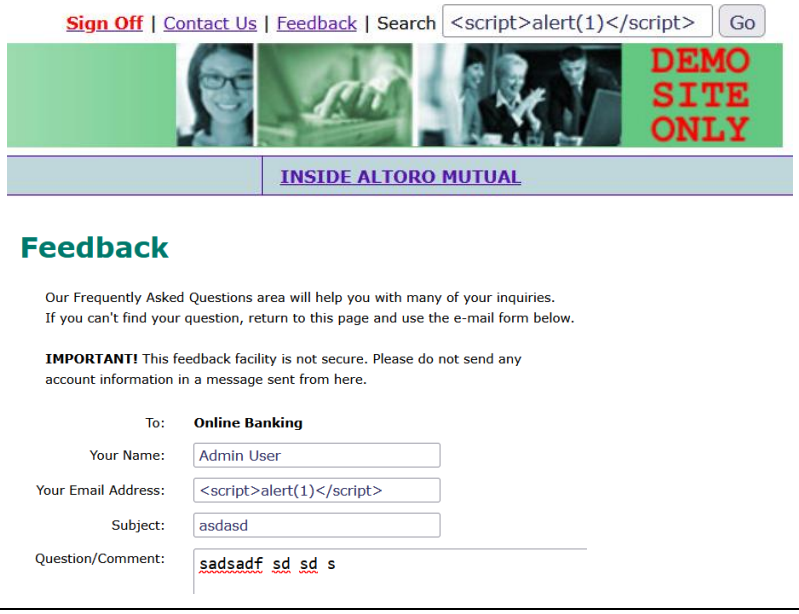


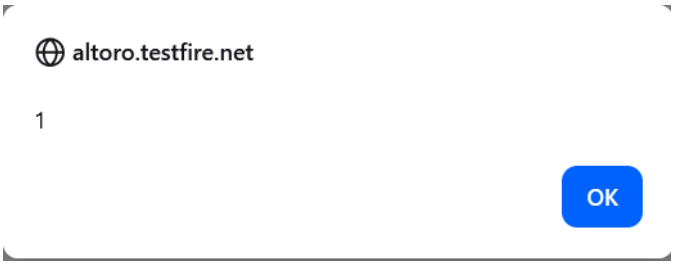


Highly Confidential

Management response	djaGDJGjfakj

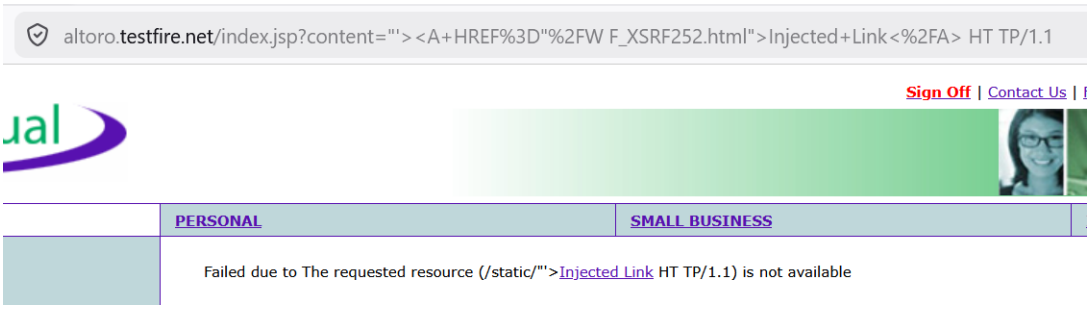
6. Reflected Cross-Site Scripting (XSS)

Risk Rating	High
Category	Injection
Description	The application is vulnerable to Reflected Cross-Site Scripting (XSS) due to improper validation and sanitization of user-supplied input. Malicious scripts injected into input fields are reflected in the response and executed in the user's browser. For example, injecting: <script>alert(1)</script> in the query parameter successfully executes JavaScript, confirming the presence of XSS.
Impact	• Execution of arbitrary JavaScript in victim's browser • Session hijacking via cookie theft • Phishing attacks and user redirection • Defacement of web pages • Unauthorized actions performed on behalf of users
Recommendation	1. Implement proper input validation and output encoding 2. Use context-aware encoding (HTML, JavaScript, URL) 3. Apply Content Security Policy (CSP) headers 4. Sanitize all user inputs on both client and server side 5. Use secure frameworks/libraries that automatically escape output
Finding Id	
Target Date	
Affected URLs	
Evidences	



Management response	djaGDJGjfakj

7. Reflected Cross-Site Scripting (XSS) via HTML Injection in content Parameter

Risk Rating	High
Category	Injection
Description	The application is vulnerable to Reflected Cross-Site Scripting (XSS) through the content parameter in /index.jsp. User-supplied input is reflected in the HTTP response without proper sanitization or encoding, allowing HTML content to be injected and rendered in the browser. It was observed that crafted input containing HTML tags was successfully rendered as a clickable link in the response, confirming improper handling of user input.
Impact	• Attackers can inject malicious HTML content into web pages • Enables phishing attacks by embedding deceptive links • Can manipulate the user interface and mislead users • May be escalated to full XSS (JavaScript execution)
Recommendation	1. Implement proper input validation and sanitization 2. Apply context-aware output encoding (HTML encoding) 3. Use security controls such as Content Security Policy (CSP) 4. Avoid directly reflecting user input in responses without filtering
Finding Id	
Target Date	
Affected URLs	
Evidences	

Highly Confidential

Management response	djaGDJGjfakj

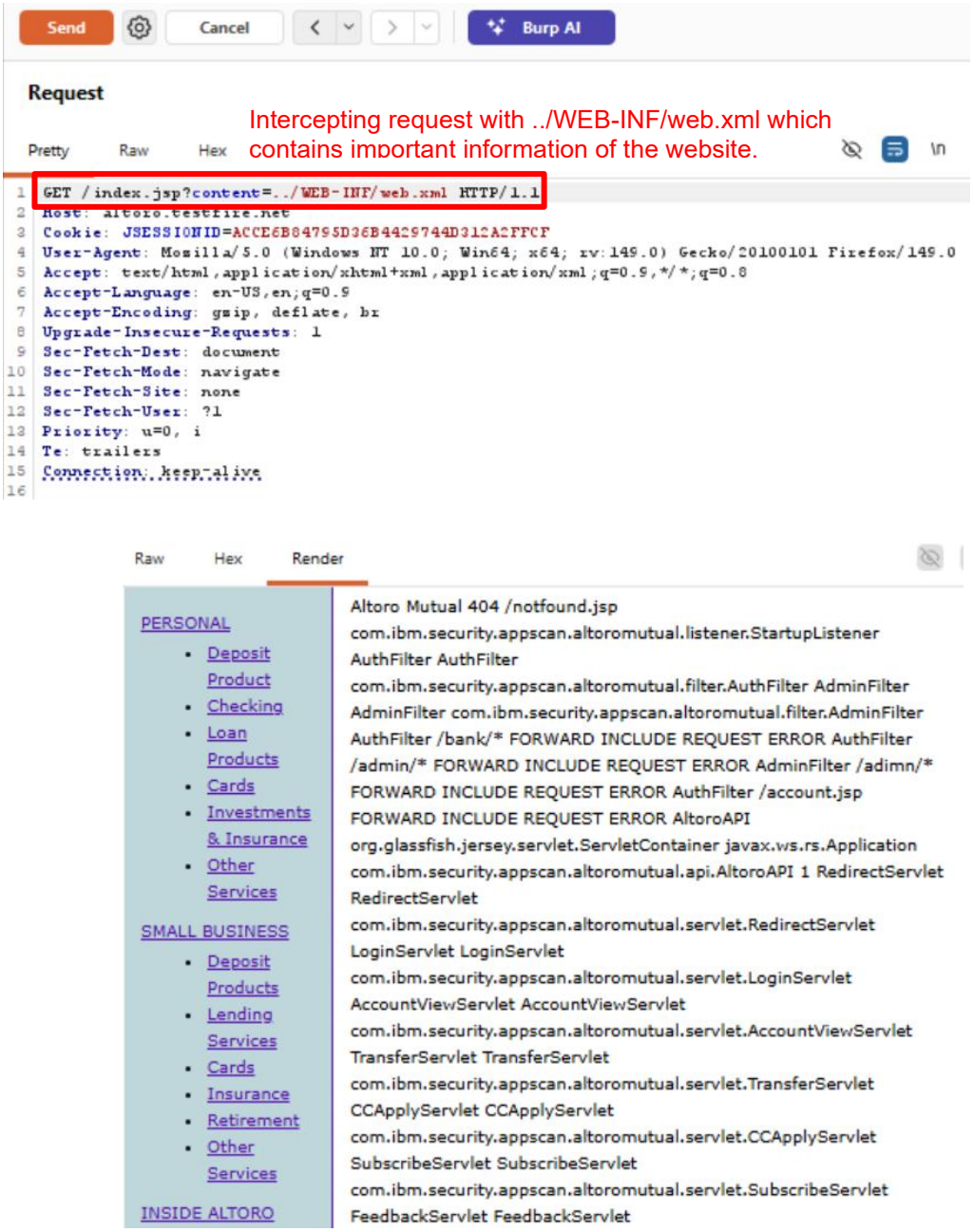
8. Sensitive Information Disclosure via Verbose Error Message in content Parameter

Risk Rating	**Medium**
Category	**Security Misconfiguration**
Description	The application exposes internal implementation details when invalid input is supplied to the content parameter: `/index.jsp?content=../WEB-INF/web.xml` Instead of returning a generic error, the application discloses: • Internal Java class names • Servlet mappings • Application structure • Framework details (e.g., GlassFish, IBM AppScan components) This indicates improper error handling and lack of secure configuration.

Impact	This vulnerability allows attackers to:
	• Map internal application architecture
	• Identify hidden endpoints such as:
	  ○ /admin/addUser
	  ○ /admin/changePassword
	  ○ /doTransfer
	• Discover backend technologies and frameworks
	• Perform **targeted attacks** (IDOR, SQLi, auth bypass, etc.)
	This significantly **reduces attack complexity** and helps chain further exploits.

Recommendation	1. Disable Verbose Error Messages 2. Implement Custom Error Pages 3. Remove Sensitive Data from Responses 4. Input Validation 5. Restrict Access to Internal Resources
Finding Id	
Target Date	
Affected URLs	

Highly Confidential

Evidences	
Management response	djaGDJGjfakj

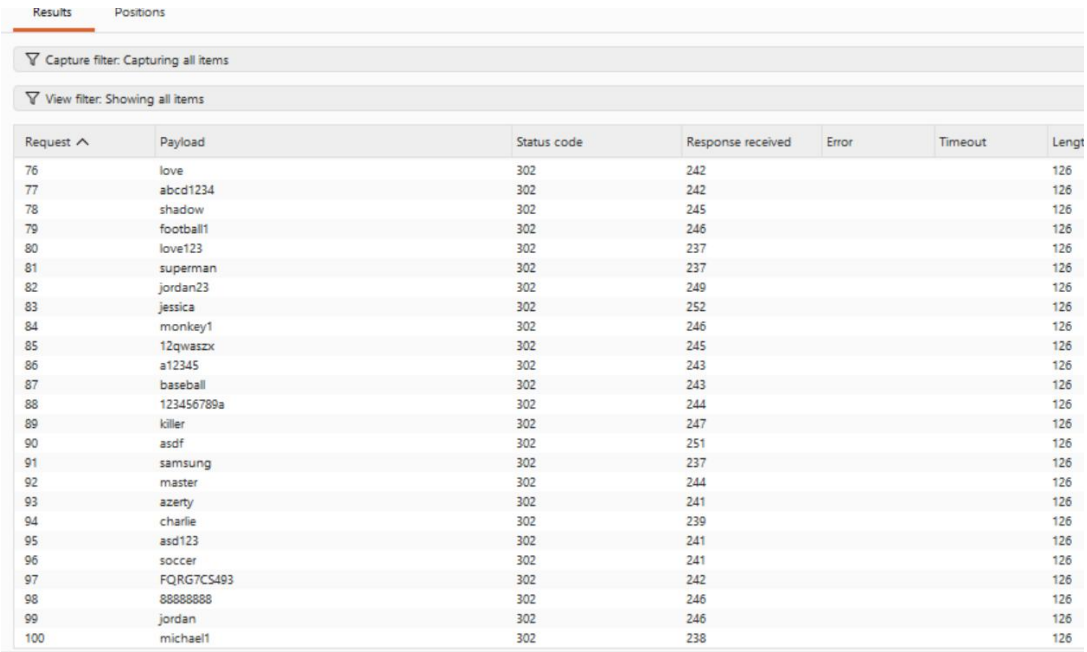
9. Use of Outdated Apache Tomcat (Apache-Coyote/1.1) with Known CVE Vulnerabilities

Risk Rating	**Medium**
Category	Software Supply Chain Failures
Description	The application discloses the use of Apache-Coyote/1.1, which is the HTTP connector used by Apache Tomcat. This indicates that the backend server is running a version of Apache Tomcat that may be outdated or unpatched, potentially exposing the application to publicly known vulnerabilities (CVEs). Server identification is typically observed in HTTP response headers such as:

	Server: Apache-Coyote/1.1 Outdated versions of Apache Tomcat are associated with multiple critical vulnerabilities, including but not limited to: • Remote Code Execution (RCE) • Information Disclosure • Authentication bypass • Denial of Service (DoS) The application exposes numerous internal and administrative endpoints that are accessible or discoverable by unauthenticated or low-privileged users.
Impact	• Remote Code Execution (RCE) • Sensitive file disclosure • Server compromise • Unauthorized access to internal resources • Full takeover of application server
Recommendation	1. Upgrade Server 2. Remove Version Disclosure 3. Disable Unused Services 4. Regular Vulnerability Management
Finding Id	
Target Date	
Affected URLs	
Evidences	CVE-2005-2090 Jakarta Tomcat 5.0.19 (Coyote/1.1) and Tomcat 4.1.24 (Coyote/1.0) allows remote attackers to poison the web cache, bypass web application firewall protection, and conduct XSS attacks via an HTTP request with both a "Transfer-Encoding: chunked" header and a Content-Length header, which causes Tomcat to incorrectly handle and forward the body of the request in a way that causes the receiving server to process it as a separate HTTP request, aka "HTTP Request Smuggling." https://nvd.nist.gov/vuln/detail/cve-2005-2090
Management response	djaGDJGjfakj

Highly Confidential

10. Insufficient Logging and Monitoring of Authentication Attempts

Risk Rating	**Medium**
Category	Security Logging and Monitoring Failures
Description	The application fails to implement adequate logging and monitoring mechanisms for authentication activities. Multiple failed login attempts can be performed without triggering any alerts, account lockout, or defensive response. This indicates that login attempts are either not properly logged or not actively monitored, allowing malicious activities to go undetected.
Impact	• Attackers can perform brute-force or credential stuffing attacks without detection • Delayed or no response to ongoing attacks • Increased risk of account compromise • Lack of forensic data for incident investigation
Recommendation	1. Implement detailed logging for all authentication attempts (successful and failed) 2. Configure real-time monitoring and alerting for suspicious activities (e.g., multiple failed logins) 3. Integrate SIEM solutions for centralized monitoring. 4. Establish thresholds for triggering alerts and automated responses. 5. Regularly review logs and conduct security audits.
Finding Id	
Target Date	
Affected URLs	
Evidences	

Results Positions

Capture filter: Capturing all items

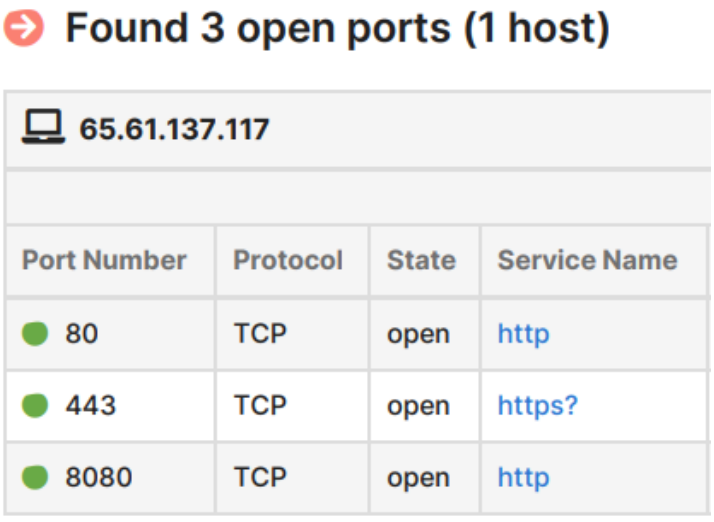
View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Lengt
76	love	302	242			126
77	abcd1234	302	242			126
78	shadow	302	245			126
79	football1	302	246			126
80	love123	302	237			126
81	superman	302	237			126
82	jordan23	302	249			126
83	jessica	302	252			126
84	monkey1	302	246			126
85	12qwaszx	302	245			126
86	a12345	302	243			126
87	baseball	302	243			126
88	123456789a	302	244			126
89	killer	302	247			126
90	asdf	302	251			126
91	samsung	302	237			126
92	master	302	244			126
93	azerty	302	241			126
94	charlie	302	239			126
95	asd123	302	241			126
96	soccer	302	241			126
97	FQRG7CS493	302	242			126
98	88888888	302	246			126
99	jordan	302	246			126
100	michael1	302	238			126

Highly Confidential

Management response	djaGDJGjfakj

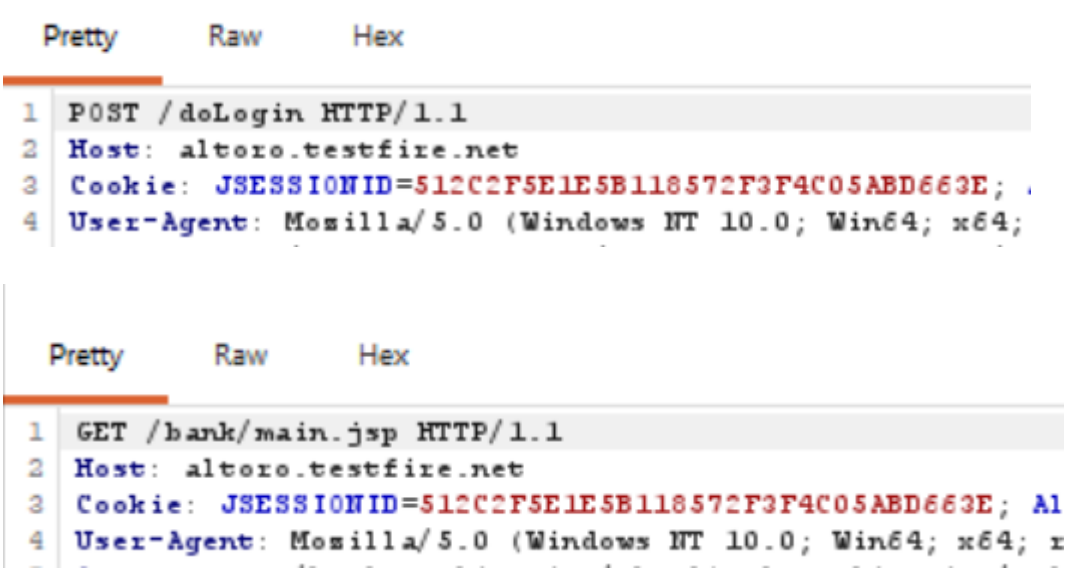
11. Multiple Sensitive Open Ports Exposing Internal Services

Risk Rating	**Medium**
Category	Security Misconfiguration
Description	The application server exposes multiple network ports (21, 22, 25, 80, 443, 8080, 8443, 3306) that are accessible externally. Several of these ports correspond to sensitive services such as FTP (21), SSH (22), SMTP (25), Tomcat/Web services (8080, 8443), and MySQL database (3306). These services are exposed without proper access restrictions, increasing the attack surface and indicating improper server hardening and configuration.
Impact	• Unauthorized access to critical services (FTP, SSH, Database) • Potential database compromise via exposed MySQL port • Exposure of administrative interfaces (e.g., Tomcat on 8080/8443) • Increased risk of brute-force and service-specific attacks • Higher likelihood of exploitation of known vulnerabilities
Recommendation	1. Close all unnecessary and unused ports 2. Restrict access to sensitive services using firewall rules (IP allowlisting) 3. Disable publicly accessible database services (e.g., MySQL) 4. Secure administrative interfaces behind authentication and VPN 5. Regularly perform port scanning and service audits 6. Follow secure server hardening best practices
Finding Id	
Target Date	
Affected URLs	
Evidences	

Highly Confidential

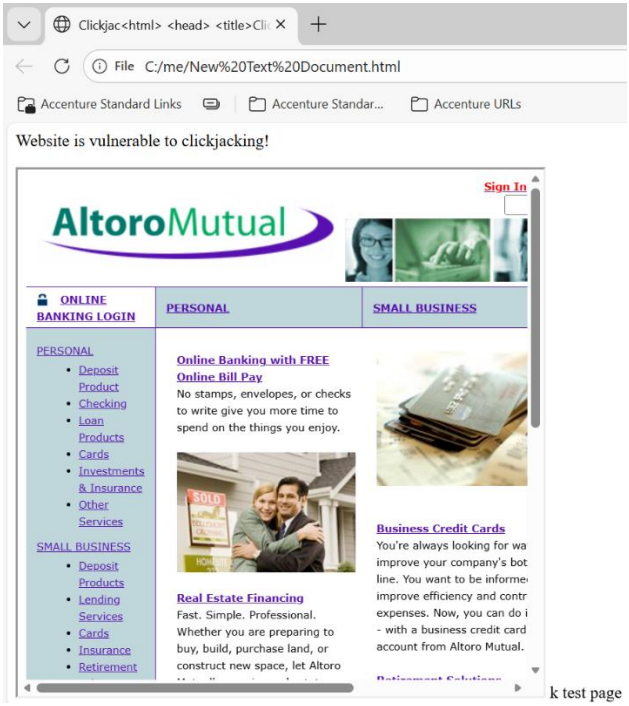
Management response	djaGDJGjfakj

12. Weak Session Management – Session ID Not Invalidated After Logout

Risk Rating	**Medium**
Category	Authentication Failures
Description	The application does not properly invalidate user sessions upon logout. It was observed that the same session identifier (JSESSIONID) remains valid even after the user logs out and logs back in. This indicates that session tokens are not being regenerated or invalidated, allowing reuse of existing sessions.
Impact	• Session fixation attacks may be possible • Unauthorized users can reuse old session IDs to gain access • Increased risk of account compromise if session tokens are exposed • Users may remain authenticated unintentionally
Recommendation	1. Invalidate session tokens immediately upon logout 2. Generate a new session ID upon successful login 3. Implement proper session lifecycle management 4. Use secure cookie attributes such as HttpOnly and Secure 5. Set appropriate session timeout and expiration policies
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

Highly Confidential

13. Clickjacking Vulnerability due to Missing Frame Protection

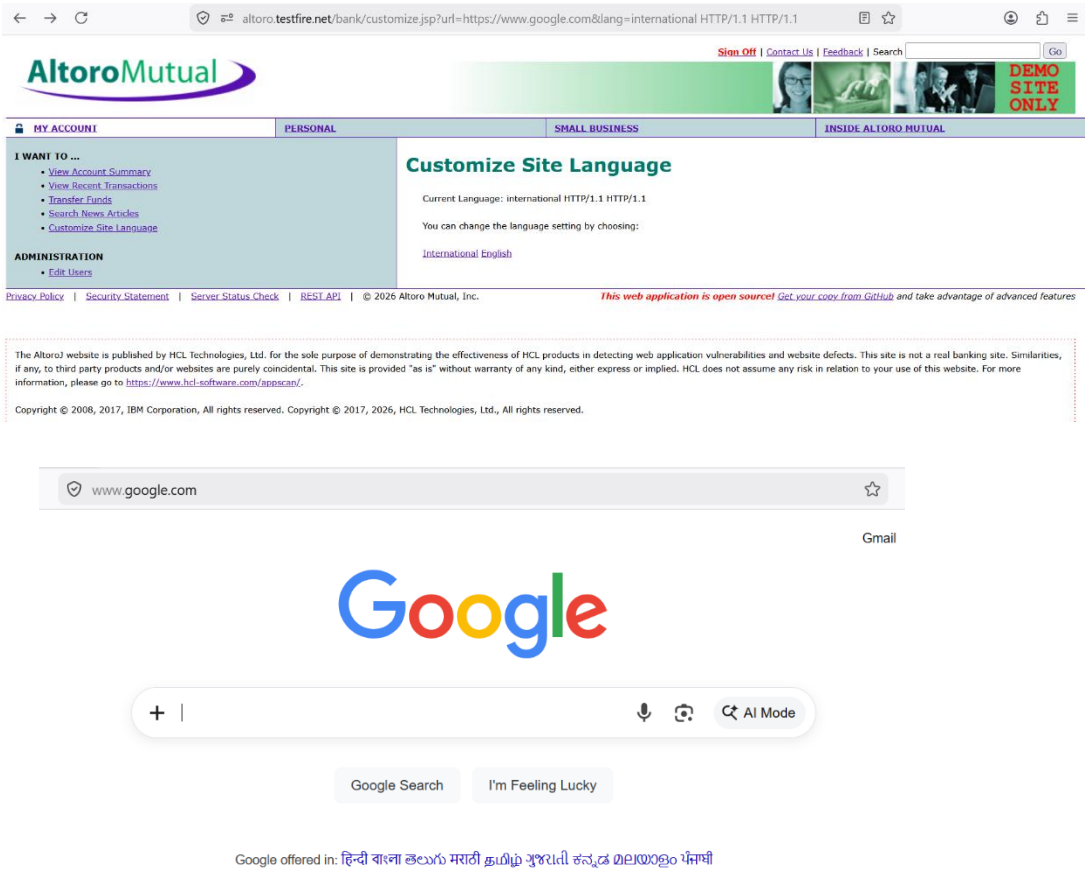
Risk Rating	**Medium**
Category	Security Misconfiguration
Description	The application is vulnerable to Clickjacking as it allows its pages to be embedded within an <iframe> on an external domain. This occurs due to the absence of appropriate security headers such as X-Frame-Options or Content-Security-Policy. A proof-of-concept HTML page was created that embeds the application inside an iframe, successfully loading the target page. This confirms that the application does not restrict framing and is susceptible to UI redressing attacks.
Impact	• Attackers can trick users into clicking hidden or disguised elements • Unauthorized actions can be performed on behalf of the user • Potential compromise of sensitive operations (e.g., fund transfers) • Increased risk of phishing and social engineering attacks
Recommendation	1. Implement X-Frame-Options header with values DENY or SAMEORIGIN 2. Use Content-Security-Policy with frame-ancestors 'none' or trusted domains only. 3. Ensure all sensitive pages are protected against framing 4. Perform regular security header audits
Finding Id	
Target Date	
Affected URLs	
Evidences	

Highly Confidential

	Payload: <html> <head> <title>Clickjac<html> <head> <title>Clickjack test page</title> </head> <body> Website is vulnerable to clickjacking! <iframe src="http://testfire.net/" width="500" height="500"></iframe> </body> </html>k test page</title> </head> <body> Website is vulnerable to clickjacking! <iframe src="http://testfire.net/" width="500" height="500"></iframe> </body> </html>
Management response	djaGDJGjfakj

14. Open Redirect via URL Parameter

Risk Rating	Low
Category	Broken Access Control
Description	The application accepts a user-controlled url parameter (e.g., /bank/customize.jsp?url=...) and reflects or processes it without proper validation. This allows an attacker to manipulate the parameter and redirect users to unintended or potentially malicious destinations. The observed behavior shows that injected values are processed and displayed, indicating insufficient validation or sanitization of redirect inputs.
Impact	Attackers can craft malicious links that appear to originate from a trusted domain, leading users to phishing sites or malicious content. This can result in credential theft, session hijacking, or malware distribution. It also damages user trust in the application.
Recommendation	Validate and restrict redirect URLs to a predefined allowlist of trusted domains. Avoid directly using user-supplied input for redirection. Implement proper input validation and encode output where necessary. Additionally, use indirect references (e.g., mapping IDs to URLs) instead of exposing raw URLs in parameters.
Finding Id	
Target Date	
Affected URLs	

Evidences	
Management response	djaGDJGjfakj

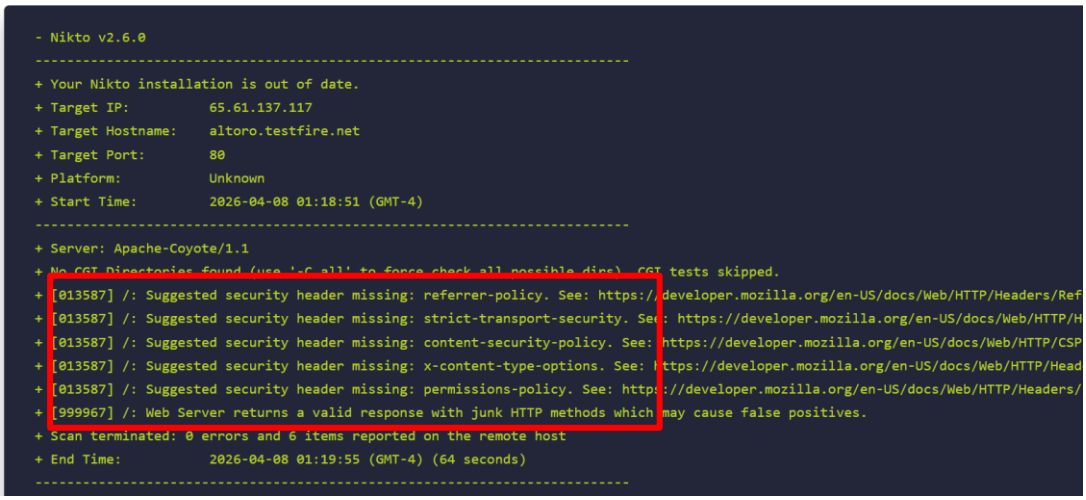
15. Cross-Site Request Forgery (CSRF) in Login Functionality

Risk Rating	Low
Category	Broken Access Control
Description	The application is vulnerable to Cross-Site Request Forgery (CSRF) in the /doLogin endpoint. The login request does not include any anti-CSRF token or additional validation mechanism to verify the authenticity of the request. As a result, an attacker can craft a malicious request that forces a victim's browser to perform unintended login actions without their consent.
Impact	• Attackers can force users to log in as a different account (Login CSRF) • May lead to session confusion or misuse of application functionality • Can be used as a stepping stone for further attacks such as phishing
Recommendation	1. Implement anti-CSRF tokens for all state-changing requests 2. Validate CSRF tokens on the server side 3. Use SameSite cookie attributes (Strict or Lax) 4. Require re-authentication or additional verification for sensitive actions
Finding Id	

Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

16. Missing Security Headers Leading to Increased Attack Surface

Risk Rating	**Low**
Category	Security Misconfiguration
Description	The application is missing several important HTTP security headers that help protect against common web attacks. From the scan results: • Missing **Strict-Transport-Security (HSTS)** • Missing **Content-Security-Policy (CSP)** • Missing **X-Content-Type-Options** These headers are essential for enforcing browser-side security controls. Their absence increases the risk of exploitation through client-side attacks.
Impact	• Increased risk of **Man-in-the-Middle (MITM)** attacks (no HSTS) • Exposure to **Cross-Site Scripting (XSS)** (no CSP) • MIME sniffing attacks (no X-Content-Type-Options) • Overall weaker browser-side security posture While not directly exploitable alone, these significantly **amplify other attacks**.
Recommendation	• Implement Security Headers Strict-Transport-Security: max-age=31536000; includeSubDomains Content-Security-Policy: default-src 'self' X-Content-Type-Options: nosniff
Finding Id	
Target Date	

Affected URLs	
Evidences	
Management response	djaGDJGjfakj

Highly Confidential

6 Next Steps

Findings will have to be remediated within the Accenture Security remediation timelines. The BU security officer would ensure remediation as per the agreed timelines. Accenture Security will document and track the progress within the Archer eGRC system.

7 Vulnerability Remediation Timeline

The below remediation timeline is according to Accenture Security Standards.

Severity Level	Severity Description
Critical/High	The exploitation of a "Critical/High" vulnerability could cause a significant damage to the functionality of underlying system coupled with substantial data loss/exposure to an unauthorized user. Such findings must be fixed in less than 30 days.
Medium	Medium severity bugs allow attackers to read or modify limited amounts of information, or which are not harmful on their own but potentially harmful when combined with other bugs. Such findings must be fixed within 90 days.
Low	A vulnerability caused by unnecessary services or services providing information leakage which could be useful to an attacker to execute other more sophisticated attacks. Such findings must be fixed within 180 days.

Highly Confidential