Altoro Mutual

Penetration testing report

-

Confidentiality & Legal Notice

This document contains sensitive security information related to the penetration test conducted on the Altoro Mutual Bank web application.

Access is strictly limited to authorized personnel of the client organization. Unauthorized access, distribution, or reproduction is prohibited.

Handling & Security Guidelines

Access Control:
This report is classified and should only be shared with individuals explicitly authorized by the client organization.

Prohibited Actions:
Do not distribute, copy, or disclose any portion of this document without prior approval.

Secure Storage & Disposal:
Store securely in encrypted digital storage or locked physical locations.

Dispose of securely via shredding (physical copies) or secure deletion (digital copies).

Legal Disclaimer

The findings and recommendations in this report are based on controlled testing within the agreed scope.

While every effort has been made to ensure accuracy, the client organization bears sole responsibility for addressing identified risks.

The penetration testing team assumes no liability for any misuse, misinterpretation, or consequences arising from this report.

Executive Summary

This penetration test was conducted on the **Altoro Mutual Bank web application** (http://altoro.testfire.net/) to assess its security posture and identify exploitable vulnerabilities.

The objective was to uncover weaknesses that could be leveraged by malicious actors and provide actionable remediation recommendations.

Scope: The assessment focused on the web application and its associated services.

Testing Methodology: A **black-box** approach was used, incorporating **reconnaissance**, **vulnerability scanning**, **exploitation**, **and post-exploitation analysis** to simulate real-world attack scenarios.

Key Findings:

- **SQL Injection** – Gained unauthorized access to the admin dashboard via SQL Injection.
- **Insecure Direct Object Reference (IDOR)** – Unauthorized users can access or modify other users' data by manipulating object identifiers in API requests, leading to data exposure or privilege escalation.
- **Denial of Service (DoS) Vulnerability** – The server is susceptible to a Slowloris DoS attack.
- **Reflected Cross-Site Scripting (XSS)** – Multiple instances detected through automated testing.
- **Cross-Site Request Forgery (CSRF)** – Forms lack anti-CSRF tokens, exposing users to potential CSRF attacks.
- **Insecure Cookie Handling** – JSESSIONID cookie lacks the Secure flag, making it susceptible to session hijacking.
- **Exposed API Documentation** – Swagger UI was publicly accessible at /swagger/index.html, potentially aiding attackers in reconnaissance.

The identified vulnerabilities pose significant security risks, emphasizing the need for prompt remediation to safeguard user data and system integrity.

Risk Assessment

The identified vulnerabilities pose significant risks to the **confidentiality, integrity, and availability** of sensitive data.
The most critical issues — **SQL Injection, CSRF, and Slowloris DoS** — could allow attackers to gain unauthorized access, manipulate data, or disrupt services.
Addressing these vulnerabilities is crucial to maintaining system security and preventing potential exploitation.

Recommendations

Immediate Actions:
Patch **SQL Injection** vulnerabilities to prevent unauthorized database access.
Implement **anti-CSRF tokens** to mitigate CSRF attacks.
Deploy **rate-limiting and connection management** to defend against Slowloris DoS attacks.

Long-Term Strategies:
Adopt **secure coding practices** to prevent common web vulnerabilities.
Conduct **regular security audits and penetration tests** to identify emerging threats.
Implement **developer and user security awareness training** to strengthen overall security posture.

Introduction

Purpose

This penetration test was conducted to evaluate the **security posture** of the **Altoro Mutual Bank web application** (http://altoro.testfire.net/).

The objective was to identify and assess vulnerabilities that could be exploited by malicious actors and to provide **actionable recommendations** for mitigation.

The assessment focused on weaknesses in both **design and implementation,** ensuring the **confidentiality, integrity, and availability** of sensitive data.

Scope

- **Target Systems:** The web application hosted at http://altoro.testfire.net/.
- **Exclusions:** No exclusions were specified for this engagement.
- **Testing Period:** Conducted from **[02/09/2025] to [02/09/2025]**.

Methodology

The penetration test followed a **structured approach** to ensure thorough evaluation and identification of security weaknesses.

Reconnaissance

- Conducted **passive and active information gathering** using tools such as **Nmap, Shodan, Feroxbuster, nslookup, dig, and Wapiti**.
- Identified **open ports, running services, directories, and exposed files**.

Vulnerability Scanning

- Performed **automated scanning using OWASP ZAP, Wapiti, and manual testing**.
- Identified common vulnerabilities, including **SQL Injection, Cross-Site Scripting (XSS), and CSRF.**

Exploitation

- Exploited identified vulnerabilities using **SQLmap, custom scripts, and manual testing** to assess their severity.

Post-Exploitation

- Evaluated the impact of successful exploits, including **unauthorized access to sensitive data, system compromise, and denial of service.**

Detailed Findings

1. SQL Injection on Login Page

Description

The login page is **vulnerable to SQL Injection**, allowing attackers to manipulate input fields and execute arbitrary SQL queries.
This can lead to **authentication bypass** or the extraction of sensitive data from the database.

- **Affected System:** http://altoro.testfire.net/bank/login.aspx
- **Attack Vector:** Malicious input in the **username** and **password** fields.

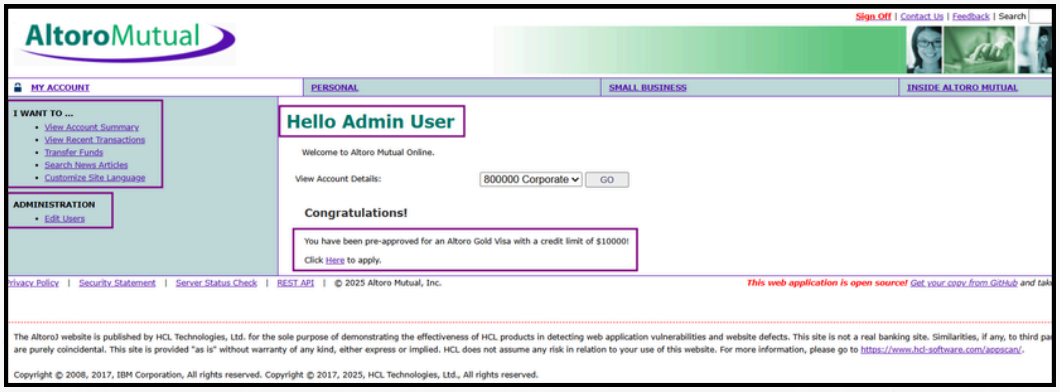
Risk Rating: Critical

Impact

- An attacker could bypass authentication, gain unauthorized access to user accounts, and extract or manipulate sensitive data within the database.

Proof of Concept (PoC)

Input: admin' --
Outcome: Bypasses authentication and grants access.



Recommendations

- Implement **parameterized queries** or **prepared statements** to prevent direct SQL execution.
- Enforce **input validation and sanitization** for all user inputs.
- Regularly scan for SQL Injection vulnerabilities using tools like **OWASP ZAP** or **SQLmap**.

Steps to Reproduce

1. Navigate to the login page: http://altoro.testfire.net/bank/login.aspx.
2. In the **username** field, input: **admin' --**.
3. In the password field, input a random character.
4. Click **Login**.
5. Result: Successful login without valid credentials, granting access to the admin dashboard.

This vulnerability must be addressed immediately to prevent unauthorized access and data exposure.

2. Insecure Direct Object Reference (IDOR)

Description

The application is vulnerable to **Insecure Direct Object Reference (IDOR)** due to the use of predictable account IDs (e.g., 800004, 800001) and Base64-encoded sensitive data. An attacker can decode, modify, and re-encode the Base64 string to access unauthorized accounts.

- **Affected System:** User account access functionality.
- **Attack Vector:** Manipulation of Base64-encoded account IDs.

Risk Rating: Critical

Impact

- Unauthorized access to sensitive financial data (e.g., balances, credit card details).
- Potential for financial fraud or data breaches.

Recommendations

- Implement **authorization checks** to ensure users can only access their own data.
- Replace **predictable account IDs** with **non-predictable identifiers** (e.g., **UUIDs**).
- Avoid exposing sensitive data in **Base64** or weakly encoded formats.
- Use **encryption** to secure sensitive information.

Steps to Reproduce

1. Access an account and extract the Base64-encoded data with **Burp Suite**.
- **ODAwMDA0flNhdmluZ3N+LTM2OTMzNTkuMHw4MDAwMDV+Q2hlY2tpbmd+MjUuMHw0NDg1OTgzMzU2MjQyMjE3fkNyZWRpdCBDYXJkfi00LjQ0MTUzODkxMTc4Nzc3MkUxNXw=**
2. Decode the Base64 string:
- **800004~Savings~-3693359.0|800005~Checking~25.0|4485983356242217~CreditCard~-4.4415389117877 72E15|**
3. Modify the account ID (e.g., change **800004** to **800001**).
4. Re-encode the modified string using Base64.
5. Submit the re-encoded string to the application.
6. **Result:** The application returns sensitive data for the modified account ID, confirming the **IDOR** vulnerability.

This issue must be addressed immediately to prevent unauthorized access to sensitive user data.

3. Slowloris Denial of Service (DoS) Vulnerability

Description

The web server is **vulnerable to Slowloris Denial of Service (DoS)** attacks, which involve sending partial HTTP requests and keeping them open indefinitely.
This can eventually exhaust the server's resources and cause a denial of service, preventing legitimate users from accessing the application.

- **CVE:** CVE-2007-6750
- **Affected Systems:** Web server running on ports **80, 443,** and **8080.**
- **Attack Vector:** Sending partial HTTP requests to the server.

Risk Rating: Critical

Impact

- **Denial of Service** (DoS), making the web application **unavailable** to legitimate users.
- **Server resource exhaustion**, impacting the application's **availability** and **performance**.

Recommendations

- Implement **rate-limiting** on incoming connections to prevent resource exhaustion.
- Use a **Web Application Firewall (WAF)** to detect and block Slowloris attack patterns.
- **Update web server software** to the latest version to patch known vulnerabilities.
- Configure the server to **close idle connections faster**, e.g., reducing **Timeout** values.
- Regularly **monitor server logs** for unusual traffic patterns to detect potential DoS attacks early.

Steps to Reproduce

1. Use **slowhttptest** or similar tools to simulate a Slowloris attack against the server.
2. Configure the tool to send partial HTTP requests and keep them open indefinitely.
3. **Result:** The server becomes unresponsive, confirming the vulnerability to **Slowloris attacks**.

This vulnerability must be addressed promptly to prevent potential service disruption and ensure the availability of the web application for legitimate users.

4. Reflected Cross-Site Scripting (XSS)

Description

Multiple instances of **Reflected Cross-Site Scripting (XSS)** vulnerabilities were identified in various pages of the application.
These vulnerabilities occur when user input is reflected back in the response without proper sanitization, allowing attackers to inject malicious JavaScript into the page.

- **Affected Pages and Parameters:**
  - **/index.jsp :** Injection in the **content** parameter
  - **/search.jsp :** Injection in the **query** parameter
  - **/sendFeedback :** Injection in the **name** and **email_addr** parameters
  - **/customize.jsp :** Injection in the **lang** parameter
- **Attack Vector:** Malicious input in URL parameters or form fields.

Risk Rating: Critical

Impact

- **Execution of arbitrary JavaScript** in a victim's browser.
- Potential for **session cookie theft, phishing attacks,** or other **malicious actions** that compromise the security of the user.

Proof of Concept (PoC)

Input: Injected payload in the query parameter of **/search.jsp.**
Outcome: JavaScript alert box popped up.

Recommendations

- **Sanitize all user inputs** before reflecting them in the response.
- Use a **Web Application Firewall (WAF)** to detect and block malicious payloads.
- Implement **Content Security Policy (CSP)** headers to restrict the execution of inline scripts.
- Regularly test all input fields and parameters for **XSS vulnerabilities** using both automated tools (e.g., OWASP ZAP) and manual testing.

Steps to Reproduce

1. Navigate to the **/search.jsp** page.
2. Inject the following payload into the **query** parameter:
   <svg onload=alert("SHARDS")>
3. Submit the request.
4. **Result:** The injected JavaScript executes and a JavaScript alert box pops up, confirming the presence of a **Reflected XSS** vulnerability.

This vulnerability must be fixed to prevent attackers from exploiting user inputs and executing arbitrary scripts in victims' browsers.

5. Lack of Anti-CSRF Tokens

Description

Multiple forms within the web application are vulnerable to **Cross-Site Request Forgery (CSRF)** attacks due to the absence of **anti-CSRF tokens**.
This vulnerability allows an attacker to trick an authenticated user into performing unintended actions by crafting malicious requests that execute actions on the user's behalf.

- **Affected Forms:**
  - **/doLogin**
  - **/doSubscribe**
  - **/sendFeedback**
  - **/bank/doTransfer**
  - **/bank/main**
- **Attack Vector:** Crafting malicious requests to perform unauthorized actions on behalf of authenticated users.

Risk Rating: Critical

Impact

- An attacker can perform **unauthorized actions** on behalf of an authenticated user, leading to potential **data loss, configuration changes,** or other **malicious activities** that compromise the integrity of the application and its data.

Recommendations

- Implement **CSRF tokens** in all forms and API requests to verify the legitimacy of requests.
- Use the **SameSite** cookie attribute to mitigate CSRF risks.
- Validate the **origin** of requests using **Referer** or **Origin** headers.
- Educate developers on secure coding practices to prevent CSRF vulnerabilities in future updates.
- Inform users about the **risks of clicking suspicious links** to reduce the likelihood of being targeted.

Steps to Reproduce

1. Craft an **HTML form** with a **malicious POST request** targeting the **/doLogin** form, containing an attacker-controlled action.
2. Submit the form on behalf of the authenticated user.
3. **Result:** The malicious request successfully logs the user in without their interaction, confirming the absence of anti-CSRF tokens.

To prevent CSRF attacks, the implementation of **anti-CSRF tokens** is crucial to ensuring the integrity of user actions and protecting against unauthorized modifications.

6. Insecure Cookie Handling

Description

The JSESSIONID cookie used by the application lacks both the **Secure** and **HttpOnly** flags.

As a result, it is vulnerable to interception over unencrypted connections and can be accessed via client-side scripts.

- **Affected Systems:** Web application at **http://altoro.testfire.net/**
- **Attack Vector:**
  - Interception of cookies over unencrypted connections (HTTP instead of HTTPS).
  - Accessing cookies through client-side scripts, allowing for potential session hijacking.

Risk Rating: Critical

Impact

- An attacker could potentially intercept the **JSESSIONID** cookie when transmitted over an unencrypted connection or access it using client-side scripts, leading to **session hijacking.** This could allow the attacker to impersonate a user and perform unauthorized actions.

Recommendations

- **Set the Secure flag** on all cookies to ensure they are only transmitted over HTTPS, preventing interception over unencrypted connections.
- **Set the HttpOnly flag** to prevent client-side scripts from accessing cookies, reducing the risk of session hijacking via XSS attacks.
- Regularly audit cookie settings to ensure proper security flags are in place.

Steps to Reproduce

1. **Create two user accounts** on the application.
2. **Intercept traffic** between the client and server using **Burp Suite.**
3. **Copy the JSESSIONID cookie** from the first user's session.
4. **Inject the copied JSESSIONID cookie** into the second user's session.
5. **Result: Access the second user's account** with the hijacked session, successfully confirming the vulnerability.

This issue must be addressed immediately to prevent session hijacking and unauthorized access to user accounts.

7. Access to Sensitive Configuration File (web.xml)

Description

The application allows unauthorized access to the **web.xml** configuration file located in the **/WEB-INF/** directory. This file contains sensitive configuration details, including servlet mappings, filters, and initialization parameters, which could assist an attacker in exploiting the application further.

- **Affected Systems:** Web server accessible at **http://altoro.testfire.net/**
- **Attack Vector:** Accessing the **web.xml** file via a **directory traversal** payload.

Risk Rating: High

Impact

- Exposure of **sensitive information** regarding the application's structure and behavior.
- Potential exploitation of **misconfigured security settings** (e.g., unprotected administrative endpoints), allowing further attack vectors.

Recommendations

- Restrict access to the **/WEB-INF/** directory and sensitive files through proper **file access controls.**
- Implement **input validation** to prevent **directory traversal** attacks.
- Regularly audit **file permissions** and ensure proper **access controls** are in place for sensitive files.

Steps to Reproduce

1. Access the web application via the following URL, incorporating a **directory traversal payload** to access **web.xml**:
   http://altoro.testfire.net/index.jsp?content=../WEB-INF/web.xml
2. **Result:** The server returns the contents of the **web.xml** file, revealing sensitive configuration details such as servlet mappings and initialization parameters.

By restricting access to sensitive files and implementing stronger security controls, you can mitigate the risk of exposing critical configuration details that could aid attackers.

8. Swagger UI Exposure

Description

The Swagger API documentation is publicly accessible at **/swagger/index.html.** Exposing this documentation can provide attackers with detailed information about the application's endpoints, which could assist in identifying vulnerabilities or planning further attacks.

- **Affected Systems:** Web application at **http://altoro.testfire.net/**
- **Attack Vector:** Accessing the Swagger UI documentation exposed on the server.

Risk Rating: Medium

Impact

- Exposing API documentation increases the attack surface by revealing sensitive information such as endpoint details, methods, and parameters. This could allow attackers to identify potential weaknesses in the application.

Recommendations

- **Restrict access** to the Swagger UI by requiring authentication, ensuring only authorized users can view the documentation.
- **Remove or disable Swagger UI** in production environments to minimize exposure.
- Regularly **audit web servers** to ensure no sensitive documentation is unintentionally exposed.

Steps to Reproduce

1. **Access the exposed Swagger UI documentation** at **/swagger/index.html** on the target application.
2. **Review the exposed API endpoints,** which may provide information useful for identifying potential vulnerabilities.

This issue must be addressed immediately to prevent attackers from gaining insights into the application's API endpoints, which could be leveraged for further exploitation.

9. Missing Security Headers

Description

The application is missing critical HTTP security headers, leaving it vulnerable to attacks such as Clickjacking and MIME type sniffing. Specifically, the following headers are absent:

- **X-Frame-Options:** Not set, making the application vulnerable to Clickjacking attacks.
- **X-Content-Type-Options:** Not set, increasing the risk of MIME type sniffing attacks.

Risk Rating: Medium

Impact

The absence of these headers can allow attackers to perform the following:

- **Clickjacking:** Users may be tricked into interacting with hidden frames or elements, which could result in unintended actions.
- **MIME Type Sniffing:** Attackers may exploit the absence of content-type validation, leading to the execution of malicious content.

Recommendations

- **Set the X-Frame-Options** header to **DENY** or **SAMEORIGIN** to mitigate Clickjacking risks.
- **Set the X-Content-Type-Options** header to **nosniff** to prevent MIME type sniffing attacks.
- Regularly **review and update HTTP headers** to ensure they comply with security best practices.

Steps to Reproduce

1. **Use a security scanner** like Wapiti to perform a header scan.
2. **Check for the absence** of **X-Frame-Options** and **X-Content-Type-Options** in the HTTP response headers.
3. The **scan confirms** the missing headers, verifying the vulnerability.

This issue must be addressed immediately to mitigate the risk of Clickjacking, MIME type sniffing, and other attacks that exploit the absence of essential security headers.

10. Information Disclosure in Source Code

Description

Sensitive information, such as a reference to an "admin login" and a contact phone number, was found in the HTML source code of the login page (**/login.jsp**). This information could assist attackers in reconnaissance or social engineering attacks.

Risk Rating: Medium

Impact

Disclosing such information may lead to:

- **Targeted Attacks:** Attackers may exploit the "admin login" reference to attempt unauthorized access.
- **Social Engineering:** The exposed phone number could be used to conduct phishing or other types of social engineering attacks.

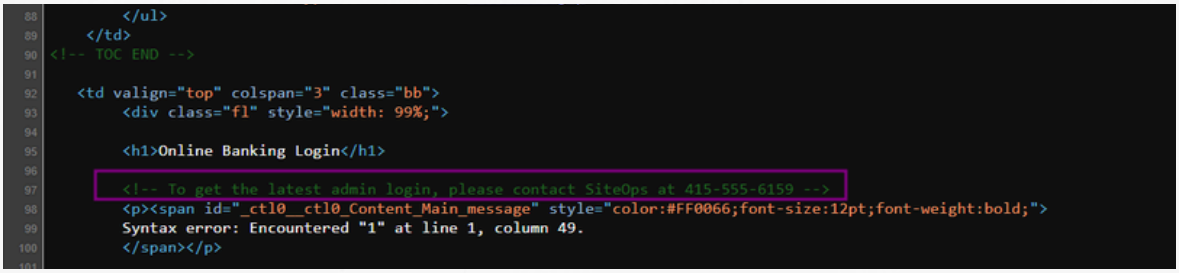
Recommendations

- **Remove sensitive comments** from the source code.
- Implement a **secure development process** to prevent inclusion of sensitive information in production code.
- Use **automated tools** to detect and remove sensitive comments during the build process.

Steps to Reproduce

1. Navigate to the login page: **http://altoro.testfire.net/login.jsp.**
2. **View the page source** and look for sensitive information.
3. The **following comment** is visible in the source code:

<!-- To get the latest admin login, please contact SiteOps at 415-555-6159 -->



This issue must be addressed immediately to prevent attackers from using exposed sensitive information for reconnaissance, social engineering, or unauthorized access attempts.

11. Information Disclosure - Executives & Management Details

Description

The application exposes detailed information about executives and management, including their names, titles, and roles, on the inside_executives.htm page. This data can be leveraged by attackers for phishing, social engineering, or targeted attacks.

Risk Rating: Medium

Impact

Exposing this information increases the risk of:

- **Phishing and Social Engineering Attacks:** Attackers may target employees, customers, or partners.
- **Impersonation:** Potential compromise of sensitive data or systems by impersonating executives.
- **Reputation Damage:** If the information is misused, it could harm the organization's public image.

Recommendations

- Restrict access to pages containing sensitive information, especially about executives.
- Implement **role-based access control** (RBAC) to limit viewing rights of internal organizational details.
- Avoid publishing unnecessary personal or professional information on public-facing pages.
- Train **employees** to recognize phishing and social engineering attempts based on publicly available information.
- Regularly review **public-facing content** to ensure no sensitive information is inadvertently exposed.

Steps to Reproduce

1. Access the **inside_executives.htm** page:
   http://altoro.testfire.net/index.jsp?content=inside_executives.htm
2. The page revealsdetailed information about executives and management, including:

- **Karl Fitzgerald** - Chairman & Chief Executive Officer
- **Rebecca Saddlemire** - President and Chief Operating Officer
- **Alison Debus** - Vice Chairman, Regional Banking Group

And other sensitive details.

This issue must be addressed immediately to prevent attackers from leveraging exposed sensitive details for phishing, social engineering, or targeted attacks.

Risk Assessment

Vulnerability	Risk Level
SQL Injection on Login Page	Critical
Insecure Direct Object Reference (IDOR)	Critical
Slowloris Denial of Service (DoS) Vulnerability	Critical
Reflected Cross-Site Scripting (XSS)	Critical
Lack of Anti-CSRF Tokens	Critical
Insecure Cookie Handling	Critical
Access to Sensitive Configuration File (web.xml)	High
Swagger UI Exposure	Medium
Missing Security Headers	Medium
Information Disclosure in Source Code	Medium
Information Disclosure - Executives & Management Details	Medium

Risk Rating Definitions

Risk Level	Description
Critical	Exploitation could result in severe consequences (e.g., data breach, system compromise).
High	Exploitation could lead to significant damage but may require more effort.
Medium	Moderate impact with limited likelihood of exploitation.
Low	Minimal impact with low likelihood of exploitation.

Recommendations

To mitigate the critical and high-risk vulnerabilities identified, the following actions are recommended:

- **SQL Injection on Login Page**
  - Replace dynamic SQL queries with parameterized queries or prepared statements.
  - Sanitize and validate user inputs to ensure only expected data formats are accepted.
  - Regularly test for SQL injection vulnerabilities using tools like OWASP ZAP or SQLmap.

- **Insecure Direct Object Reference (IDOR)**
  - Implement access controls to ensure users can only access resources they are authorized to.
  - Use indirect references (e.g., GUIDs or hashed identifiers) for sensitive objects rather than direct file paths or IDs.
  - Validate user input to ensure no unauthorized access to objects.

- **Slowloris Denial of Service (DoS) Vulnerability**
  - Configure rate-limiting to restrict the number of concurrent connections per IP address.
  - Deploy a Web Application Firewall (WAF) to detect and block Slowloris attacks.
  - Keep server software up to date and monitor logs for suspicious activity.

- **Reflected Cross-Site Scripting (XSS)**
  - Implement input validation and output encoding to prevent malicious scripts from executing.
  - Use Content Security Policy (CSP) headers to restrict the execution of inline scripts and unauthorized resources.

- **Lack of Anti-CSRF Tokens**
  - Implement anti-CSRF tokens in all forms and API requests to ensure that they are initiated by legitimate users.
  - Validate the origin of requests using the Referer or Origin headers to confirm they come from trusted sources.

- **Insecure Cookie Handling**
  - Mark sensitive cookies (e.g., JSESSIONID) with Secure and HttpOnly flags to prevent unauthorized access.
  - Periodically review and audit cookie configurations to ensure they comply with best practices.

- **Access to Sensitive Configuration File (web.xml)**
  - Restrict access to sensitive configuration files like web.xml using proper file permissions.
  - Ensure sensitive files are stored outside the web root directory to prevent direct access.

These actions will enhance security posture and mitigate identified vulnerabilities.

Long-Term Strategy

- **Conduct Regular Security Audits and Penetration Tests** :
  - Perform periodic assessments to identify and remediate vulnerabilities proactively.

- **Train Developers on Secure Coding Practices** :
  - Educate developers about secure coding techniques to prevent common vulnerabilities like SQL Injection, XSS, and CSRF.

- **Implement a Robust Incident Response Plan** :
  - Develop and test an incident response plan to handle potential breaches effectively.

- **Monitor and Log Security Events** :
  - Implement logging and monitoring solutions to detect and respond to suspicious activities.

- **Adopt a Defense-in-Depth Strategy** :
  - Layer security controls to minimize the risk of successful attacks.

- **Use Automated Dependency Scanning** :
  - Scan third-party libraries and dependencies for known vulnerabilities using tools like Snyk, Dependabot, or OWASP Dependency-Check.

- **Encrypt Sensitive Data** :
  - Encrypt sensitive data at rest and in transit using strong encryption algorithms (e.g., AES-256).

Contact Information

Penetration Testing Team : TeamName

- **Lead Tester** :
  - Name: Mickael Benlolo
  - Email: john.doe@example.com
  - Phone: +1-234-567-8901

Client Point of Contact

- **Name** : Client Name
- **Email** : client.email@example.com
- **Phone** : +1-123-456-7890