

WEB APPLICATION PENETRATION TEST REPORT

Professional Security Assessment – Training / Authorized Lab

Client / Target	Authorized Web Application / Altoro Matual (demo.testfire.net)
Assessment Type	Web Application Penetration Test
Assessment Status	Demonstration Report – Evidence placeholders clearly marked
Test Window	[19/08/2026 – 21/08/2026]
Classification	Confidential – Authorized Security Testing
Performed By	Nourhan Essam - Shorouk Ali - Aya Abdelkader - Merola Medhat - Hager Hassan
Reviewed By	Eng. Rehab Mostafa / Project Supervisor

1. Executive Summary

The assessment was designed to identify security weaknesses in the authorized web application and validate their practical security impact using controlled, non-destructive proof-of-concept testing. The report contains three representative findings across High, Medium, and Low risk levels.

ID	Vulnerability	Risk	Business Impact
WEB-001	Sql injection	HIGH	Unauthorized access to backend database, authentication bypass, and potential data manipulation
WEB-002	Reflected Cross-Site Scripting	HIGH	Arbitrary JavaScript execution in the victim's browser
WEB-003	Sensitive Data Exposure / PII Disclosure	HIGH	Exposure of confidential user financial data, including credit card numbers and account details.

2. Scope of Work

The scope of work defines the authorized activities performed during the web application security assessment.

- External and authenticated testing of the in-scope web application.
- Identification of application entry points, parameters, authentication and authorization controls.
- Manual validation of access-control weaknesses, input-validation issues, client-side injection, and security configuration issues.
- Controlled proof-of-concept validation using non-destructive payloads and test accounts.
- Documentation of affected URLs, evidence, impact, risk, and remediation guidance.

Out of scope: denial-of-service testing, destructive exploitation, persistence, attacks against unrelated third-party infrastructure, and collection of real personal/payment data.

3. Test Team / Responsibilities

Lead Tester	Nourhan Essam - Web Application Security Tester
Supporting Tester	Aya Abdelkader, Shorouk Ali, Hager Hassan, Mirola Medhat -Web Application Security Testers, responsible for reconnaissance, scanning, exploitation, and evidence collection
Reviewer	Eng. Rehab Mostafa - quality review and validation of findings
Client Contact	NTI - Penetration Testing & Application Security

4. Methodology

Testing follows an OWASP-style web application assessment approach: information gathering and attack-surface mapping using Nmap for service scanning and Gobuster for directory enumeration, identification of application entry points by merging endpoints into OWASP ZAP, manual testing of security controls, controlled exploitation/validation, impact assessment, and reporting. OWASP recommends that findings document the affected item, technical description, testing evidence, impact, remediation, and severity.

5. Information Gathering & Reconnaissance

1- Port & Service Scanning (Nmap):

```
Session Actions Edit View Help
(kali@kali)-[~]
$ nmap -Pn -sS -sV -sC -O -p- demo.testfire.net
Starting Nmap 7.99 ( https://nmap.org ) at 2026-08-19 13:35 -0400
Nmap scan report for demo.testfire.net (65.61.137.117)
Host is up (0.18s latency).
Not shown: 65532 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
80/tcp    open  http      Apache Tomcat/Coyote JSP engine 1.1
|_http-title: Altoro Mutual
|_http-server-header: Apache-Coyote/1.1
443/tcp   open  ssl/http  Apache Tomcat/Coyote JSP engine 1.1
|_ssl-date: 2026-08-19T17:44:27+00:00; 0s from scanner time.
|_ssl-cert: Subject: commonName=demo.testfire.net
| Subject Alternative Name: DNS:demo.testfire.net
| Not valid before: 2025-05-21T00:00:00
| Not valid after: 2026-06-21T23:59:59
|_http-server-header: Apache-Coyote/1.1
|_http-title: Altoro Mutual
8080/tcp   open  http      Apache Tomcat/Coyote JSP engine 1.1
|_http-server-header: Apache-Coyote/1.1
|_http-title: Altoro Mutual
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: VoIP adapter|bridge|general purpose
Running (JUST GUESSING): AT&T embedded (93%), Oracle Virtualbox (92%), Slirp (92%), QEMU (90%)
OS CPE: cpe:/o:oracle:virtualbox cpe:/a:danny_gasparovski:slirp cpe:/a:qemu:qemu
Aggressive OS guesses: AT&T BGW210 voice gateway (93%), Oracle Virtualbox Slirp NAT bridge (92%), QEMU user mode network gateway (90%)
No exact OS matches for host (test conditions non-ideal).

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 544.33 seconds
```

2-Directory & Path Enumeration (Gobuster):

```
(kali@kali)-[~]
$ gobuster dir -u http://demo.testfire.net -w /usr/share/wordlists/dirb/common.txt -x php,html,txt,jsp,bak -timeout 20s

Gobuster v3.8.2
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://demo.testfire.net
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.8.2
[+] Extensions: bak,php,html,txt,jsp
[+] Timeout: 20s
```

```

Starting gobuster in directory enumeration mode

Progress: 1202 / 27678 (4.34%) [ERROR] error on word @.txt: timeout occurred during the request
account.jsp      (Status: 302) [Size: 0] [→ /login.jsp]
admin            (Status: 302) [Size: 0] [→ /login.jsp]
aux             (Status: 200) [Size: 0]
bank            (Status: 302) [Size: 0] [→ /login.jsp]
com3            (Status: 200) [Size: 0]
com1            (Status: 200) [Size: 0]
com2            (Status: 200) [Size: 0]
con             (Status: 200) [Size: 0]
feedback.jsp    (Status: 200) [Size: 8351]
images          (Status: 302) [Size: 0] [→ /images/]
index.jsp       (Status: 200) [Size: 9227]
login.jsp       (Status: 200) [Size: 8360]
logout.jsp      (Status: 302) [Size: 0] [→ index.jsp]
lpt1            (Status: 200) [Size: 0]
lpt2            (Status: 200) [Size: 0]
notfound.jsp    (Status: 200) [Size: 6821]
nul             (Status: 200) [Size: 0]
pr              (Status: 302) [Size: 0] [→ /pr/]
prn             (Status: 200) [Size: 0]
search.jsp      (Status: 200) [Size: 6866]
static          (Status: 302) [Size: 0] [→ /static/]
subscribe.jsp    (Status: 200) [Size: 8358]
util            (Status: 302) [Size: 0] [→ /util/]
Progress: 27678 / 27678 (100.00%)

Finished

```

3- URL Scope Merging & Processing:

```

(kali@kali)-[~]
$ awk '{print $1}' gobuster_urls > gobuster_clean

(kali@kali)-[~]
$ cat gobuster_clean /media/sf_New_folder/zap-urls | sort -u > all_combined-urls
cat: /media/sf_New_folder/zap-urls: No such file or directory

(kali@kali)-[~]
$ ls /media/sf_New_folder
zap-urls.txt
finished

(kali@kali)-[~]
$ cat gobuster_clean /media/sf_New_folder/zap-urls.txt | sort -u > all_combined-urls

(kali@kali)-[~]
$ cp all_combined-urls /media/sf_New_folder/

```

4-OWASP ZAP Active Scan Results for Standard User Account:

The screenshot displays the OWASP ZAP interface during an Active Scan. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, and Help. The main window is titled 'Manual Explore' and features a sidebar on the left showing a tree view of sites and documents. The central pane shows the 'Manual Explore' screen with a URL to explore set to 'http://demo.testfire.net'. The bottom pane shows the 'Alerts' tab with a list of 24 alerts. The 'Cross Site Scripting (Reflected)' alert is selected, showing details such as URL, Risk (High), Confidence (Medium), and a description of the attack.

Manual Explore

This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP. The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

URL to explore:

Enable HUD: ☐

Explore your application:

Alerts (24)

- Cross Site Scripting (Reflected) (2)
- PII Disclosure (2)
- SQL Injection (3)
- Absence of Anti-CSRF Tokens (Systemic)
- Content Security Policy (CSP) Header Not Set (Systemic)
- Cross-Domain Misconfiguration
- Missing Anti-clickjacking Header (Systemic)
- Secure Pages Include Mixed Content (Including Scripts)
- Sub Resource Integrity Attribute Missing
- Cookie No HttpOnly Flag
- Cookie Without Secure Flag
- Cookie without SameSite Attribute (Systemic)
- Cross-Domain JavaScript Source File Inclusion
- HTTPS Content Available via HTTP (Systemic)

Cross Site Scripting (Reflected)

URL: <https://demo.testfire.net/sendFeedback>

Risk: **High**

Confidence: Medium

Parameter: name

Attack: `</p><script>alert(1);</script><p>`

Evidence: `</p><script>alert(1);</script><p>`

CWE ID: 79

WASC ID: 8

Source: Active (40012 - Cross Site Scripting (Reflected))

Input Vector: Form Query

Description: Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Other Info:

Alerts: 3 6 10 5 Main Proxy: localhost:8081 Current Status: 0

5-OWASP ZAP Active Scan Results for Admin User Account:

The screenshot displays the OWASP ZAP interface during an Active Scan. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, and Help. The main window is titled 'Manual Explore' and features a sidebar on the left showing a tree view of sites and documents. The central pane shows the 'Manual Explore' screen with a URL to explore set to 'http://demo.testfire.net'. The bottom pane shows the 'Alerts' tab with a list of 23 alerts. The 'Cross Site Scripting (Reflected)' alert is selected, showing details such as URL, Risk (High), Confidence (Medium), and a description of the attack.

Manual Explore

This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP. The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

URL to explore:

Enable HUD: ☐

Explore your application:

Alerts (23)

- Cross Site Scripting (Reflected) (2)
- GET: [https://demo.testfire.net/search.jsp\(query\)](https://demo.testfire.net/search.jsp(query))
- POST: [https://demo.testfire.net/sendFeedback\(\)\(file.com\)](https://demo.testfire.net/sendFeedback()(file.com))
- SQL Injection
- Absence of Anti-CSRF Tokens (Systemic)
- Content Security Policy (CSP) Header Not Set (Systemic)
- Cross-Domain Misconfiguration
- Missing Anti-clickjacking Header (Systemic)
- Secure Pages Include Mixed Content (Including Scripts)
- Sub Resource Integrity Attribute Missing
- Cookie No HttpOnly Flag
- Cookie Without Secure Flag
- Cookie without SameSite Attribute (Systemic)
- Cross-Domain JavaScript Source File Inclusion

Cross Site Scripting (Reflected)

URL: <https://demo.testfire.net/sendFeedback>

Risk: **High**

Confidence: Medium

Parameter: name

Attack: `</p><script>alert(1);</script><p>`

Evidence: `</p><script>alert(1);</script><p>`

CWE ID: 79

WASC ID: 8

Source: Active (40012 - Cross Site Scripting (Reflected))

Input Vector: Form Query

Description: Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Other Info:

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.

Alerts: 2 6 10 5 Main Proxy: localhost:8081 Current Status: 0

6. Detailed Findings

Vulnerability ID: 1	Sql Injection
Vulnerability Name	Sql injection
URL	https://demo.testfire.net/doLogin
Type	Sql injection / CWE 89
Risk Rating	HIGH
Affected Function	User Authentication / form login

Description

A SQL Injection vulnerability occurs when an application fails to properly sanitize or validate user-supplied input before incorporating it into a backend SQL query. This allows an attacker to inject arbitrary SQL commands into the database execution context.

Proof of Concept

ZAP' OR '1'='1--

Injecting ' OR '1'='1-- is a classic Authentication Bypass technique that tricks the database as follows:

- **ZAP:** treated by the server as the input username string.
- **Single Quote ('):** Closes the intended input string.
- **Condition (OR '1'='1'):** Forces the query logic to evaluate to TRUE.
- **Comment (--):** Truncates the query, completely bypassing the password check.



Figure 1 – Illustrative sql injection evidence.

Impact

Successful unauthorized access to user accounts, including administrative privileges, fully compromising system security and allowing the exposure, modification, or deletion of sensitive database records.

Recommendation

- **Use Prepared Statements (Parameterized Queries):** Ensure the database engine treats user input strictly as data, never as executable SQL code.
- **Input Validation & Sanitization:** Enforce strict allow-lists for user inputs.
- **Principle of Least Privilege:** Restrict database account permissions to limit the damage if a query is compromised

Vulnerability ID: 2	Reflected Cross-Site Scripting (XSS)
Vulnerability Name	Reflected Cross-Site Scripting (XSS)
URL	https://demo.testfire.net/search.jsp?query=%3C%2Fp%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cp%3E
Type	CWE-79 – Improper Neutralization of Input During Web Page Generation
Risk Rating	HIGH
Affected Parameter	searchTerms

Description

The application reflects attacker-controlled input into an HTML response without appropriate context-aware output encoding. The vulnerable context allows crafted markup to be interpreted by the browser as active content.

Proof of Concept

```
</p><script>alert(1);</script><p>
```

The payload is a non-destructive proof of concept. Successful execution of the alert confirms that attacker-controlled JavaScript can execute in the vulnerable page context.

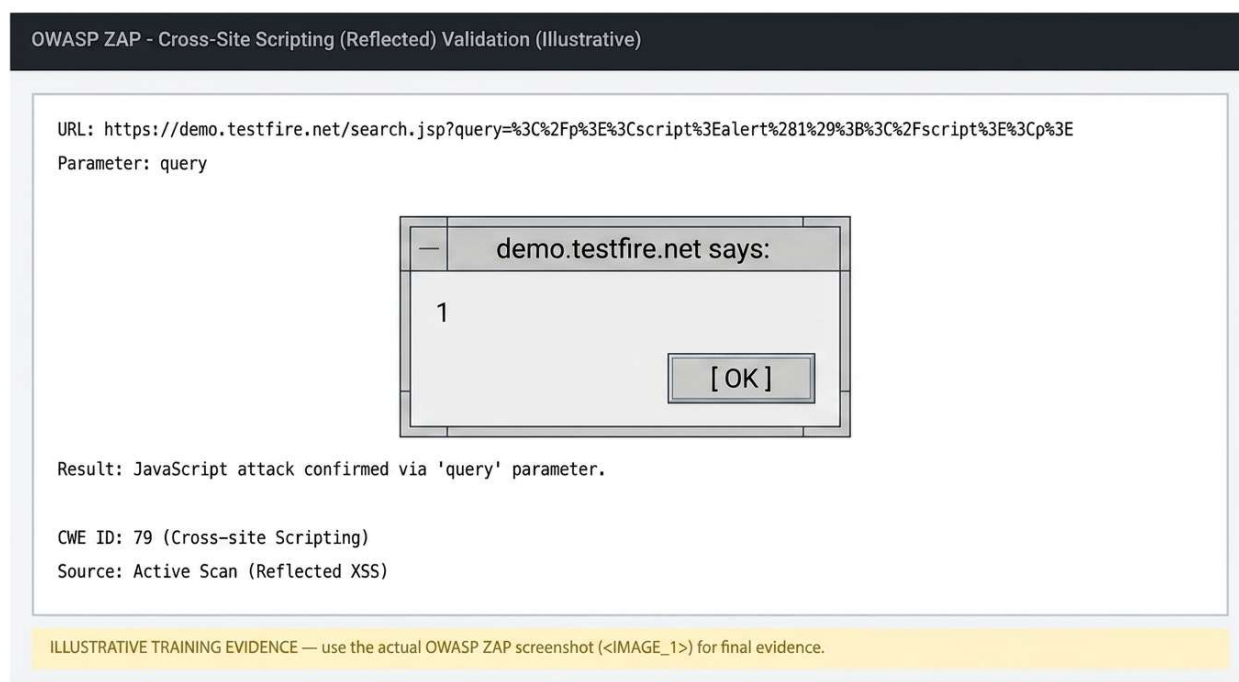


Figure 2 – Illustrative browser evidence.

Impact

An attacker can exploit this Reflected Cross-Site Scripting (Reflected XSS) vulnerability by tricking a victim into clicking a crafted URL containing malicious JavaScript. Since the target is a banking web application, the successful exploitation leads to:

Account Takeover / Session Hijacking: Accessing non-HTTPOnly sensitive cookies or session tokens, allowing the attacker to impersonate the victim and compromise their session.

Credential Harvesting (Phishing): Injecting malicious HTML/DOM elements (e.g., fake login prompts or session expiration popups) into the legitimate domain to steal user credentials or sensitive banking details.

Unauthorized Financial Actions: Executing actions on behalf of the authenticated victim (e.g., transferring funds, changing user details, or performing administrative tasks if the victim holds admin privileges).

Malicious Redirection: Redirecting users to external malicious domains to host drive-by download attacks or further phishing schemes.

Recommendation

- Apply context-aware output encoding to all untrusted data before placing it in HTML, attributes, JavaScript, CSS, or URLs.
- Avoid unsafe DOM sinks such as innerHTML when a text-only operation is sufficient.
- Use a restrictive Content Security Policy as defense in depth.
- Review all reflected parameters and test the same input in each output context.
- Add automated security tests for reflected and DOM-based XSS.

Vulnerability ID: 3	Security Misconfiguration – Directory Listing
Vulnerability Name	Personally identifiable information (pii)
URL	https://demo.testfire.net/bank/main.jsp
Type	Information disclosure / CWE-359
Risk Rating	HIGH
Affected Resource	/bank/main.jsp

Description

the application displays response data containing exposed **Personally Identifiable Information (PII)** and highly sensitive data, such as credit card numbers and Social Security Numbers (SSN). This sensitive information is rendered in plain text without proper masking, encryption, or access controls, leading to severe privacy violations, financial fraud risks, and non-compliance with regulatory security standards (e.g., PCI-DSS, GDPR).

Proof of Concept

Request:

GET /bank/main.jsp HTTP/1.1

Reviewing the HTTP request to /bank/main.jsp confirms that sending this request exposes sensitive user session cookies and returns unmasked PII data in the response.

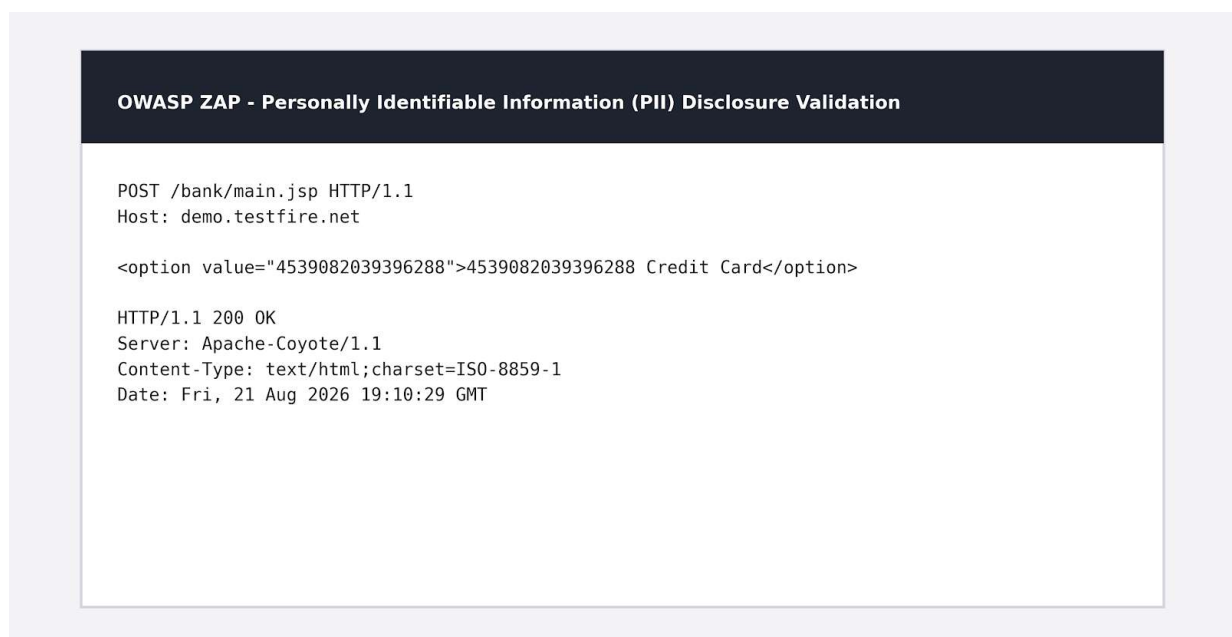


Figure 3 – Illustrative directory-pii evidence.

Impact

Unauthorised access to sensitive financial data—specifically Payment Card Information—exposes users to immediate financial fraud and unauthorized transactions. Furthermore, this compromise can lead to complete account takeover, identity theft, and severe reputational damage to the organization.

Recommendation

To prevent Personally Identifiable Information (PII) leakage in HTTP responses, implement the following remediation controls:

- **Data Masking & Redaction:** Ensure that sensitive information (such as credit card numbers, national IDs, and account numbers) is masked or partially redacted before returning it to the user interface (e.g., display only XXXX-XXXX-XXXX-6288).
- **Strict Attribute Filtering:** Modify application endpoints and JSP views to strictly return only necessary data fields, omitting full account details or sensitive credentials from HTML tags or <option> values.
- **Server-Side Authorization Checks:** Verify that the requesting user is fully authenticated and explicitly authorized to view the specific data being rendered in the response.
- **Response Header Security:** Enforce strict caching headers (e.g., Cache-Control: no-store, no-cache) to prevent sensitive HTTP responses from being cached on client side or intermediate proxies

7. Remediation Priority

Priority 1 – WEB-001 (HIGH): Remediate SQL Injection by enforcing parameterized queries (prepared statements) and input validation.

Priority 2– WEB-002 (HIGH): Mitigate XSS by applying context-aware output encoding and enforcing Content Security Policy (CSP).

Priority 3– WEB-003 (HIGH): Fix PII and Payment Card Data disclosure by implementing server-side data masking and authorization checks.

8. Retest Requirements

After remediation, the tester should reproduce each finding using the same test cases and confirm that database manipulation, unauthorized payment card data exposure, and malicious script execution are no longer possible. Any residual behavior should be documented as a retest finding.

9. Evidence & Reporting Notes

The report structure follows OWASP reporting guidance: clear scope, test parameters, findings summary, detailed findings, evidence, impact, remediation, and retest information.

All target identifiers, session values, and sensitive payment card details (PII) should be redacted in the final client-facing report. Evidence should be taken from the actual authorized target and should not include unrelated third-party data.

10. Disclaimer

This document is a professional reporting document created for security assessment purposes. The target names, URLs, HTTP responses, and evidence screenshots are documented solely to demonstrate valid vulnerabilities observed during the authorized security evaluation.