

**CENTER FOR DEVELOPMENT OF ADVANCE COMPUTING  
(C-DAC), THIRUVANANTHAPURAM, KERALA**

**A PROJECT ON  
“VAPT Report on testfire.net ”**

**SUBMITTED TOWARDS THE**



**PG-DCSF AUG 2024**

**BY  
Group No. – 07**

**MOHD HAMZA KAZI**

**PRN:240860940022**

**SWAYAM GUNDAWAR**

**PRN:240860940049**

**ADITI DIXIT**

**PRN:240860940016**

**ROSHAN MUNGANE**

**PRN:240860940038**

**SHELLY PANT**

**PRN:240860940043**

**Under The Guidance Of  
Mr. Shreedeeep A. L.**

# Acknowledgement

*We avail ourselves of this opportunity to express our deep sense of gratitude and wholehearted thanks to our guide, **Mr. Shreedip A. L.** for his valuable guidance, inspiration, and affectionate encouragement to embark on this project.*

*We also extend our gratitude to the esteemed faculty members of CDAC for their valuable suggestions and support throughout this endeavor.*

*We are deeply thankful to the **Head of our CDAC Center, Mr. Jayram P.** for their esteemed suggestions and encouragement, which have been a constant source of motivation for us to travel towards the successful completion of this project.*

*Lastly, we would like to express our heartfelt thanks to our teachers, parents, supporting technical staff, and friends who helped us directly or indirectly in our endeavor and contributed their support for the successful completion of this project.*

**Aditi Dixit**

**Shelly Pant**

**Roshan Mungane**

**Swayam Gundawar**

**Mohd Hamza Kazi**

**PGDCSF**

**(C-DAC Thiruvananthapuram)**

# Table of Contents

|  |          |
|--|----------|
| <b>List of Figures .....</b>                                       | <b>i</b> |
| <b>Abstract</b>  |          |
| <b>Chapter 1 INTRODUCTION .....</b>                                | <b>1</b> |
| 1.1 Motivation.....  | 1        |
| 1.2 Aim .....  | 1        |
| 1.3 Objective .....  | 2        |
| 1.4 Scope.....   | 2        |
| <b>Chapter 2 FUNDAMENTALS OF VAPT .....</b>                        | <b>3</b> |
| 2.1 What is VAPT? .....  | 3        |
| 2.2 Importance of Ethical Hacking .....                            | 3        |
| 2.3 Types of Penetration Testing .....                             | 3        |
| 2.3.1 Black Box Testing .....                                      | 3        |
| 2.3.2 White Box Testing .....                                      | 4        |
| 2.3.3 Gray Box Testing.....  | 4        |
| 2.4 Tools and Techniques Used in VAPT .....                        | 4        |
| 2.4.1 Vulnerability Scanning Tools .....                           | 4        |
| 2.4.2 Penetration Testing Tools .....                              | 4        |
| 2.4.3 Code Analysis Tools.....                                     | 4        |
| 2.4.4 Other Techniques .....                                       | 4        |
| 2.5 Common Vulnerabilities in Web Applications: OWASP Top 10 ..... | 5        |
| <b>Chapter 3 METHODOLOGIES .....</b>                               | <b>7</b> |
| 3.1 Phases of VAPT .....   | 7        |
| 3.1.1 Scope Definition .....                                       | 7        |
| 3.1.2 Information Gathering (Reconnaissance) .....                 | 8        |
| 3.1.3 Vulnerability Identification.....                            | 9        |
| 3.1.4 Exploitation.....  | 9        |
| 3.1.5 Post-Exploitation.....                                       | 10       |
| 3.1.6 Reporting.....   | 11       |

|   |    |
|---|----|
| <b>Chapter 4 ANALYSIS AND RECOMMENDATIONS</b> | 12 |
| Reconnaissance & Enumerations                 | 12 |
| 4.1 Results of Reconnaissance                 | 12 |
| N-map   | 12 |
| Whois Domain Tool                             | 13 |
| Go-Buster                                     | 15 |
| Nikto   | 16 |
| Nuclei  | 17 |
| 4.2 Vulnerability Analysis                    | 19 |
| 4.2.1 Critical Vulnerabilities                | 20 |
| IDOR (Indirect Object Reference)              | 20 |
| SQL Injection (Critical)                      | 22 |
| 4.2.2 High Risk Vulnerabilities               | 24 |
| Default Credentials                           | 24 |
| Login Brute force                             | 25 |
| 4.2.3 Medium Risk Vulnerabilities             | 28 |
| Click-Jacking                                 | 28 |
| URL Redirection                               | 29 |
| Cryptographic Failure                         | 31 |
| Cross-Site Scripting (XSS)                    | 33 |
| 4.2.4 Informative Vulnerabilities             | 36 |
| Information Leakage                           | 36 |
| Internal Server Error Display                 | 37 |
| <b>Chapter 5 CONCLUSION</b>                   | 39 |
| <b>Chapter 6 REFERENCES</b>                   | 12 |

## List of Figures

|           |                                      |    |
|-----------|--------------------------------------|----|
| Fig. 4.1  | Host Discovery.....                  | 12 |
| Fig. 4.2  | Stealth Scan.....                    | 12 |
| Fig. 4.3  | Stealth Scan Domain Info .....       | 13 |
| Fig. 4.5  | GoBuster directory enumeration.....  | 15 |
| Fig. 4.7  | Nuclei Scan .....                    | 17 |
| Fig. 4.8  | Nuclei Scan .....                    | 18 |
| Fig. 4.9  | IDOR.....                            | 20 |
| Fig. 4.11 | SQL Injection.....                   | 22 |
| Fig. 4.13 | Default Credential.....              | 24 |
| Fig. 4.15 | Burp-Suit Intruder .....             | 26 |
| Fig. 4.17 | Login brute-force .....              | 27 |
| Fig. 4.19 | Nikto scan Click-Jacking .....       | 28 |
| Fig. 4.20 | URL Redirection .....                | 29 |
| Fig. 4.21 | URL Redirection burp-suit.....       | 30 |
| Fig. 4.23 | URL Redirection POC .....            | 30 |
| Fig. 4.24 | Cryptographic failure burp-suit..... | 32 |
| Fig. 4.25 | XSS .....                            | 33 |
| Fig. 4.27 | HTML Injection .....                 | 34 |
| Fig. 4.31 | Information Leakage .....            | 36 |
| Fig. 4.31 | Internal server error display .....  | 38 |

# CVSS Table for Vulnerabilities

| Vulnerability          | Severity      | CVSS Base Score |
|------------------------|---------------|-----------------|
| IDOR                   | Critical      | 9.1             |
| SQL Injection          | Critical      | 9.8             |
| Default Credentials    | High          | 8.8             |
| Brute Force            | High          | 7.5             |
| Click-Jacking          | Medium        | 6.1             |
| URL Redirection        | Medium        | 5.3             |
| Cryptographic Failures | Medium        | 6.5             |
| XSS                    | Medium        | 5.4             |
| Information Leakage    | Informational | 3.7             |
| Internal Server Error  | Informational | 4.0             |

## **ABSTRACT**

This report presents the findings of a Vulnerability Assessment and Penetration Testing (VAPT) conducted on a demo banking web application. The objective of the assessment was to identify security vulnerabilities and evaluate the overall security posture of the application. The scope of the testing encompassed the web application's front-end, back-end, and authentication mechanisms. Tools such as Burp Suite, Nmap, and OWASP ZAP were employed to detect vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), and insecure authentication practices. The assessment revealed critical, high, and medium-risk vulnerabilities, accompanied by actionable recommendations to mitigate these risks. This report emphasizes the importance of addressing these vulnerabilities to enhance the application's security and protect against potential threats.

## Chapter 1

# INTRODUCTION

In today's digital-first era, the proliferation of advanced technologies and interconnected systems has brought both unprecedented opportunities and heightened security challenges. As organizations increasingly rely on digital platforms for operations, communication, and data management, the risk of cyber threats, vulnerabilities, and breaches has surged. To address these risks, a proactive and systematic approach to identifying, assessing, and mitigating vulnerabilities is essential.

This Vulnerability Assessment and Penetration Testing (VAPT) project focuses on evaluating the security posture of [testfire.net](https://testfire.net) a web application used as a testbed for security analysis. By simulating real-world attack scenarios, the project aims to uncover weaknesses and demonstrate the importance of robust security measures in mitigating cyber threats.

### 1.1 Motivation

The increasing frequency and sophistication of cyberattacks pose a significant threat to the confidentiality, integrity, and availability of digital systems. Organizations must prioritize the identification and remediation of vulnerabilities to protect their assets and maintain stakeholder trust. This project is motivated by the need to understand how vulnerabilities in web applications can be exploited and to explore effective ways to enhance their defenses. The project also serves as a hands-on opportunity to apply theoretical knowledge of cybersecurity to a practical scenario, contributing to a deeper understanding of the challenges and solutions in securing digital ecosystems.

### 1.2 Aim

The primary aim of this project is to perform a comprehensive Vulnerability Assessment and Penetration Testing (VAPT) of [testfire.net](https://testfire.net) to identify security vulnerabilities, evaluate their potential impact, and recommend actionable measures to mitigate risks.



### 1.3 Objective

- To systematically identify vulnerabilities in the web application's architecture, components, and configurations.
- To simulate real-world attack scenarios and analyze potential exploitation paths.
- To evaluate the effectiveness of existing security measures and identify areas for improvement.
- To provide detailed recommendations and remediation strategies for mitigating identified vulnerabilities.
- To enhance understanding of VAPT processes and promote the adoption of security best practices for securing web applications.

### 1.4 Scope of the Project

This section outlines the boundaries and extent of the Vulnerability Assessment and Penetration Testing (VAPT) conducted for [testfire.net](https://testfire.net)

- Assessment of vulnerabilities in the web application's architecture, functionalities, and configurations.
- Focused testing on areas prone to exploitation, such as authentication, input validation, and session management.
- Usage of automated and manual testing methodologies to identify and analyze potential vulnerabilities.
- Exclusion of server-side and hardware-level vulnerabilities beyond the application environment.

## Chapter 2

# FUNDAMENTALS OF VAPT

## 2.1 What is VAPT?

Vulnerability Assessment and Penetration Testing (VAPT) is a systematic process of evaluating an application, system, or network for vulnerabilities that could be exploited by attackers. VAPT combines two key practices:

- **Vulnerability Assessment:** Identifies potential vulnerabilities in the target system through automated tools and manual analysis.
- **Penetration Testing:** Simulates real-world attack scenarios to exploit identified vulnerabilities, demonstrating their potential impact.

The primary objective of VAPT is to assess the security posture of the target system and provide actionable recommendations for mitigating identified risks

## 2.2 Importance of Ethical Hacking

Ethical hacking involves authorized attempts to identify and fix security vulnerabilities in a system. Unlike malicious hackers, ethical hackers work within legal and ethical boundaries to enhance system security. Ethical hacking is crucial for:

- **Proactive Defense:** Identifying vulnerabilities before malicious actors can exploit them.
- **Regulatory Compliance:** Meeting security standards like ISO 27001, PCI DSS, and GDPR.
- **Risk Mitigation:** Reducing the risk of financial, reputational, and operational damage.
- **Building Trust:** Ensuring stakeholders and customers that their data is secure.

## 2.3 Types of Penetration Testing

Penetration testing can be categorized into three primary types based on the tester's knowledge and access level:

### 2.3.1 Black Box Testing

- **Definition:** The tester has no prior knowledge of the system's architecture or internal workings.
- **Focus:** Simulates an external attacker's perspective.
- **Strengths:** Effective for testing external defenses.

- **Limitations:** Limited coverage of internal vulnerabilities.

### 2.3.2 White Box Testing

- **Definition:** The tester has complete knowledge of the system, including source code, architecture, and configurations.
- **Focus:** Provides a comprehensive evaluation of both internal and external vulnerabilities.
- **Strengths:** High accuracy and detailed analysis.
- **Limitations:** Time-intensive and require extensive resources.

### 2.3.3 Gray Box Testing

- **Definition:** The tester has partial knowledge of the system.
- **Focus:** Balances the depth of white-box testing and the real-world simulation of black-box testing.
- **Strengths:** Efficient and provide focused testing.
- **Limitations:** May miss some vulnerabilities without complete access.

## 2.4 Tools and Techniques Used in VAPT

### 2.4.1 Vulnerability Scanning Tools

- **Examples:** Nessus, OpenVAS, Qualys.
- **Purpose:** Automated identification of vulnerabilities like misconfigurations, outdated software, and weak passwords.

### 2.4.2 Penetration Testing Tools

- **Examples:** Metasploit, Burp Suite, Nmap.
- **Purpose:** Simulates attack scenarios to exploit vulnerabilities and assess their impact.

### 2.4.3 Code Analysis Tools

- **Examples:** SonarQube, Checkmarx.
- **Purpose:** Identifies vulnerabilities in the source code during development.

### 2.4.4 Other Techniques

- **Manual Testing:** Identifying vulnerabilities through expert analysis and custom attack vectors.

- **Social Engineering:** Testing human factors by simulating phishing attacks and other manipulative techniques.
- **Network Mapping:** Analyzing network topology and identifying potential attack paths.

## 2.5 Common Vulnerabilities in Web Applications: OWASP Top 10

The OWASP Top 10 is a globally recognized list of the most critical security risks affecting web applications. It provides developers, security professionals, and organizations with a clear understanding of common vulnerabilities to prioritize their mitigation efforts. The current OWASP Top 10 includes:

1. **Broken Access Control:** Weak enforcement of access policies allows unauthorized users to access sensitive data or functions.  
**Example:** A user changes the URL from */user/profile* to */admin/panel* and gains unauthorized admin access because access controls are not enforced.
2. **Cryptographic Failures:** Inadequate protection of sensitive information during storage or transmission can lead to data breaches.  
**Example:** Storing passwords in plain text instead of hashing them, allowing attackers to read sensitive user data if the database is compromised.
3. **Injection:** Vulnerabilities exploited by injecting malicious inputs, such as SQL, OS, or LDAP injections, into the application.  
**Example:** A login form accepts input without validation, and an attacker enters *admin'--* as the username in an SQL query, bypassing authentication.
4. **Insecure Design:** Lack of sufficient security controls during the software design phase leads to inherent vulnerabilities.  
**Example:** An application doesn't enforce strong password requirements, making it easier for attackers to guess or brute-force passwords.
5. **Security Misconfiguration:** Misconfigured servers, frameworks, or applications expose sensitive data and increase attack surfaces.  
**Example:** Leaving the default admin username and password on a database server, allowing attackers to log in easily.

6. **Vulnerable and Outdated Components:** Using outdated software components with known vulnerabilities compromises the application's integrity.

**Example:** Using an old version of a library with a known vulnerability that allows remote code execution.

7. **Identification and Authentication Failures:** Weak or flawed authentication mechanisms allow unauthorized access to systems and data.

**Example:** An application allows users to log in without verifying email addresses, making it easier for attackers to impersonate legitimate users.

8. **Software and Data Integrity Failures:** Exploitation of insecure software updates or reliance on untrusted data sources introduces malicious elements.

**Example:** A software update is downloaded from an unverified source, and attackers inject malware into the update process.

9. **Security Logging and Monitoring Failures:** Insufficient or ineffective logging and monitoring prevent timely detection of security incidents.

**Example:** An attacker repeatedly tries different passwords on a login page, but no alerts are triggered because logging isn't implemented.

10. **Server-Side Request Forgery (SSRF):** Exploitation of server functionality to make unauthorized requests to internal or external systems.

**Example:** An attacker tricks a server into making a request to *http://internal-service/admin* by manipulating a URL parameter.

## Chapter 3

# METHODOLOGIES

This chapter describes the systematic approach employed during the Vulnerability Assessment and Penetration Testing (VAPT) process. It outlines the phases involved, methodologies applied, and tools leveraged to identify and analyze security vulnerabilities. Each phase was conducted following the industry's best practices and ethical guidelines.

## 3.1 Phases of VAPT

### 3.1.1 Scope Definition

- **Objective:** To clearly define the boundaries and focus areas of the assessment, ensuring alignment with the organizational goals and compliance requirements.
- **Description**
  - Begin by identifying critical assets to be tested, such as web applications, networks, servers, and endpoints.
  - Exclude assets that are outside the organization's operational purview or could lead to unintended consequences if tested.
  - Ensure clear communication with stakeholders to define the purpose and limitations of the VAPT process.
  - Draft and finalize a Rules of Engagement (RoE) document that includes testing hours, approved tools, and emergency contact details.
  - Obtain written authorization to ensure the testing process remains compliant with legal and organizational policies.
- **Outcome:**

A detailed scope document that outlines all in-scope and out-of-scope systems, ensuring a focused and legally compliant assessment.

### 3.1.2 Information Gathering (Reconnaissance)

- Reconnaissance is the first phase of the Vulnerability Assessment and Penetration Testing (VAPT) lifecycle and is often referred to as the "information-gathering phase." In this stage, a security tester or attacker collects as much information as possible about the target system, application, or network to identify potential entry points for an attack.
- **Description:**
  - **Passive Reconnaissance:**
    - Utilize publicly available data sources to gather information without directly interacting with the target.
    - Examine WHOIS records to determine domain ownership details, contact information, and domain registration dates.
    - Analyze DNS records for subdomain enumeration and email server configurations.
    - Use search engines and OSINT tools to uncover exposed sensitive information like credentials or development files.
  - **Active Reconnaissance:**
    - Conduct network scans using tools like Nmap to identify open ports, services, and operating systems.
    - Use tools like Shodan to discover publicly accessible devices and configurations.
    - Employ enumeration techniques to identify software versions, frameworks, and security patches.
- **Tools Used:** WHOIS lookup, Nmap, Shodan, Google Dorking.
- **Outcome:**
  - A comprehensive understanding of the target environment, including open ports, exposed services, and potential entry points.

- Detailed findings to guide the vulnerability identification phase.

### 3.1.3 Vulnerability Identification

- **Objective:** To identify weaknesses and vulnerabilities in the target systems using automated tools and manual testing techniques.
- **Description:**
  - Perform automated scans with tools like Nessus and OpenVAS to identify misconfigurations, unpatched software, and known vulnerabilities.
  - Test for common web vulnerabilities, such as SQL injection, Cross-Site Scripting (XSS), and insecure deserialization, using Burp Suite.
  - Conduct manual verification of tool-generated findings to eliminate false positives and confirm critical vulnerabilities.
  - Cross-check vulnerabilities against industry-standard frameworks such as OWASP Top 10 and CVSS scoring.
  - Validate the impact of each vulnerability by examining access levels, data exposure, and system behavior.
- **Tools Used:** Nessus, OpenVAS, Burp Suite, Nikto.
- **Outcome:**
  - A prioritized list of vulnerabilities with a risk rating to inform further exploitation attempts.
  - Verified evidence of each vulnerability to ensure the validity of findings.

### 3.1.4 Exploitation

- **Objective:** To validate the exploitability of identified vulnerabilities by simulating potential attack scenarios.
- **Description:**
  - Develop proof-of-concept (PoC) exploits for confirmed vulnerabilities to demonstrate their impact.



- Use Metasploit to automate the exploitation of vulnerabilities, such as remote code execution and privilege escalation.
  - Perform brute-force attacks on exposed login interfaces using tools like Hydra to test password strength and authentication mechanisms.
  - Ensure strict adherence to ethical guidelines by conducting exploitation in isolated or sandboxed environments.
- **Tools Used:** Metasploit, SQLMap, Hydra, Exploit-DB.
- **Outcome:**
  - Evidence of successful exploitation, such as screenshots of compromised systems or command execution.
  - Insights into potential damage that could occur if vulnerabilities were exploited by malicious actors.

### 3.1.5 Post-Exploitation

- **Objective:** To assess the extent of impact and identify methods attackers could use to maintain persistence.
- **Description:**
  - Evaluate the level of access gained through exploitation, including access to sensitive files, databases, and administrative controls.
  - Simulate data exfiltration to understand the potential for sensitive information leakage.
  - Test persistence mechanisms, such as adding backdoor users, modifying scripts, or planting malware.
  - Perform lateral movement analysis to identify risks of attackers compromising additional systems within the network.
- **Tools Used:** Cobalt Strike, PowerShell scripts, manual configuration analysis.

- **Outcome:**

- A detailed assessment of post-exploitation risks and recommendations for containment.
- Insights into the potential scale of a breach and the effectiveness of current monitoring mechanisms.

### 3.1.6 Reporting

- **Objective:** To compile all findings, methodologies, and recommendations into a comprehensive report.

- **Description:**

- Document the methodologies followed, including the tools used and observations made during each phase.
- Provide a prioritized list of vulnerabilities, supported by screenshots and PoC outputs.
- Include actionable recommendations for each vulnerability, categorized into short-term fixes and long-term strategies.
- Ensure compliance with reporting standards and align findings with organizational goals and regulatory requirements.

- **Tools Used:** Microsoft Word, report generators (e.g., Nessus, Burp Suite).

- **Outcome:**

- A professional report that serves as a roadmap for remediation and future security improvements.

## Chapter 4

# ANALYSIS AND RECOMMENDATIONS

## Reconnaissance & Enumerations

- **Common Tools and Techniques**
  - **DNS Enumeration:** Tools like **dig**, **nslookup**, or **Recon-ng** to gather domain-related data.
  - **Port Scanning:** Tools like **Nmap** or **Zenmap** to identify open ports and services.
  - **Website Analysis:** Tools like **Burp Suite**, **OWASP ZAP**, or **Wappalyzer** to analyze the technologies used on a website.
  - **OSINT (Open-Source Intelligence):** Using resources like **Shodan**, **Censys**, or public databases to collect information.

## 4.1 Results of Reconnaissance

### N-map

- **Host Discovery**

Host discovery is the process of identifying active hosts (devices) within a network.

```
(kali㉿kali)-[~]  
$ nmap -sn testfire.net  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-11 01:48 EST  
Nmap scan report for testfire.net (65.61.137.117)  
Host is up (0.00067s latency).  
Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

Fig. 4.1: Host Discovery

- **Stealth Scan**

```
(kali㉿kali)-[~]  
$ nmap -sS -sV testfire.net  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-11 01:52 EST  
Nmap scan report for testfire.net (65.61.137.117)  
Host is up (0.028s latency).  
Not shown: 997 filtered tcp ports (no-response)  
PORT      STATE SERVICE VERSION  
80/tcp    open  http      Apache Tomcat/Coyote JSP engine 1.1  
443/tcp   open  ssl/http  Apache Tomcat/Coyote JSP engine 1.1  
8080/tcp  open  http      Apache Tomcat/Coyote JSP engine 1.1  
  
Service detection performed. Please report any incorrect results at  
Nmap done: 1 IP address (1 host up) scanned in 44.92 seconds
```

Fig. 4.2: Stealth Scan

- Identify open ports on the target system while avoiding detection by intrusion detection systems (IDS) and firewalls.

### Observations:

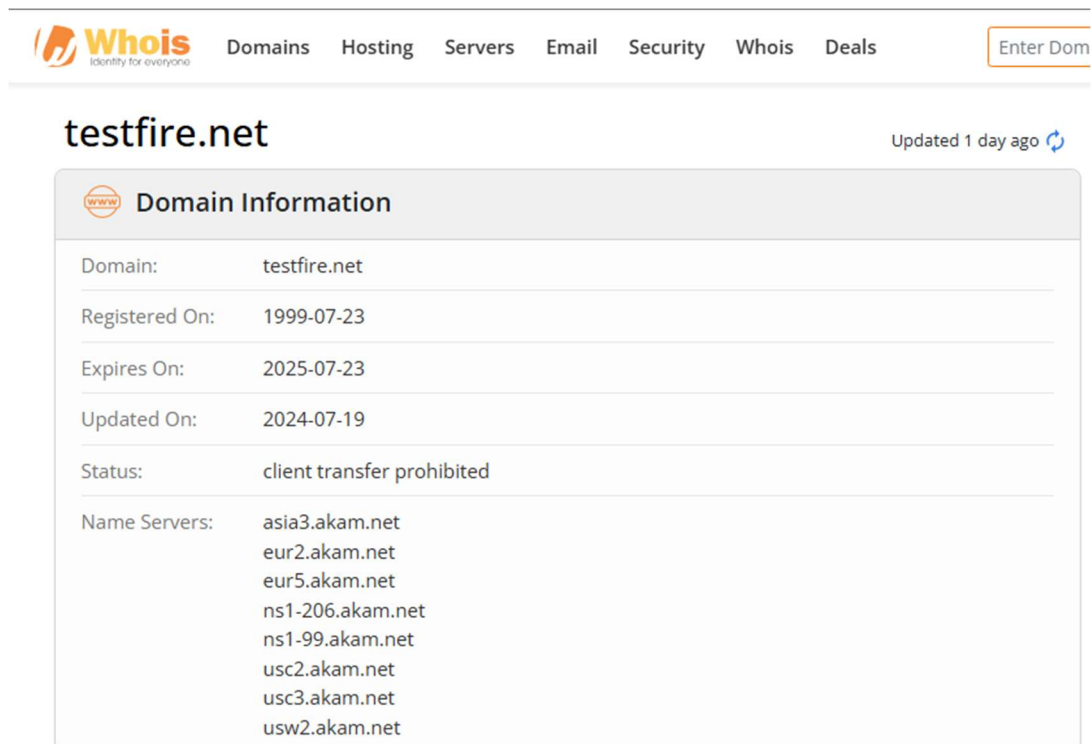
- **Port 8080 Open**
  - **Redundancy:** Port 8080 duplicates the service on port 80 and is not required for application functionality.
  - **Threats:** Exposes an unnecessary attack surface, risks exploitation, and may transmit unencrypted data.

### Recommendations:

Disable port 8080, block external access via firewalls, and monitor for redundant open ports.

### Whois Domian tool

- To gather publicly available information about the domain **testfire.net** including registrar details, domain ownership, and DNS information.
  - To identify administrative and technical contacts.
  - To gather information about the domain's registration and expiration dates.
  - To analyze DNS and registrar details for security assessment.



The screenshot shows the Whois tool interface for the domain testfire.net. The tool's header includes navigation links for Domains, Hosting, Servers, Email, Security, Whois, and Deals, along with a search bar labeled 'Enter Dom'. The domain name 'testfire.net' is entered in the search bar, and the results are displayed below. The results are organized into a table with the following information:

| Domain Information |  |
|--------------------|--|
| Domain:            | testfire.net   |
| Registered On:     | 1999-07-23   |
| Expires On:        | 2025-07-23   |
| Updated On:        | 2024-07-19   |
| Status:            | client transfer prohibited   |
| Name Servers:      | asia3.akam.net<br>eur2.akam.net<br>eur5.akam.net<br>ns1-206.akam.net<br>ns1-99.akam.net<br>usc2.akam.net<br>usc3.akam.net<br>usw2.akam.net |

**Fig. 4.3: Stealth Scan Domain Info**

| ® Registrar Information |                             |
|-------------------------|-----------------------------|
| Registrar:              | CSC Corporate Domains, Inc. |
| IANA ID:                | 299                         |
| Abuse Email:            | domainabuse@cscglobal.com   |
| Abuse Phone:            | 8887802723                  |

**Fig. 4.4: Stealth Scan Domain Info**

#### Observation:

- **Registrar and Status:** The domain is registered with **CSC Corporate Domains, Inc.**, created on **July 23, 1999**, and expires on **July 23, 2025**. Privacy protection is enabled, and transfer is restricted.
- **DNS and Hosting:** Uses robust Akamai name servers but lacks **DNSSEC**, and the IP (**65.61.137.117**) is shared with 5 other sites.

#### Recommendations:

- **Enable DNSSEC:** Implement DNS Security Extensions to prevent DNS spoofing and cache poisoning attacks.
- **Evaluate Hosting Setup:** Consider isolating the domain on a dedicated IP to reduce shared hosting risks.

#### Go-Buster

- **GoBuster** is a fast and efficient tool used for **directory and file brute-forcing** on web servers. It helps discover hidden files, directories, virtual hosts, and DNS subdomains by sending requests to a target system and analyzing the responses.

#### Common Usage Scenarios

- Locating admin panels or sensitive files on web servers.
- Enumerating DNS subdomains for further attack vectors.
- Testing security configurations to ensure no unintended exposure of resources.

```
(kali㉿kali)-[~]
$ gobuster dir --url http://65.61.137.117 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url:          http://65.61.137.117
[+] Method:       GET
[+] Threads:      10
[+] Wordlist:      /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent:    gobuster/3.6
[+] Timeout:      10s

Starting gobuster in directory enumeration mode

/images      (Status: 302) [Size: 0] [→ /images/]
/admin       (Status: 302) [Size: 0] [→ /login.jsp]
/static      (Status: 302) [Size: 0] [→ /static/]
/pr          (Status: 302) [Size: 0] [→ /pr/]
/util        (Status: 302) [Size: 0] [→ /util/]
/bank        (Status: 302) [Size: 0] [→ /login.jsp]
/con         (Status: 200) [Size: 0]
```

Fig. 4.5: GoBuster directory enumeration

#### Observations:

- **/images, /static:** Likely host public resources (low risk).
- **/admin, /bank:** Redirect to login pages; potential for unauthorized access (high risk).
- **/util, /pr:** Internal directories; possible sensitive data or misconfigurations (medium to high risk).
- **/con:** Accessible; potential for sensitive file exposure (medium to high risk).

#### Recommendations:

- **Secure Access:** Restrict **/admin, /bank, /util, and /pr** with authentication, MFA, and IP whitelisting.
- **Harden Endpoints:** Use rate limiting, CAPTCHAs, and a WAF for login pages.
- **Content Review:** Audit **/static** and **/images** to remove unused or sensitive files.
- **Inspect /con:** Identify and secure exposed files or remove the directory if unnecessary.
- **General:** Monitor logs, remove unused directories, and perform a full vulnerability scan.

## Nikto

- **Nikto** is an open-source web server scanner used to identify vulnerabilities and misconfigurations in web servers. It is designed to perform comprehensive testing by analyzing web servers for dangerous files, outdated versions, and security issues.

```
- Nikto v2.5.0
+ 0 host(s) tested
(kali@kali)-[~]
$ nikto -h 65.61.137.117
- Nikto v2.5.0

+ Target IP:      65.61.137.117
+ Target Hostname: 65.61.137.117
+ Target Port:    80
+ Start Time:     2025-02-11 02:39:53 (GMT-5)

+ Server: Apache-Coyote/1.1
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content using content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OPTIONS: Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS .
+ HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web server.
+ HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server.
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
```

Fig. 4.6: GoBuster directory enumeration

## Observation

- **Web Server Identified:** Apache-Coyote/1.1 running on IP **65.61.137.117** (Port 80).
- **Headers Missing:**
  - **X-Frame-Options:** Missing, making the site vulnerable to clickjacking attacks.
  - **X-Content-Type-Options:** Missing, potentially allowing MIME-type mismatches.
- **HTTP Methods Allowed:**
  - **PUT:** Could allow unauthorized users to upload files to the server.
  - **DELETE:** Could allow unauthorized deletion of files.
- Junk HTTP methods return a valid response, indicating possible misconfigurations.
- **No CGI Directories Found:** No CGI directories were detected during the scan.

## Weaknesses Identified

- **Missing Security Headers:**
  - Increases the risk of **clickjacking** and content-sniffing attacks.

- **Dangerous HTTP Methods:**

- **PUT** and **DELETE** methods pose a significant risk of unauthorized file manipulation.

## Recommendations

- **Implement Security Headers:**

- Add **X-Frame-Options** to prevent clickjacking.
- Add **X-Content-Type-Options** to enforce proper content type handling.

- **Restrict HTTP Methods:**

- Disable unnecessary methods like **PUT** and **DELETE** on the web server.

- **Server Hardening:**

- Validate and limit supported HTTP methods to **GET**, **HEAD**, and **POST**.
- Review and correct any junk method responses.

## Nuclei

```
(root@kali) ~# nuclei -u testfire.net

Nuclei
v3.3.7
projectdiscovery.io

[WRN] Found 2 templates with runtime error (use -validate flag for further examination)
[INF] Current nuclei version: v3.3.7 (outdated)
[INF] Current nuclei-templates version: v10.1.2 (latest)
[WRN] Scan results upload to cloud is disabled.
[INF] New templates added in latest release: 52
[INF] Templates loaded for current scan: 7657
[WRN] Loading 380 unsigned templates for scan. Use with caution.
[INF] Executing 7277 signed templates from projectdiscovery/nuclei-templates
[INF] Targets loaded for current scan: 1
[INF] Running httpx on input host
[INF] Found 1 URL from httpx
[INF] Templates clustered: 1698 (Reduced 1598 Requests)
[INF] Using Interactsh Server: oast.online
[swagger-api] [http] [info] https://testfire.net/swagger/index.html [paths="/swagger/index.html"]
[waf-detect:apache-generic] [http] [info] https://testfire.net
[deprecated-tls:tls-1.1] [ssl] [info] testfire.net:443 ["tls11"]
[tls-version] [ssl] [info] testfire.net:443 ["tls10"]
[weak-cipher-suites:tls-1.0] [ssl] [low] testfire.net:443 ["tls10 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA"]
[tls-version] [ssl] [info] testfire.net:443 ["tls11"]
[deprecated-tls:tls-1.0] [ssl] [info] testfire.net:443 ["tls10"]
[weak-cipher-suites:tls-1.1] [ssl] [low] testfire.net:443 ["tls11 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA"]
```

Fig. 4.7: Nuclei Scan



```

[tlsversion] [ssl] [info] testfire.net:443 ["tls1"]
[deprecated-tls:tls_1.0] [ssl] [info] testfire.net:443 ["tls10"]
[weak-cipher-suites:tls-1.1] [ssl] [low] testfire.net:443 ["tls11 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA"]
[tlsversion] [ssl] [info] testfire.net:443 ["tls12"]
[rdap-whois:lastChangeDate] [http] [info] https://rdap.verisign.com/net/v1/domain/testfire.net ["2024-07-19T05:14:16Z"]
[rdap-whois:expirationDate] [http] [info] https://rdap.verisign.com/net/v1/domain/testfire.net ["2025-07-23T13:52:32Z"]
[rdap-whois:nameServers] [http] [info] https://rdap.verisign.com/net/v1/domain/testfire.net ["ns1-99.AKAM.NET","USC2.AKAM.NET"]
[rdap-whois:secureDNS] [http] [info] https://rdap.verisign.com/net/v1/domain/testfire.net ["false"]
[rdap-whois:status] [http] [info] https://rdap.verisign.com/net/v1/domain/testfire.net ["client transfer prohibited"]
[rdap-whois:registrationDate] [http] [info] https://rdap.verisign.com/net/v1/domain/testfire.net ["1999-07-23T13:52:32Z"]
[http-missing-security-headers:cross-origin-resource-policy] [http] [info] https://testfire.net
[http-missing-security-headers:strict-transport-security] [http] [info] https://testfire.net
[http-missing-security-headers:cross-origin-embedder-policy] [http] [info] https://testfire.net
[http-missing-security-headers:cross-origin-opener-policy] [http] [info] https://testfire.net
[http-missing-security-headers:x-content-type-options] [http] [info] https://testfire.net
[http-missing-security-headers:x-permitted-cross-domain-policies] [http] [info] https://testfire.net
[http-missing-security-headers:referrer-policy] [http] [info] https://testfire.net
[http-missing-security-headers:clear-site-data] [http] [info] https://testfire.net
[http-missing-security-headers:content-security-policy] [http] [info] https://testfire.net
[http-missing-security-headers:permissions-policy] [http] [info] https://testfire.net
[http-missing-security-headers:x-frame-options] [http] [info] https://testfire.net
[apache-detect] [http] [info] https://testfire.net ["Apache-Coyote/1.1"]
[spf-record-detect] [dns] [info] testfire.net ["v=spf1 mx/24 -all"]
[txt-fingerprint] [dns] [info] testfire.net ["v=spf1 mx/24 -all"]
[caa-fingerprint] [dns] [info] testfire.net
[dmARC-detect] [dns] [info] _dmarc.testfire.net ["v=DMARC1; p=reject; fo=1; rua=mailto:dmarc_rua@emaildefense.proofpoint.com"]
[nameserver-fingerprint] [dns] [info] testfire.net ["eur2.akam.net","usc3.akam.net","eur3.akam.net","ns1-99.akam.net"]
[ssl-issuer] [ssl] [info] testfire.net:443 ["Sectigo Limited"]
[mismatched-ssl-certificate] [ssl] [low] testfire.net:443 ["CN: demo.testfire.net"]
[ssl-dns-names] [ssl] [info] testfire.net:443 ["demo.testfire.net","altoromstual.com"]
root@kali:~/home/kali

```

Fig. 4.8: Nuclei Scan

## Observations

- **Deprecated TLS Protocols Detected:**

- TLS 1.0 and TLS 1.1 are still supported, which are outdated and insecure.
- Weak cipher suites like TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA are being used.

- **Missing Security Headers:**

- **X-Frame-Options:** Absent, making the site vulnerable to clickjacking.
- **Strict-Transport-Security (HSTS):** Missing, exposing the site to MITM attacks.
- Other missing headers: **X-Content-Type-Options**, **Content-Security-Policy**, **Referrer-Policy**, **Permissions-Policy**.

- **Misconfigured SSL Certificate:**

- SSL certificate is mismatched, with a Common Name (CN) of demo.testfire.net.

- **RDAP Information:**

- **Secure DNS Disabled:** RDAP reports the Secure DNS (DNSSEC) status as "false."
- Registrar information matches prior WHOIS findings (CSC Corporate Domains, Inc.).

- **Other Vulnerabilities:**

- SPF and DMARC misconfigurations were detected, indicating potential email spoofing risks.

### Recommendations:

- **Upgrade TLS Protocols and Cipher Suites:**

- Disable TLS 1.0 and 1.1; enforce TLS 1.2 or higher.
- Replace weak ciphers with modern ones like AES-GCM or ChaCha20.

- **Implement Security Headers:**

- Add **X-Frame-Options**, **Strict-Transport-Security**, **Content-Security-Policy**, and other missing headers to mitigate web-based attacks.

- **Fix SSL Certificate Mismatch:**

- Update the SSL certificate to match the domain name `testfire.net`.

- **Harden Email Security:**

- Correct SPF, DKIM, and DMARC configurations to prevent email spoofing.

## 4.2 Vulnerability Analysis

Burp Suite is a comprehensive web vulnerability scanner widely used for penetration testing and security assessments. Alongside Burp Suite, we leveraged manual analysis and other specialized tools during the assessment to ensure thorough coverage and accuracy. These tools enabled us to detect, analyze, and validate vulnerabilities in the target application.

This section presents the findings of our analysis, categorizing the vulnerabilities based on their severity Critical, High, Medium, and Informational. Each vulnerability is described in detail, including its potential impact on the target system and the associated risks.

### Features of Burp Suite

- **Intercepting Proxy:** Intercepts and modifies HTTP/HTTPS traffic in real time.
- **Vulnerability Scanner:** Detects issues like SQL injection and XSS.
- **Intruder:** Automates attacks like brute force and fuzzing.
- **Repeater:** Replays and modifies HTTP requests for testing.
- **Extensibility:** Customizable with plugins via the BApp Store.

#### 4.2.1 Critical Vulnerabilities:

##### IDOR (Indirect Object Reference)

Testfire.net exposes account details through direct numerical references in the 'listAccounts' parameter (e.g., 800003). This allows attackers to modify the parameter and access data from other user accounts.

##### Risk Assessment:

- **Likelihood:** High – An attacker can easily change the URL parameter to access another user's account.
- **Impact:** Critical – Attackers can view and manipulate sensitive banking account information of other users.

##### Tools Used:

- Manual Analysis

The screenshot shows a web browser with the URL `testfire.net/bank/showAccount?listAccounts=800002`. The page displays the 'Account History - 800002 Savings' interface. The 'Balance Detail' table shows the ending and available balances as of 2/5/25 2:54 AM. The '10 Most Recent Transactions' table lists deposits and withdrawals.

| Balance Detail                      |                |                 |
|-------------------------------------|----------------|-----------------|
| 800002 Savings                      | Select Account | Amount          |
| Ending balance as of 2/5/25 2:54 AM |                | -\$398983675.58 |
| Available balance                   |                | -\$398983675.58 |

| 10 Most Recent Transactions |             |                 |
|-----------------------------|-------------|-----------------|
| Date                        | Description | Amount          |
| 2025-02-05                  | Deposit     | \$1000.00       |
| 2025-02-05                  | Withdrawal  | -\$200000000.00 |
| 2025-02-05                  | Withdrawal  | -\$200000000.00 |
| 2025-02-05                  | Deposit     | \$1000.00       |
| 2024-12-11                  | Deposit     | \$8888.00       |
| 2024-12-11                  | Deposit     | \$8888.00       |

Fig. 4.9: IDOR

Not secure testfire.net/bank/showAccount?listAccounts=800005

oMutual

PERSONAL SMALL BUSINESS

### Account History - 800005 Checking

| Balance Detail                      |                |
|-------------------------------------|----------------|
| 800005 Checking                     | Select Account |
| Ending balance as of 2/5/25 2:53 AM | \$25.00        |
| Available balance                   | \$25.00        |

| 10 Most Recent Transactions |             |           |
|-----------------------------|-------------|-----------|
| Date                        | Description | Amount    |
| 2018-06-11                  | Deposit     | \$10.00   |
| 2018-05-15                  | Deposit     | \$10.00   |
| 2018-04-14                  | Deposit     | \$10.00   |
| 2018-01-10                  | Withdrawal  | -\$100.00 |

**Fig. 4.10: IDOR****Recommendations:**

- **Implement Proper Access Controls:**
  - Use server-side authorization checks to ensure users can only access data they are authorized to view.
- **Use Indirect References:**
  - Replace direct numerical identifiers (e.g., account IDs) with indirect references, such as UUIDs or tokens, that cannot be guessed or manipulated.
- **Validate User Inputs:**
  - Ensure all incoming parameters are validated against the logged-in user's permissions.
- **Audit and Monitor Access Logs:**
  - Regularly audit logs to detect any unauthorized access attempts and respond to suspicious activities promptly.
- **Conduct Regular Security Testing:**
  - Perform periodic vulnerability assessments and penetration tests to identify and fix access control issues proactively.

## SQL Injection (Critical)

The login functionality of Testfire.net is vulnerable to SQL Injection, allowing attackers to execute malicious SQL queries. This can be exploited to bypass authentication and gain unauthorized administrative access to the system.

### Risk:

- **likelihood:** High – The vulnerability can be easily exploited using common SQL payloads in the login input fields.
- **Impact:** Critical – Attackers can compromise the confidentiality, integrity, and availability of the application by accessing or manipulating sensitive data in the database.

### Tools Used:

- Burp Suite
- SQLMap

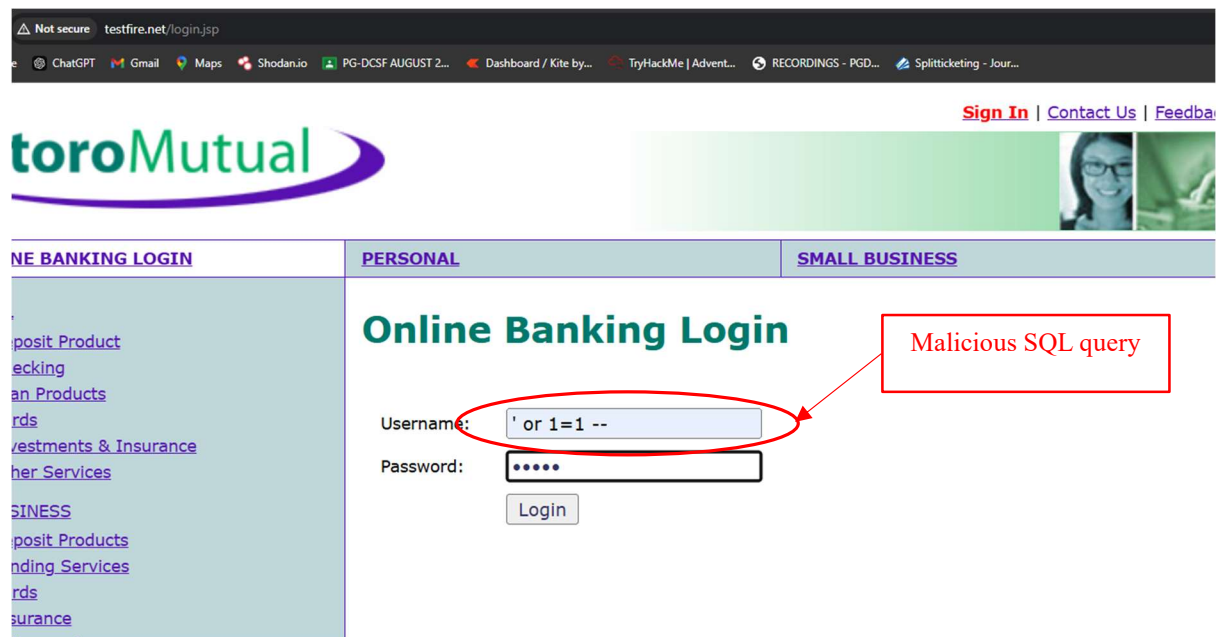


Fig. 4.11: SQL Injection

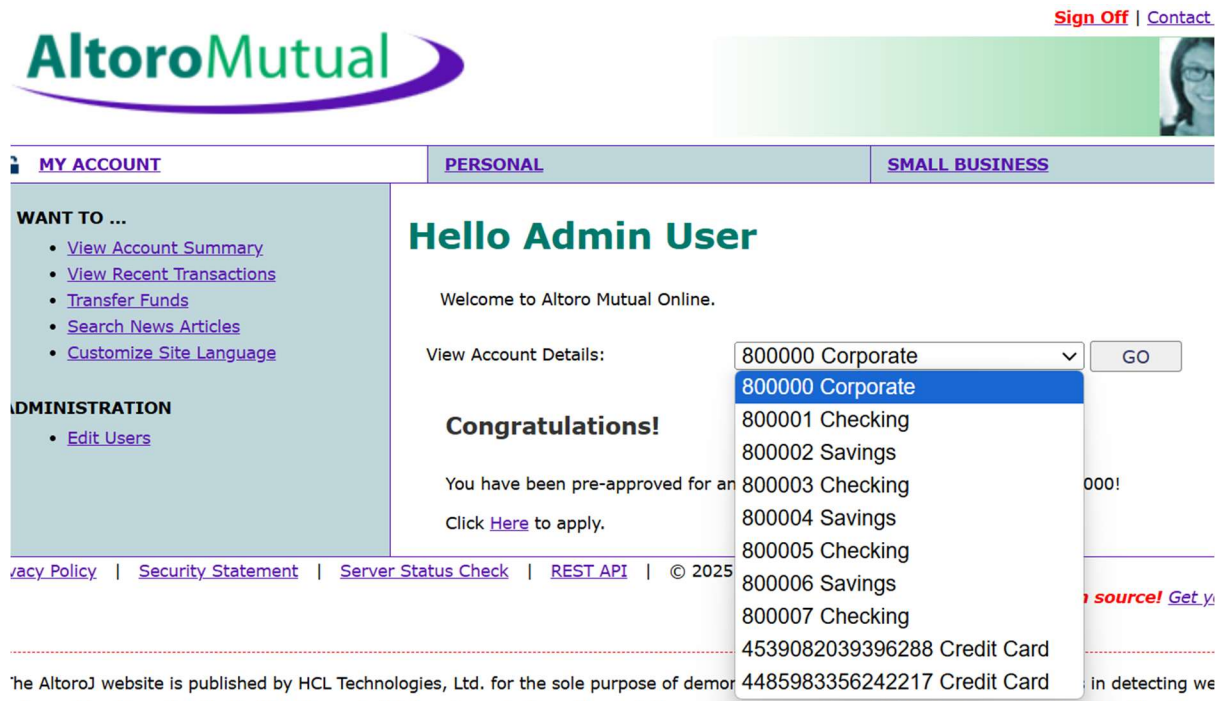


Fig. 4.12: SQL Injection

#### Recommendations:

- **Use Parameterized Queries:** Prevent SQL Injection by using prepared statements or parameterized queries in all database interactions.
- **Sanitize User Inputs:** Implement robust input validation to reject unexpected or harmful data inputs.
- **Employ Stored Procedures:** Leverage stored procedures for database operations to limit direct SQL execution.
- **Detailed Error Handling:** Conceal database error messages from users to avoid leaking sensitive information.
- **Database Privilege Management:** Restrict the database user's permissions to only essential operations.
- **Routine Security Testing:** Conduct regular security audits and penetration tests to identify and mitigate vulnerabilities.

## 4.2.2 High Risk Vulnerabilities:

### Default Credentials:

1. Default credentials are pre-set usernames and passwords often used during application setup.
2. If left unchanged, they pose a significant security risk as attackers can exploit them to gain unauthorized access.
3. Exploiting default credentials can lead to administrative account compromise, exposing sensitive data and critical system functionality.

### Observation

The login functionality of Testfire.net at <http://altoro.testfire.net/login.jsp> is vulnerable due to the presence of default administrator credentials (`admin:admin`). These credentials are publicly available, enabling attackers to gain unauthorized administrative access to the system.

### Risk:

- **Likelihood:** High – The default credentials are well-known and easy to exploit.
- **Impact:** Critical – Attackers with admin access can manipulate or delete sensitive data, compromise user accounts, and disrupt the application's integrity.

### Tools Used:

- Manual Exploitation

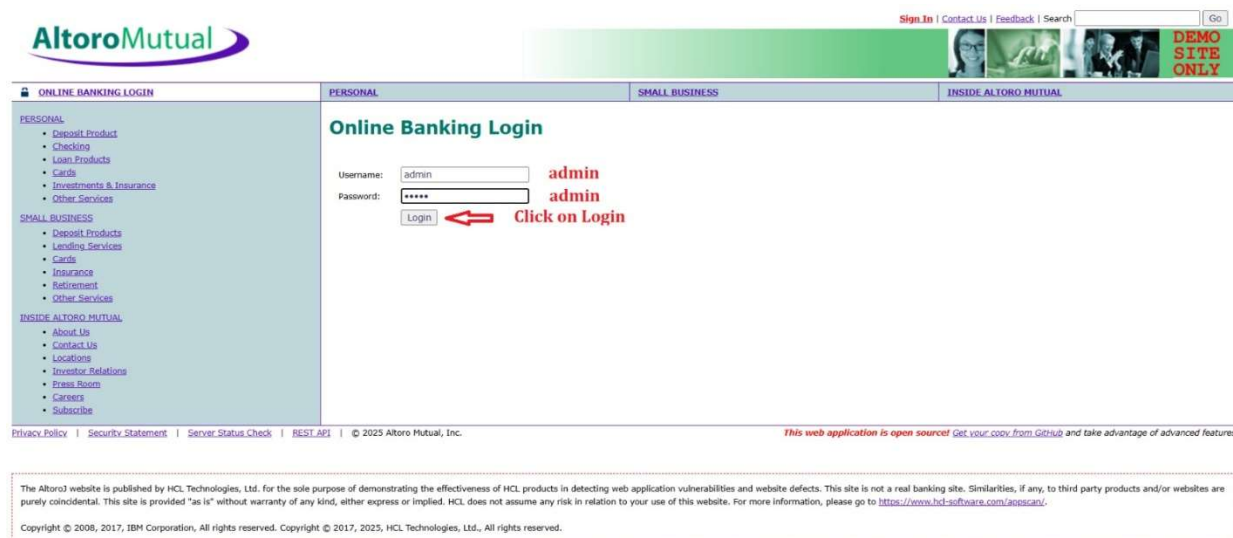
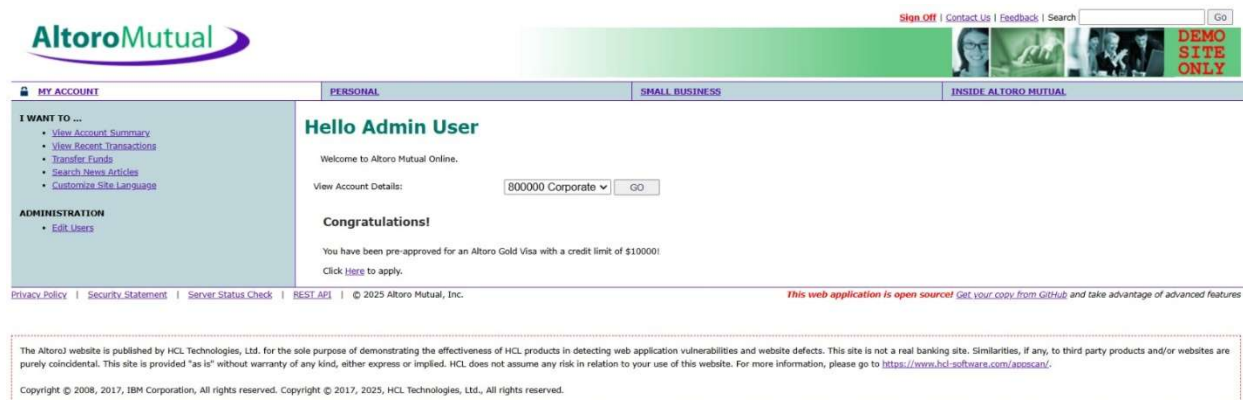


Fig. 4.13: Default Credential





**Fig. 4.14: Default Credential**

### Recommendations:

1. **Change Default Credentials Immediately:** Ensure all default passwords are replaced with strong, unique credentials upon deployment.
2. **Implement Multi-Factor Authentication (MFA):** Add an extra layer of security to administrative accounts.
3. **Enforce Strong Password Policies:** Mandate the use of complex passwords for all accounts, including admin.
4. **Disable Default Accounts:** Remove or deactivate any default accounts if they are not required.
5. **Conduct Regular Security Audits:** Monitor and audit login attempts for signs of unauthorized access.

### Login Brute force:

The login functionality of Testfire.net does not implement sufficient security mechanisms to prevent brute-force attacks. Attackers can exploit this vulnerability to repeatedly guess credentials using automated tools, potentially compromising user accounts.

### Risk:

- **Likelihood:** High – The lack of account lockout or rate limiting makes the system vulnerable to brute-force attacks.
- **Impact:** Critical – Successful attacks can lead to unauthorized access, exposing sensitive user data and system functionality.



## Tools Used:

- Hydra
- Burp Suite Intruder

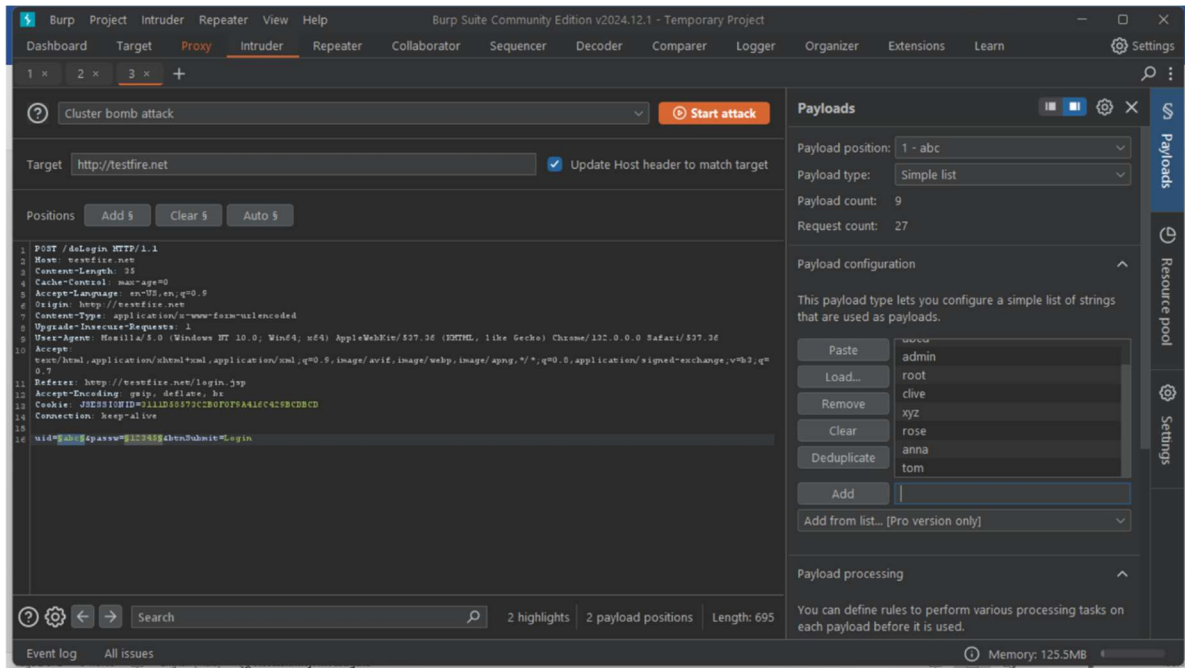


Fig. 4.15: Burp-Suit Intruder

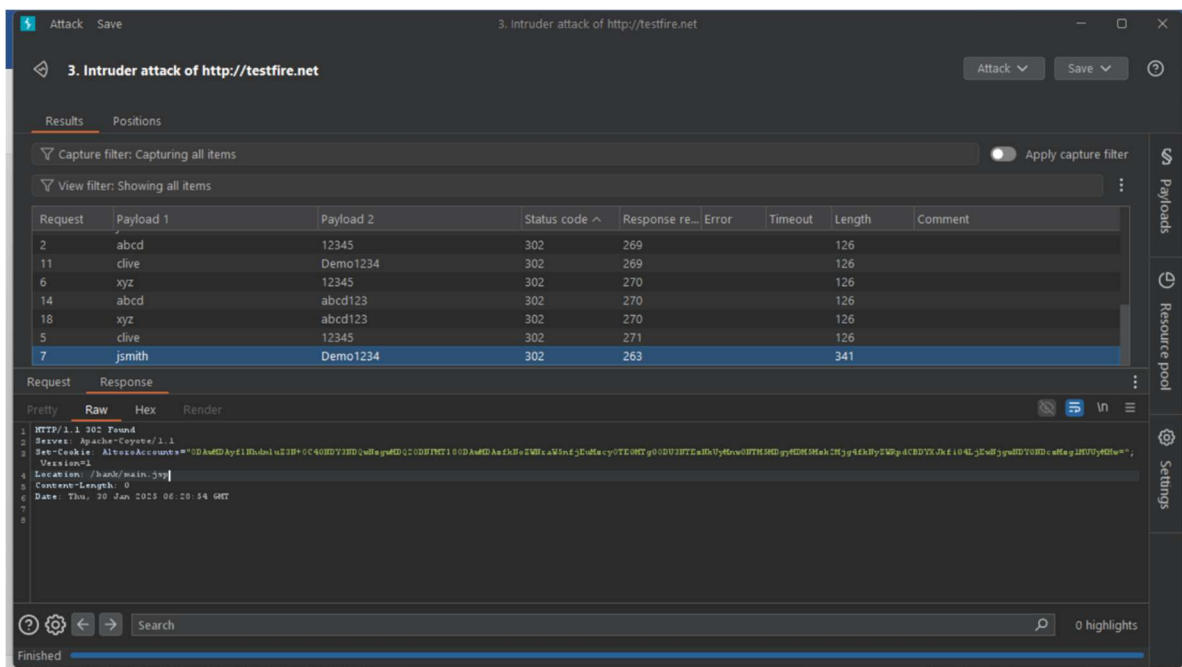


Fig. 4.16: Burp-Suit Intruder



Fig. 4.17: Login brute-force

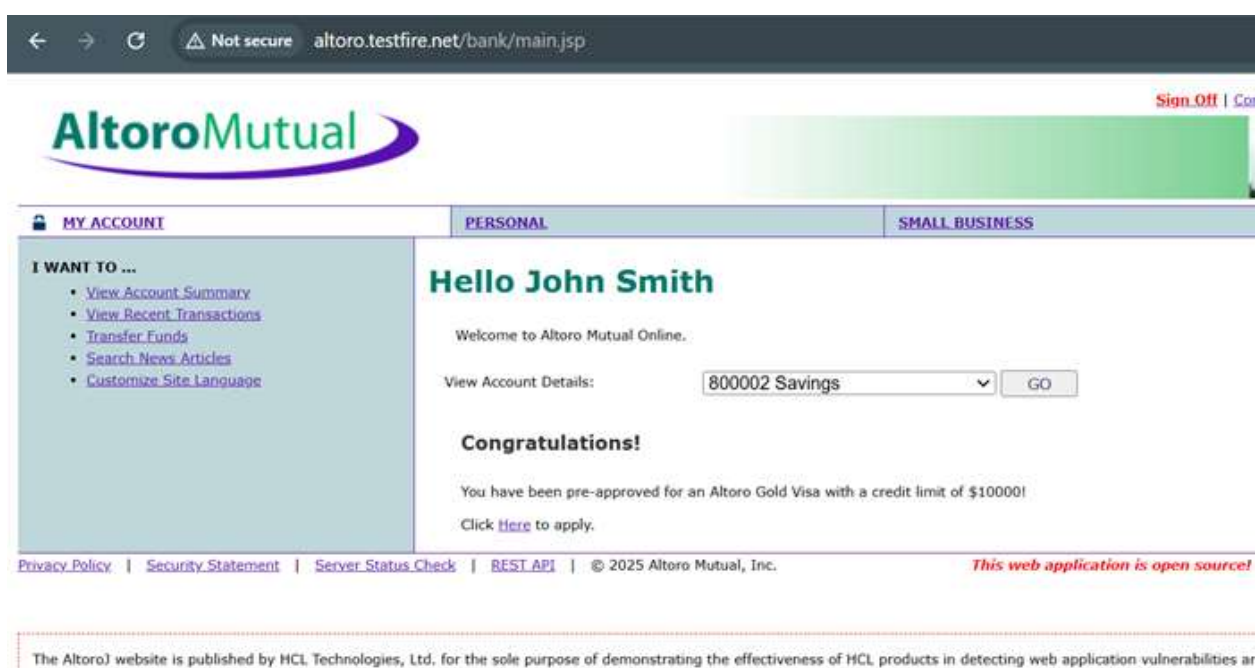


Fig. 4.18: Login brute-force

### Recommendations:

1. **Implement Account Lockout:** Temporarily lock accounts after a predefined number of failed login attempts to prevent automated attacks.
2. **Introduce CAPTCHA:** Use CAPTCHA mechanisms to distinguish between human users and automated bots.
3. **Rate Limiting:** Limit the number of login attempts from a single IP address within a specific time frame.

4. **Enable Logging and Monitoring:** Track login attempts and notify administrators of suspicious activity, such as multiple failed attempts.
5. **Promote Strong Passwords:** Enforce a strong password policy to reduce the effectiveness of brute-force attacks.
6. **Use Multi-Factor Authentication (MFA):** Add an extra layer of security for user and admin accounts to mitigate unauthorized access.

#### 4.2.3 Medium Risk Vulnerabilities:

##### Click-Jacking

The application is vulnerable to clickjacking due to the absence of `X-Frame-Options` or `Content Security Policy` headers. Attackers can embed the application in an `iframe` to trick users into performing unintended actions.

##### Risk:

- **Likelihood:** Medium – Attack success depends on user interaction with a malicious site.
- **Impact:** Medium – Unauthorized actions can harm user accounts or data integrity.

**Tool Used:** Nikto

```
(kali㉿kali)-[~]
$ nikto -h testfire.net
- Nikto v2.5.0

+ Target IP:          65.61.137.117
+ Target Hostname:    testfire.net
+ Target Port:        80
+ Start Time:         2025-02-05 04:16:20 (GMT-5)

+ Server: Apache-Coyote/1.1
+ /: The anti-clickjacking X-Frame-Options header is not present.
e-Options
```

**Fig. 4.19: nikto scan Click-Jacking**

## Recommendations:

1. **Set X-Frame-Options Header:** Use `DENY` or `SAMEORIGIN` in the HTTP response header to prevent the application from being embedded in iframes on unauthorized domains.
2. **Implement Content Security Policy (CSP):** Define a CSP to restrict the resources that can be loaded and frame sources to trusted domains only.
3. **User Education:** Inform users about the risks of interacting with unknown or untrusted websites.
4. **Secure Forms:** Use additional mechanisms, like CSRF tokens, to validate user actions within the application.
5. **Periodic Security Testing:** Regularly test for clickjacking vulnerabilities using automated tools or manual analysis.

## URL Redirection

The application has an open URL redirection vulnerability in the `content` parameter of the `customize.jsp` page. This allows attackers to redirect users to malicious websites, facilitating phishing attacks or malware distribution.

### Risk:

- **Likelihood:** Medium – The vulnerability is easy to exploit by crafting a malicious URL.
- **Impact:** Medium – Successful exploitation can lead to loss of user trust and potential compromise of user credentials or data.

### Tools Used:

- Burp Suite
- OWASP ZAP
- Manual Analysis

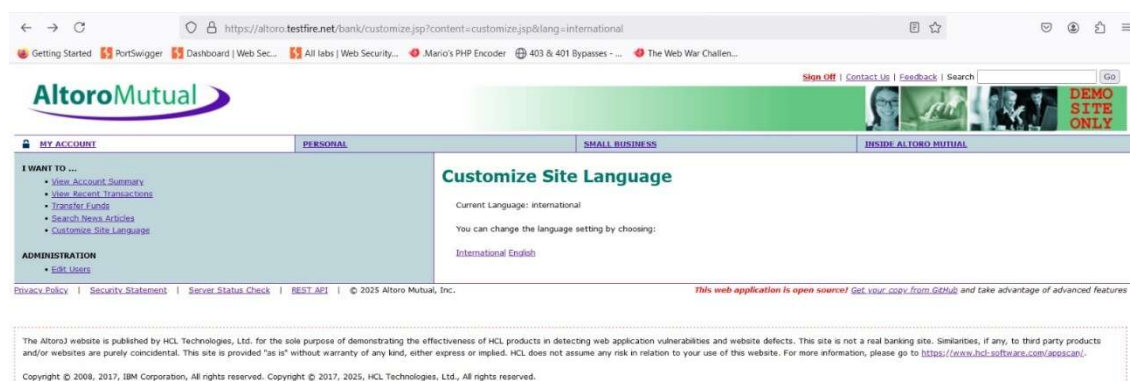


Fig. 4.20: URL Redirection

```

Pretty    Raw    Hex
1 GET /bank/customize.jsp?content=customize.jsp&lang=international HTTP/1.1
2 Host: altoro.testfire.net
3 Cookie: JSESSIONID=B5AEA280D64D06713479C1A61AE98271; AltoroAccounts=0DAwMDAwfkNvcnB
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Fire
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Priority: u=0, i
14 Te: trailers
15 Connection: close
16
17

```

Fig. 4.21: URL Redirection burp-suit

```

GET /bank/customize.jsp?content=https://google.com/&lang=international HTTP/1.1
Host: altoro.testfire.net
Cookie: JSESSIONID=B5AEA280D64D06713479C1A61AE98271; AltoroAccounts=0DAwMDAwfkNvcnB
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefo
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: close

```

Fig. 4.22: URL Redirection burp-suit

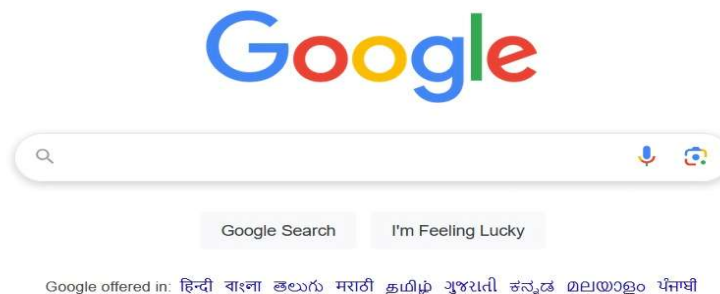


Fig. 4.23: URL Redirection POC

**Recommendations:**

1. **Validate Redirect Parameters:** Strictly validate and sanitize user input for URL redirection parameters. Allow only pre-approved URLs or domains.
2. **Use a Whitelist:** Implement a whitelist of allowed redirection targets within the application to restrict external redirects.
3. **Warn Users of Redirects:** Display a warning or confirmation message before redirecting users to external sites.
4. **Implement CSP Headers:** Use Content Security Policy (CSP) headers to restrict the loading of malicious content from untrusted domains.
5. **Monitor Logs:** Keep track of redirection events in server logs to identify and investigate suspicious activity.
6. **Educate Users:** Encourage users to verify the URL before clicking on links, especially those provided in emails or messages.

**Cryptographic Failure**

Sensitive data, such as user credentials or authentication tokens, is transmitted in plaintext over an insecure HTTP connection, making it vulnerable to interception and unauthorized access.

**Risk:**

- **Likelihood:** Medium – Attackers can exploit this using tools like packet sniffers in public or insecure networks.
- **Impact:** Medium – Intercepted sensitive information, such as login credentials or personal details, can lead to data breaches.

**Tools Used:**

- Wireshark
- Burp Suite
- Manual Analysis

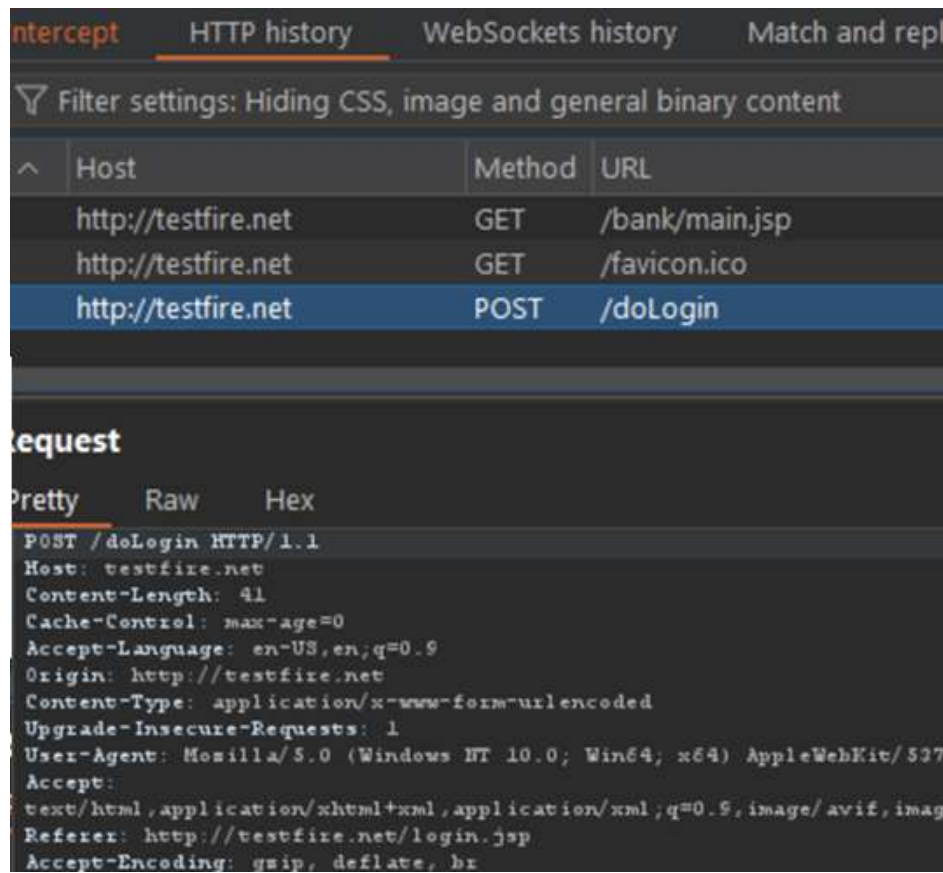


Fig. 4.24: Cryptographic failure burp-suit

#### Recommendations:

1. **Enforce HTTPS:** Ensure all communications between the client and server use HTTPS by implementing TLS (Transport Layer Security).
2. **Use Secure Certificates:** Obtain a valid SSL/TLS certificate from a trusted Certificate Authority (CA) and configure it correctly.
3. **Redirect HTTP to HTTPS:** Configure the server to automatically redirect HTTP requests to HTTPS to ensure secure connections.
4. **Implement HSTS:** Use HTTP Strict Transport Security (HSTS) headers to prevent users from making insecure HTTP connections.
5. **Regularly Update Certificates:** Monitor and renew SSL/TLS certificates before they expire to maintain secure connections.
6. **Avoid Weak Ciphers:** Use strong cryptographic algorithms and disable deprecated protocols, such as TLS 1.0 and TLS 1.1, to enhance security.



## Cross-Site Scripting (XSS)

The application is vulnerable to reflected XSS, allowing attackers to inject malicious scripts via GET and POST parameters. These scripts can execute in the victim's browser, compromising user data and trust.

### Risk:

- **Likelihood:** Medium – Easily exploitable with crafted URLs or payloads.
- **Impact:** Medium – Attackers can steal sensitive user data, session cookies, or deliver malicious content.

### Tools Used:

- Burp Suite
- OWASP ZAP
- Manual Payload Injection



Fig. 4.25: XSS



Fig. 4.26: XSS Message





Fig. 4.27: HTML Injection

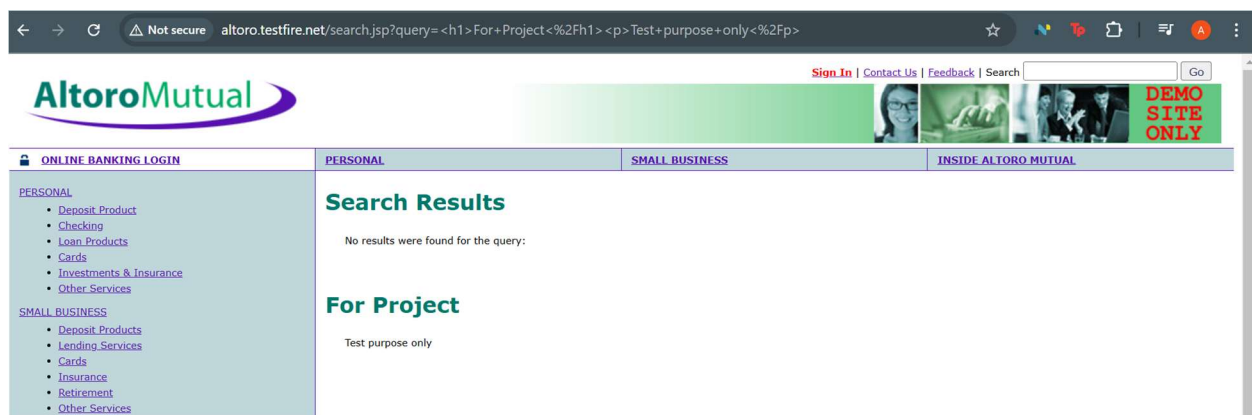
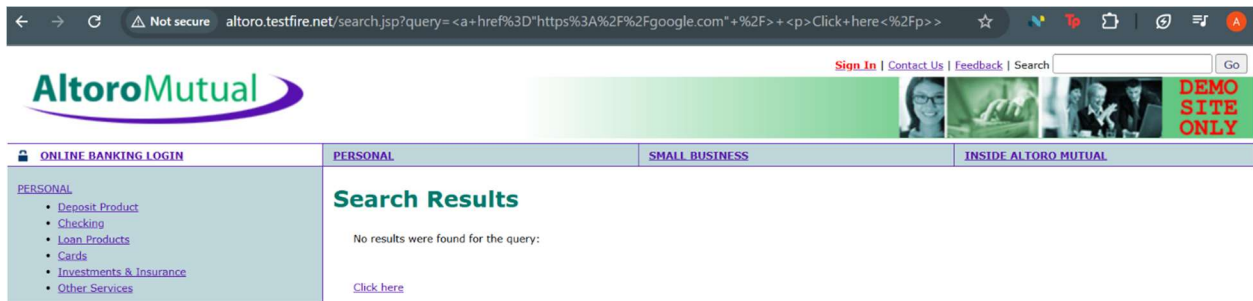


Fig. 4.28: Reflected HTML Get



Fig. 4.29: Reflected HTML Post



**Fig. 4.30: Reflected HTML Post**

### Recommendations:

1. **Input Validation:** Validate and sanitize all user inputs to remove potentially malicious characters or scripts.
2. **Output Encoding:** Apply proper encoding (e.g., HTML, JavaScript) to user-generated content before rendering it on the web page.
3. **Use Content Security Policy (CSP):** Implement CSP headers to restrict sources of executable scripts and mitigate XSS attacks.
4. **Escape User Input:** Use server-side frameworks that automatically escape user inputs in dynamic content.
5. **Implement HTTPOnly and Secure Cookies:** Prevent session cookie theft by setting HTTPOnly and Secure flags on cookies.
6. **Regular Security Testing:** Conduct periodic vulnerability assessments, including automated and manual testing for XSS.

## 4.2.4 Informative Vulnerabilities

### Information Leakage

The source code of `login.jsp` contains sensitive comments, such as administrator contact details. This provides attackers with reconnaissance information that can aid in further exploitation.

#### Risk:

- **Likelihood:** Medium – Readily accessible to attackers inspecting the application's source code.
- **Impact:** Informational – Helps attackers gather intelligence about the system.

#### Tools Used:

- Burp Suite
- Manual Inspection

```
<td valign="top" colspan="3" class="bb">
  <div class="f1" style="width: 99%;">

    <h1>Online Banking Login</h1>

    <!-- To get the latest admin login, please contact SiteOps at 415-555-6159 -->
    <p><span id="_ct10__ct10_Content_Main_message" style="color:#ff0066;font-size:12pt;font-weight:bold;">

  </span></p>

  <form action="doLogin" method="post" name="login" id="login" onsubmit="return (confirminput(login));">
    <table>
      <tr>
        <td>
          Username:
        </td>
        <td>
          <input type="text" id="uid" name="uid" value="" style="width: 150px;">
        </td>
      </tr>
      <tr>
        <td>
          Password:
        </td>
        <td>
          <input type="password" id="passw" name="passw" style="width: 150px;">
        </td>
      </tr>
    </table>
  </form>
</td>
```

Fig. 4.31: Information Leakage

**Recommendations:**

- **Remove Sensitive Comments:** Ensure all comments containing sensitive or unnecessary information are removed from production code.
- **Obfuscate Source Code:** Minify or obfuscate the source code to make information leakage more difficult.
- **Use Secure Development Practices:** Educate developers on secure coding practices to avoid adding sensitive information to the source.
- **Periodic Code Reviews:** Regularly audit source code for any unintentional information leaks.

**Internal Server Error Display**

The application displays detailed internal server error messages, exposing Apache Tomcat version details and other sensitive server information to attackers.

**Risk:**

- **Likelihood:** Medium – Exploitable during server errors or misconfigurations.
- **Impact:** Informational – Aids attackers in identifying potential exploits for the specific server version.

**Tools Used:**

- Manual Testing

**Recommendations:**

- **Custom Error Pages:** Configure the server to display generic error messages rather than exposing detailed server information.
- **Disable Verbose Error Reporting:** Modify server settings to disable verbose error output for unhandled exceptions.
- **Update Server Software:** Regularly update Apache Tomcat and other server components to the latest secure versions.
- **Monitor Logs:** Regularly monitor logs to identify and fix error-prone areas in the application.

← → ↺ ⚠ Not secure testfire.net/bank/showAccount?listAccounts=800002%20union%20select%201,2,3,4,5,6,7#&action=go

---

## HTTP Status 500 – Internal Server Error

---

**Type** Exception Report

**Message** java.lang.NumberFormatException: For input string: "800002 union select 1,2,3,4,5,6,7"

**Description** The server encountered an unexpected condition that prevented it from fulfilling the request.

**Exception**

```
org.apache.jasper.JasperException: java.lang.NumberFormatException: For input string: "800002 union select 1,2,3,4,5,6,7"
  org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:594)
  org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:510)
  org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:395)
  org.apache.jasper.servlet.JspServlet.service(JspServlet.java:339)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
  org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
  com.ibm.security.appscan.altonomutual.filter.AuthFilter.doFilter(AuthFilter.java:67)
  com.ibm.security.appscan.altonomutual.servlet.AccountViewServlet.doGet(AccountViewServlet.java:58)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:624)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
  org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
  com.ibm.security.appscan.altonomutual.filter.AuthFilter.doFilter(AuthFilter.java:67)
```

**Root Cause**

```
java.lang.NumberFormatException: For input string: "800002 union select 1,2,3,4,5,6,7"
  java.lang.NumberFormatException.forInputString(Unknown Source)
  java.lang.Long.parseLong(Unknown Source)
  java.lang.Long.parseLong(Unknown Source)
  com.ibm.security.appscan.altonomutual.model.Account.getAccount(Account.java:41)
  org.apache.jsp.bank.balance_jsp._jspService(balance_jsp.java:170)
  org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
  org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:472)
  org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:395)
  org.apache.jasper.servlet.JspServlet.service(JspServlet.java:339)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
  org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
  com.ibm.security.appscan.altonomutual.filter.AuthFilter.doFilter(AuthFilter.java:67)
  com.ibm.security.appscan.altonomutual.servlet.AccountViewServlet.doGet(AccountViewServlet.java:58)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:624)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
  org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
  com.ibm.security.appscan.altonomutual.filter.AuthFilter.doFilter(AuthFilter.java:67)
```

**Note** The full stack trace of the root cause is available in the server logs.

---

Apache Tomcat/7.0.92

Fig. 4.31: Internal server error display

## CONCLUSION

The Vulnerability Assessment and Penetration Testing (VAPT) of the Altoro testfire.net application revealed critical, high, medium, and informational vulnerabilities using tools like Burp Suite, Nuclei, Nikto, and manual analysis. Critical issues such as SQL Injection and IDOR, along with high-risk flaws like Default Credentials and Brute Force vulnerabilities, pose significant threats to the system's confidentiality, integrity, and availability. Medium and informational vulnerabilities highlight areas for improvement in securing sensitive information. Recommendations provided include secure coding practices, strong authentication mechanisms, input validation, and adherence to security standards. Addressing these vulnerabilities will enhance the application's security posture and protect against evolving cyber threats.

## REFERENCES

1. OWASP (Open web Application Security Project)  
<https://owasp.org/www-project-top-ten/>
2. Secure Coding Practice Guide  
<https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/>
3. PortSwigger  
<https://portswigger.net/burp/documentation>
4. Nmap Reference Guide  
<https://nmap.org/book/>
5. Nikto Web Scanner  
<https://cirt.net/Nikto2>
6. CVSS (Common Vulnerability Scoring System)  
<https://www.first.org/cvss/>
7. TLS Security Configuration  
[https://wiki.mozilla.org/Security/Server\\_Side\\_TLS](https://wiki.mozilla.org/Security/Server_Side_TLS)