**Penetration Testing Report**

**Organization Name:** RedactedClient

**Date of Assessment:** 27-02-2025

**Testing Team:** Zia-ur-Rehman

**Confidentiality Statement:**

This document contains confidential and sensitive information related to the security posture of **RedactedClient**. It is intended solely for authorized recipients within **RedactedClient** and must not be shared, copied, or distributed without explicit permission from **RedactedClient** or the security assessment team.

Unauthorized disclosure of the information in this report may expose **RedactedClient** to security risks, legal consequences, or compliance violations. Any misuse, alteration, or unauthorized access to this document is strictly prohibited.

If you are not the intended recipient, please notify the sender immediately and delete this document from your records.

**RedactedClient** reserves the right to take legal action against any unauthorized use or distribution of this report.

**Executive Summary:**

#### **Overview of the Penetration Test**

This penetration test was conducted to assess the security posture of RedactedClient and identify potential vulnerabilities that could be exploited by malicious actors. The assessment focused on evaluating the effectiveness of existing security controls, identifying weaknesses, and providing actionable recommendations to mitigate risks.

#### **Objectives and Scope of the Test**

The primary objectives of this penetration test were:

* To identify security vulnerabilities in the organization's network, web applications, and critical systems.
* To assess the potential impact of identified vulnerabilities on the confidentiality, integrity, and availability of sensitive data.
* To evaluate the effectiveness of current security controls and defensive measures.
* To provide detailed remediation steps for discovered security issues.

**Scope of Testing:**

* In-Scope Systems: [List specific IP addresses, domains, applications, or network components tested]
* Testing Approach: Black-box, gray-box, or white-box testing
* Testing Methodologies Used: Automated scanning, manual exploitation, social engineering, API security testing, etc.

### **Engagement Details**

#### **Scope of Testing**

The penetration test focused on assessing the security of RedactedClient' web application to identify vulnerabilities that could be exploited by external attackers. The test targeted:

* Web Application URL: https://www.example.com(confidential)/
* Key Features Tested: User authentication, input validation, session management, API endpoints, access controls

#### **Testing Methodology**

A black-box testing methodology was used, meaning the test was conducted without prior knowledge of the application's internal workings, source code, or credentials. The focus was on identifying vulnerabilities an external attacker could exploit.

#### Testing Approach

The assessment involved:

* **Reconnaissance & Information Gathering:** Identifying publicly available data and analyzing web application structure.
* **Automated Vulnerability Scanning:** Using security tools to detect known vulnerabilities.
* **Manual Exploitation:** Validating findings through controlled exploitation.
* **Web Application Security Testing:** Checking for OWASP Top 10 vulnerabilities, including:
  + SQL Injection
  + Cross-Site Scripting (XSS)
  + Broken Authentication & Session Management
  + Insecure Direct Object References (IDOR)
  + Security Misconfigurations
  + API security flaws

**Tools Used**

The following tools were utilized for testing:

* **Burp Suite** – Intercepting and modifying web traffic.
* **Nikto** – Web server vulnerability scanning.
* **OWASP ZAP** – Automated and manual security testing.
* **SQLmap** – Detecting and exploiting SQL injection vulnerabilities.
* **Nmap** – Network enumeration (if applicable).

**Nmap Scanning:**

Potential Vulnerabilities in the Nmap Scan of example.com(confidential)

Your Nmap scan reveals several potential security concerns that could be exploited by attackers. Below are the key findings:

1. **HTTP Port 80 - Misconfigured Redirect**

Finding: The HTTP service on port 80 returns a "301 Moved Permanently" response but redirects to an invalid location ("https:///").

Risk: This misconfiguration can cause issues for legitimate users, and attackers could use it for phishing or open redirect attacks.

Fix: Ensure the redirection points to a valid HTTPS URL.

2**. Exposed HTTP Headers - Potential Information Disclosure**

Finding: The response includes various security-related headers, such as:

server: RedactedClient

x-frame-options: SAMEORIGIN

x-content-type-options: nosniff

x-xss-protection: 1;mode=block

content-security-policy

Risk: Although some security headers are present, exposing the server name (server: RedactedClient) may allow attackers to fingerprint the application and search for known vulnerabilities.

Fix: Hide or modify the Server header to reduce fingerprinting risks.

3. **Weak SSL/TLS Configuration**

Finding: The SSL certificate details:

Valid from: May 4, 2024

Expires on: June 5, 2025

ssl-date: TLS randomness does not represent time

Risk: The Nmap scan reports an issue with the SSL/TLS randomness, which may indicate weak entropy in the SSL implementation. This could lead to MITM (Man-in-the-Middle) attacks or SSL vulnerabilities (e.g., BEAST, POODLE).

Fix:

Ensure TLS 1.2 or TLS 1.3 is enforced.

Use strong cipher suites and disable weak encryption protocols.

4. **HTTP 403 Forbidden on HTTPS (Port 443)**

Finding: The HTTPS response to a normal GET request returns 403 Forbidden.

Risk: This could indicate a default deny policy, but if misconfigured, it might allow attackers to infer restricted resources or conduct brute force attacks.

Fix: Review access control policies and ensure only authorized users can access restricted content.

5. **Content Security Policy (CSP) Issues**

Finding: The CSP allows:

script-src \* 'unsafe-inline' 'unsafe-eval' blob: \*.example.com(confidential);

style-src \* 'unsafe-inline' \*.example.com(confidential);

Risk:

unsafe-inline and unsafe-eval weaken security and can allow XSS (Cross-Site Scripting) attacks.

The wildcard (\*) in CSP allows external scripts, increasing the risk of supply chain attacks.

Fix:

Remove unsafe-inline and unsafe-eval.

Restrict script sources to trusted domains only.

6. **Internal IP Exposure in CSP (Potential Misconfiguration)**

Finding: The CSP includes:

perl

CopyEdit

frame-ancestors account.example.com(confidential) http://127.0.0.1:5500 http://127.0.0.1:5501;

Risk: This suggests that localhost (127.0.0.1) is allowed to frame the website. This might allow clickjacking attacks or unauthorized access in certain environments.

Fix:

Remove localhost (127.0.0.1) from frame-ancestors unless absolutely required.

Ensure clickjacking protections are in place.

7. **Possible OS Fingerprinting & Enumeration**

Finding: The OS detection suggests the server might be running Linux (3.2, 4.4) or Windows XP/7/Server 2012.

Risk: If outdated OS versions (e.g., Windows XP, Windows Server 2012) are used, they could have unpatched vulnerabilities.

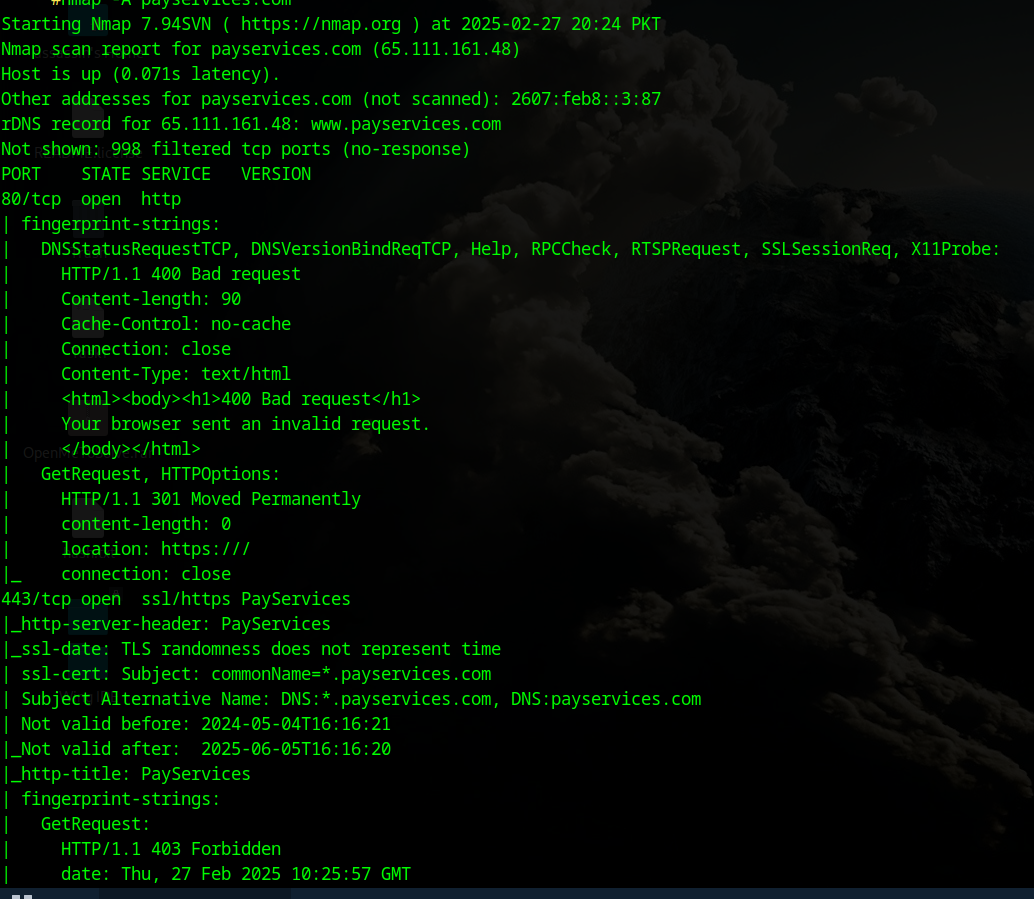
Fix: Ensure the system is fully patched and running an up-to-date operating system.

8. **High Network Latency (Potential DoS Risk)**

Finding: The scan shows a 71ms latency.

Risk: A high latency response could indicate susceptibility to DoS (Denial-of-Service) attacks.

Fix: Use rate limiting, WAF (Web Application Firewall), and load balancing to mitigate DoS risks.



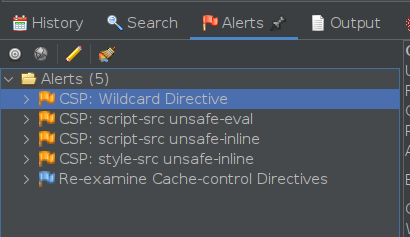
**Owasp Zap:**

**1. Content Security Policy (CSP) Issues**

* Wildcard Directive (\*) – Allows content from any source, increasing risk of XSS and data leaks.
* unsafe-eval in script-src – Permits JavaScript eval(), leading to code injection vulnerabilities.
* unsafe-inline in script-src & style-src – Enables XSS and CSS injection attacks.

**2. Weak Cache-Control Directives**

Lack of strict cache settings may expose sensitive user data in cached responses.



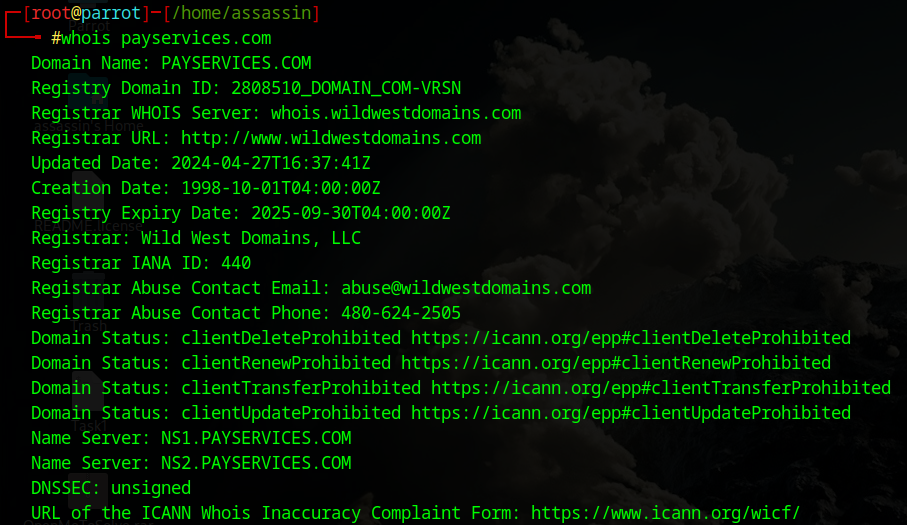
**WHOIS:**Security Vulnerabilities Identified in example.com(confidential)

**Domain Privacy Protection:**

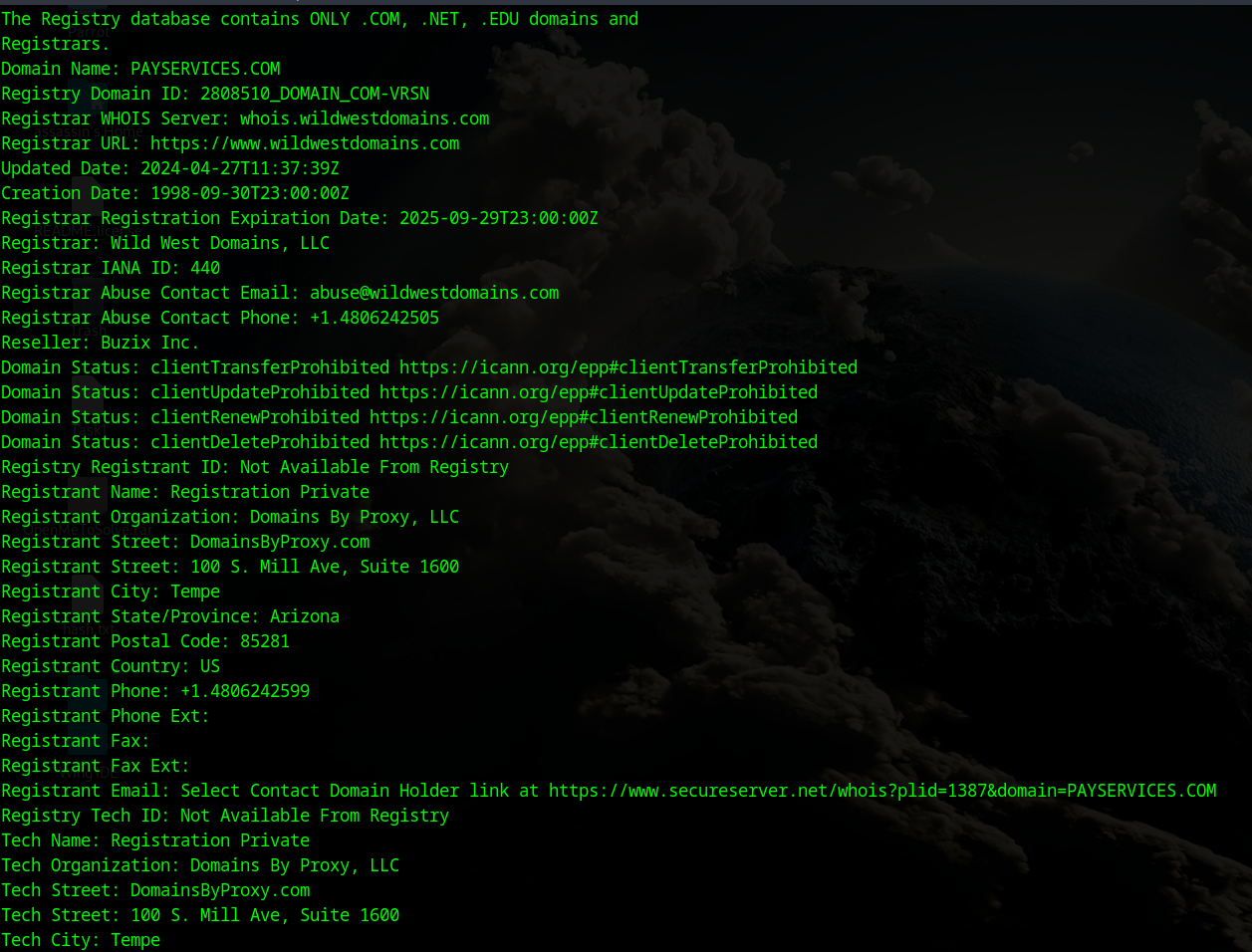
The domain uses Domains By Proxy, LLC for privacy protection. While this hides the registrant’s identity, it can also make it difficult to verify legitimate ownership and may be exploited for malicious purposes.

**DNSSEC Not Enableb:**

The domain’s DNSSEC status is "unsigned," meaning that it lacks cryptographic protection against DNS spoofing or cache poisoning attacks.







**DIG :**

### **SPF Record (v=spf1 mx a:mail.example.com(confidential) -all)**

* Issue: The SPF record is restrictive (-all), which is good, but it only allows emails from mail.example.com(confidential). If mail.example.com(confidential) is misconfigured or compromised, it could still be abused.
* Recommendation: Ensure DKIM and DMARC are properly configured to strengthen email security.

### 2. **Missing DMARC and DKIM Records**

* Issue: The ANY query does not show any DMARC (\_dmarc.example.com(confidential)) or DKIM (default.\_domainkey.example.com(confidential)) records. Without DMARC and DKIM, email spoofing is easier.
* Recommendation: Add a DMARC record (e.g., v=DMARC1; p=reject; rua=mailto:admin@example.com(confidential)) and configure DKIM to prevent email spoofing.

### 3. **CAA Records (issue "digicert.com", sectigo.com, globalsign.com)**

* Issue: These records restrict SSL certificate issuance to DigiCert, Sectigo, and GlobalSign, which is good practice. However, if wildcard certificates are allowed, they could be misused.
* Recommendation: Verify whether wildcard certificates (\*.example.com(confidential)) are restricted if not needed.

### 4. **SOA Record (2011071001, ns1.example.com(confidential))**

* Issue: The serial number (2011071001) suggests it may not have been updated since 2011, which could indicate poor DNS management.
* Recommendation: Ensure the SOA record is updated when changes are made to DNS settings.

### 5. **Mail Server (MX mail.example.com(confidential))**

* Issue: No PTR record is shown in the response, which may cause email deliverability issues.
* Recommendation: Check if mail.example.com(confidential) has a valid PTR record (reverse DNS entry).

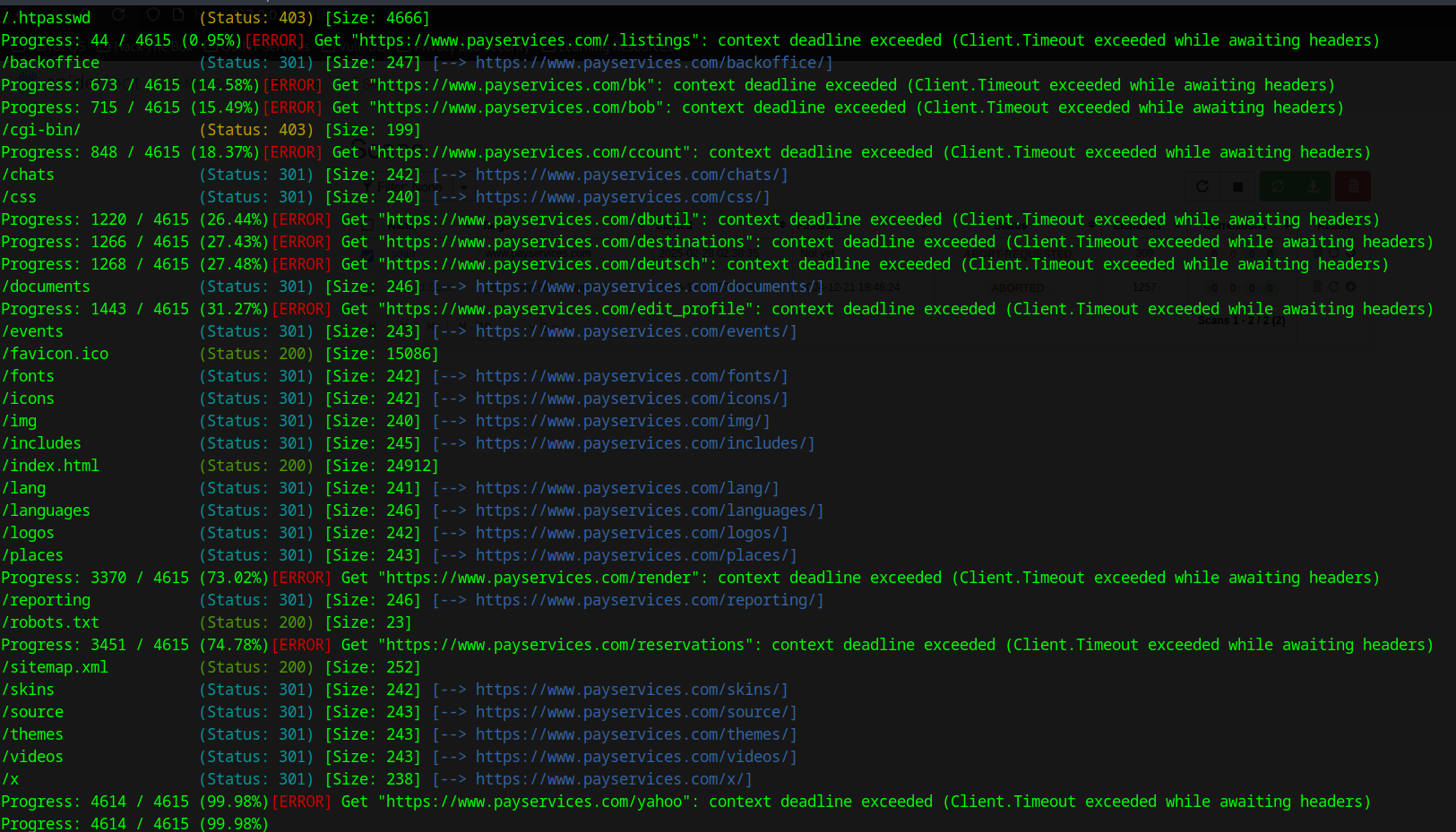
### 6. **AAAA Record (2607:feb8::3:87)**

* Issue: The IPv6 address is public. Ensure it is properly secured against unauthorized access.

### 7. **Name Servers (ns1.example.com(confidential), ns2.example.com(confidential))**

* Issue: If both name servers belong to the same network, redundancy issues could arise.
* **Recommendation**: Consider adding a third-party backup name server.

**GoBuster:**



**Restricted Directories (403 Forbidden):**

/cgi-bin/

/.hta, /.htaccess, /.htpasswd (Sensitive configuration files)

**Exposed Directories (301 Redirects):**

/backoffice/ (Potential admin panel)

/documents/, /reporting/ (Possible sensitive data)

**Publicly Accessible Files:**

/robots.txt (May contain restricted paths)

/sitemap.xml (Lists indexed pages)

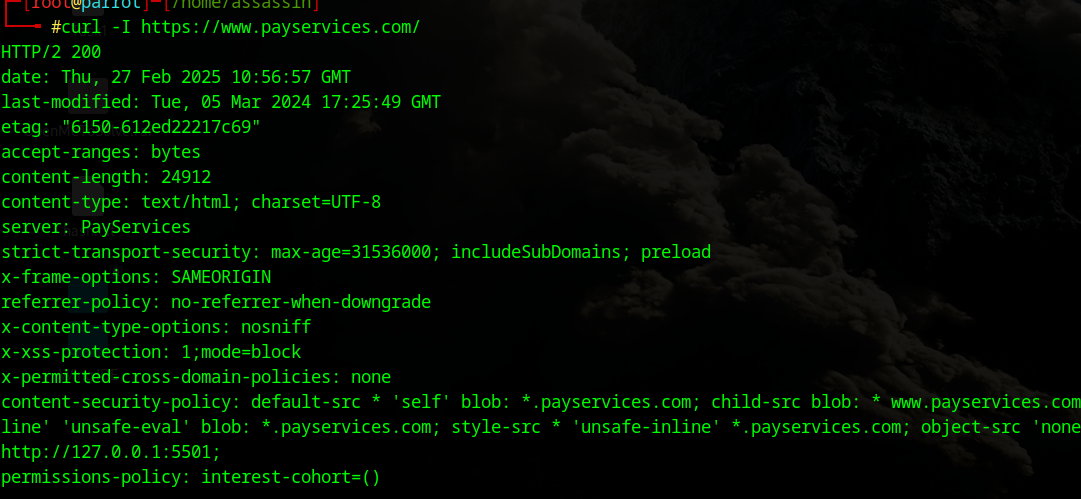
/favicon.ico (Indicates tech stack)

**Timeout Errors:**

Multiple URLs resulted in context deadline exceeded, possibly indicating rate-limiting or WAF protection.

These findings indicate potential security risks, such as exposed admin panels and restricted directories that could be targeted for enumeration.

**CURL:**



### **Content Security Policy (CSP) Issues**

**default-src \* 'self' blob: \*.example.com(confidential);**

* + Risk: The wildcard (\*) allows content from any source, which increases the risk of cross-site scripting (XSS) and data injection attacks.
  + Fix: Restrict sources by specifying trusted domains (e.g., default-src 'self' https://trusted.com;).

**script-src \* 'unsafe-inline' 'unsafe-eval' blob: \*.example.com(confidential);**

* + Risk:
    - 'unsafe-inline' allows inline JavaScript execution, making the site vulnerable to XSS attacks.
    - 'unsafe-eval' allows the execution of JavaScript via eval(), which attackers can exploit to inject malicious scripts.
  + Fix: Remove 'unsafe-inline' and 'unsafe-eval' and use nonce-based CSP (script-src 'self' 'nonce-randomString').

**style-src \* 'unsafe-inline' \*.example.com(confidential);**

* + Risk:
    - 'unsafe-inline' allows CSS injection, leading to defacement attacks and data exfiltration via CSS-based keylogging.
  + Fix: Remove 'unsafe-inline' and use nonce-based CSP (style-src 'self' 'nonce-randomString').

object-src 'none'

* + Good Practice: Blocks Flash and other plugin-based content, reducing attack surface.

### 2. **X-Frame-Options: SAMEORIGIN**

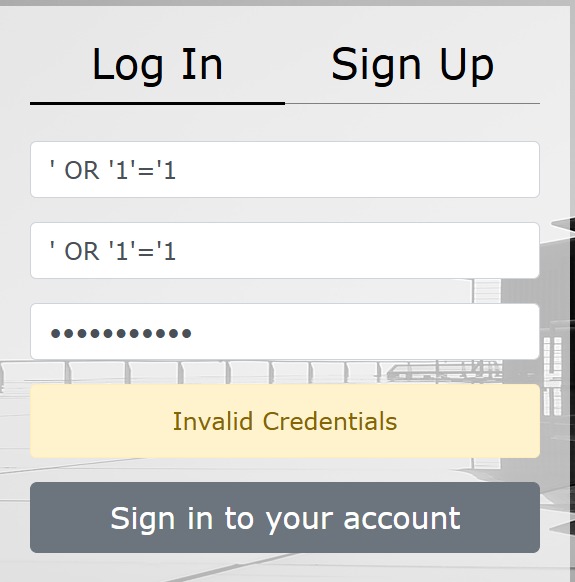
* Risk: Prevents clickjacking attacks within the same origin but still allows embedding on subdomains.
* Fix: Use DENY to prevent all framing or ALLOW-FROM https://trusted.com to whitelist specific domains.

### 3. **Referrer-Policy: no-referrer-when-downgrade**

* Risk: Allows referrer information to be sent when navigating from HTTPS to HTTP, which may leak sensitive URLs.
* Fix: Use strict-origin-when-cross-origin or no-referrer to prevent data leakage.

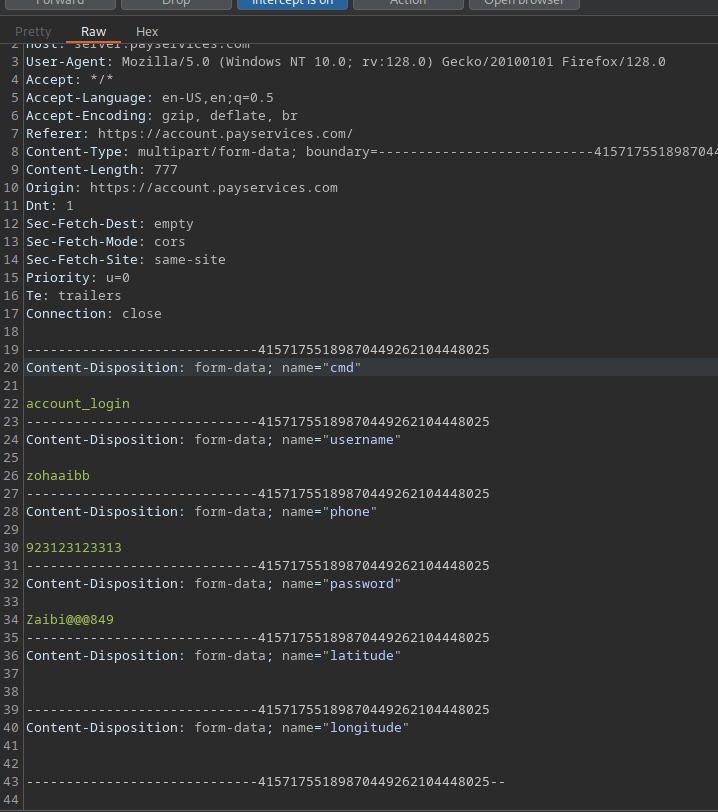
**Attacks:**

**Sql Injection:**



**This website is safe from sql injection as it has added input validation and It hasn’t allowed any kind of ‘ such characters in login page, but on login page for administrators we are able to enter sql commands which shouldn’t be allowed.**

**Broken Authentication:**



Whenever a user tries to login the website sends the credentials of user without any kind of encryption which could lead to theft of account of a user, which may lead to unauthorized access causing leakage of sensitive information as well.

**DOM XSS:**

**DOM-Based Cross-Site Scripting (XSS) Assessment**

**Test Objective:**

The objective of this assessment was to determine whether the target web application is vulnerable to DOM-Based Cross-Site Scripting (XSS) by injecting various payloads and analyzing client-side JavaScript execution.

**Methodology:**

**Input Field Analysis:**

Several input fields, including login forms, search boxes, and comment sections, were tested for XSS injection.

Various payloads, such as <script>alert(1)</script>, <img src=x onerror=alert(1)>, and event-based injections (onmouseover=alert(1)) were used.

**Burp Suite Testing:**

Requests containing XSS payloads were intercepted and modified using Burp Suite.

The application’s response was analyzed to check for script execution.

**DOM Manipulation:**

The browser's developer console was used to test JavaScript execution via document.write(), innerHTML, and other DOM modification methods.

**Findings:**

The application effectively sanitizes input fields, preventing the injection of malicious scripts.

No user-supplied input was reflected in a way that could allow script execution in the browser.

Attempts to manipulate the DOM via developer console (document.write(), innerHTML) did not result in JavaScript execution, indicating proper security controls.

**Mitigations for Identified Vulnerabilities**

### **1. Misconfigured Redirect (HTTP Port 80)**

**Mitigation:**

Ensure the HTTP redirection points to a valid HTTPS URL.

Implement HTTP Strict Transport Security (HSTS) to enforce HTTPS connections.

### **2. Exposed HTTP Headers (Information Disclosure)**

**Mitigation:**

Remove or obfuscate the "Server" header to prevent fingerprinting.

Implement security headers such as Content Security Policy (CSP) and Referrer-Policy.

### **3. Weak SSL/TLS Configuration**

**Mitigation:**

Enforce TLS 1.2 or TLS 1.3.

Use strong cipher suites and disable weak encryption protocols.

Regularly update SSL certificates and ensure proper entropy in SSL implementations.

### 4. **HTTP 403 Forbidden on HTTPS (Port 443)**

**Mitigation:**

Ensure proper access control policies are in place.

Implement Web Application Firewall (WAF) to prevent unauthorized access attempts.

### 5. **Content Security Policy (CSP) Issues**

**Mitigation:**

Remove unsafe-inline and unsafe-eval directives.

Restrict script sources to trusted domains.

Implement a strict CSP policy to prevent XSS attacks.

### **6. Internal IP Exposure in CSP**

**Mitigation:**

Remove localhost (127.0.0.1) from frame-ancestors.

Use proper access control to prevent unauthorized iframe embedding.

### 7. **OS Fingerprinting & Enumeration**

**Mitigation:**

Upgrade to the latest supported OS version and apply security patches.

Implement network segmentation and firewall rules to limit OS exposure.

### 8. **High Network Latency (Potential DoS Risk)**

**Mitigation:**

Implement rate limiting and anti-DoS mechanisms.

Use a Web Application Firewall (WAF) and load balancing techniques.

### 9. **Weak Cache-Control Directives**

**Mitigation:**

Set Cache-Control: no-store, no-cache, must-revalidate for sensitive content.

### 10. **WHOIS and DNSSEC Issues**

**Mitigation:**

Enable DNSSEC to protect against DNS spoofing attacks.

Regularly verify WHOIS information to prevent misuse.

### 11. **Email Security Issues (SPF, DKIM, DMARC)**

**Mitigation:**

Configure SPF, DKIM, and DMARC to prevent email spoofing and phishing attacks.

Use a strict DMARC policy (e.g., p=reject).

### 12. **Exposed Admin Panels & Sensitive Files**

**Mitigation:**

Restrict access to /backoffice/ and other sensitive directories.

Implement multi-factor authentication (MFA) for admin login.

### 13. **SQL Injection on Admin Login Page**

**Mitigation:**

Implement prepared statements and parameterized queries.

Use input validation and web application firewall (WAF) rules.

### 14. **Broken Authentication (Credentials Sent in Plaintext)**

**Mitigation:**

Implement HTTPS for all login forms.

Use secure password storage with hashing and salting.

15**:PlainText Password Exposure in Requests:**

Implement secure password hasing (bcrypt, Argon2), Use HTTPS, and never expose passwords in logs or requests.