PRAGYA

Defending against digital threats

Sample Web Application VAPT Report

Defending Against Digital Threats

Table of Contents

Document Name: Web application PEN Test Report – Organization Classification:

Confidential Reference: Web application PEN Test Report Version: v1.0

Date:10-03-2025

Document Confidentiality

This document contains sensitive information which needs to be shared with appropriate personnel of the Organization on a purely need to know basis. The following recommendations are issued with respect to distributing this document

- Distribute to authorized personnel only
- Comply with the appropriate security measures according to the classification level of the document

Version History

Version	Author	Comment	Date
1.0	Ananth Nandyala	Web App VAPT	10 March 2025

1. Executive Summary

1.1. Project Objective

Our primary goal within this project was to provide the Organization with an understanding of the current level of security in the web application and its infrastructure components. We completed the following objectives to accomplish this goal:

- Identifying application-based threats to and vulnerabilities in the application
- Comparing Organization current security measures with industry best practices
- Providing recommendations that Organization can implement to mitigate threats and vulnerabilities and meet industry best practices

1.2. Scope & Timeframe

The section defines the scope and boundaries of the project.

I. Constraints and Limitations

The assessment was performed with the knowledge shared by the Company Onboarding team about the target. Pragya Cyber conducted the assessments, and the result(s) / finding(s) made are highly subjective to target system(s) and service(s) visibility and availability at that given point of time.

II. Target Scope

Identify weaknesses that might be exploited by adversaries who have authorized or unauthorized access to Company Technical Skill Test and underlying infrastructure:

- Test access credentials were provided. It was a Grey-Box Testing.
- The objective is to mimic an adversary and identify the threats and vulnerabilities.

The following application was in the scope of the penetration test. Automated as well as manual security testing was conducted

Sr.no	Application Type	URL/IP/Domain
1	Web application (Organization)	https://xxxxxxxxx.com/

1.3. User Accounts provided by Organization

URL/IP/API	Username
https://xxxxxxxxx.com/	• **Username:** xxxxxxxx **Password:** xxxxxxxx

1.4. Vulnerability Summary

1	0	5	3	9
Critical	High	Medium	Low	Total

1.5. Summary of Business Risks

Vulnerability	Business Risk	Criticality
Session Hijacking via Insecure Session Management	Attackers can hijack user sessions to gain unauthorized access to company's systems, leading to data breaches, financial fraud, or disruption of digital services. This can result in regulatory penalties, reputational damage, and loss of customer trust.	CRITICAL
Missing Security Headers	Lack of security headers increases exposure to attacks like XSS and clickjacking, compromising patient data security.	MEDIUM
Browser Cache Weakness	Storing sensitive information in cache may lead to data leaks if an attacker gains access to a shared or compromised device.	MEDIUM
Improper Error Handling & SQL Validation Issues	Unhandled errors and weak SQL validation may expose database structures, leading to data breaches or injection attacks.	MEDIUM
Clickjacking	Attackers can trick users into performing unauthorized actions, potentially leading to data manipulation or financial fraud.	MEDIUM
Cross-Site Request Forgery (CSRF) Token Manipulation	Exposure of server details increases the risk of targeted attacks by providing attackers with insights into system configurations and weaknesses.	MEDIUM

CORS Misconfiguration	Misconfigured CORS can allow unauthorized access to internal APIs, exposing sensitive business logic and patient data	**LOW**
TLS Cookie Without Secure Flag Set	Cookies transmitted over HTTP can be intercepted, leading to session hijacking and unauthorized system access.	**LOW**
Expired Token Still Valid	Reuse of expired tokens can allow unauthorized access, bypassing authentication and leading to data integrity risks.	**LOW**

1.6. Standards Followed

- OWASP
- OSSTMM
- PTES
- WASC-TC

2. Technical Details

2.1 Methodology

- Penetration Testing Execution Standard (PTES)
- OWASP Top 10 Application Security Risks
- OWASP Web Security Testing Guide
- Open-Source Security Testing Methodology Manual (OSSTMM)
- Web Application Security Consortium Threat Classification (WASC-TC)

2.2 Terminology and score

CVE - is a dictionary of publicly known information security vulnerabilities and exposures. CVE's common identifiers enable data exchange between security products and provide a baseline index point for evaluating coverage of tools and services. An information security "vulnerability" is a mistake in software application, configuration or operating system that can be directly used by a hacker to gain access to a system or network.

Vulnerability - A weakness which allows a hacker to break into / compromise a system's security.

Exploit - Code which allows an attacker to take advantage of a vulnerable system.

Payload - Actual code which runs on the system after exploitation.

CWE - Common Weakness Enumeration is a tangible set of software weaknesses that

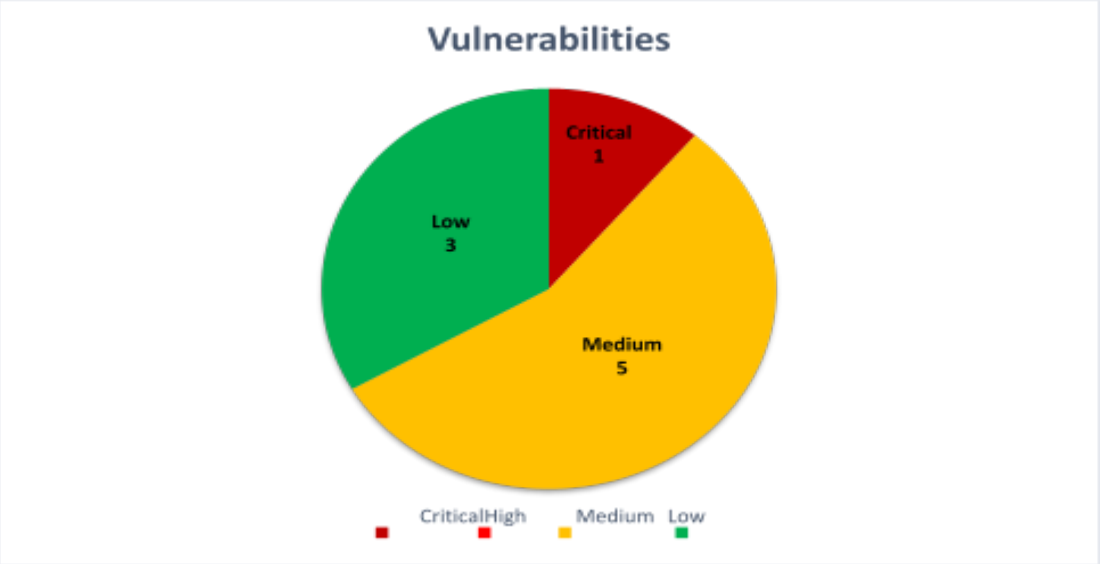
Priority Level	Severity Scale	CVSS Score	Description of Vulnerability
P1	Critical	9.0-10.0	Vulnerabilities that affect all users of the platform, and /or affect the security of the platform or host system
P2	High	7.0-8.9	Vulnerabilities that affect more than one user of the platform, and that require little or no user interaction to trigger.
P3	Medium	4.0-6.9	Vulnerabilities that affect more than one user but may also require interaction or a specific configuration
P4	Low	0.1-3.9	Issues that affect singular users and require interaction or significant prerequisites (MITM) to trigger
P5	Informational	0.0	Issues that leaking very basic information which might lead to information disclosure

2.3 Security tools used

- Manual testing: Burp Suite Pro
- Vulnerability scan: Nessus, Wapiti, nikto, ZAP, commix
- Directory enumeration: gobuster, dirsearch
- Injection testing tools: DOM Invader, SQLmap
- Encryption: sslscan

3. Finding Details

3.1 Summary of Findings



3.2 Critical Severity Findings

3.1.1 Vulnerability: Session Hijacking via Insecure Session Management

Description

Session hijacking was identified during penetration testing, where an attacker could manipulate session- related values such as loggedinusertype and userinfo to escalate privileges. The application does not enforce strict session validation, allowing a lower-privileged admin to modify session attributes and impersonate a higher-privileged admin.

Specifically, by altering the values of **cookies, loggedinusertype, and userinfo**, the session was successfully elevated without requiring authentication or revalidation. This indicates a flaw in the session handling mechanism, where the server trusts client-side session parameters instead of verifying them securely on the server.

Impact

- **Privilege Escalation:** Attackers can gain unauthorized admin access.
- **Data Exposure:** Sensitive information accessible only to higher-privileged users can be compromised.
- **Account Impersonation:** Attackers can act as legitimate users or admins.
- **Compliance Risks:** Violates security best practices (OWASP A07:2021 - Identification and Authentication Failures).

CVSS Score

- 9.1

Severity

- **Critical**

Mitigation

Enforce Server-Side Session Validation:

- Never trust or rely on client-side session attributes (e.g., loggedinusertype, userinfo).
- Maintain session state and privilege levels securely on the server.

Use Strong Session Tokens:

- Generate and validate **cryptographically secure session tokens** on every request.
- Use **JWT (JSON Web Tokens) with proper signing and expiration** or **server-stored session IDs**.

Regenerate Session IDs on Privilege Changes:

- When a user's privilege level is changed, **invalidate old sessions** and generate a new session ID.

Restrict Direct Access to Sensitive Variables:

- Ensure loggedinusertype and userinfo are never stored or modifiable on the client side.
- Validate user roles and permissions on the **server-side** for every request.

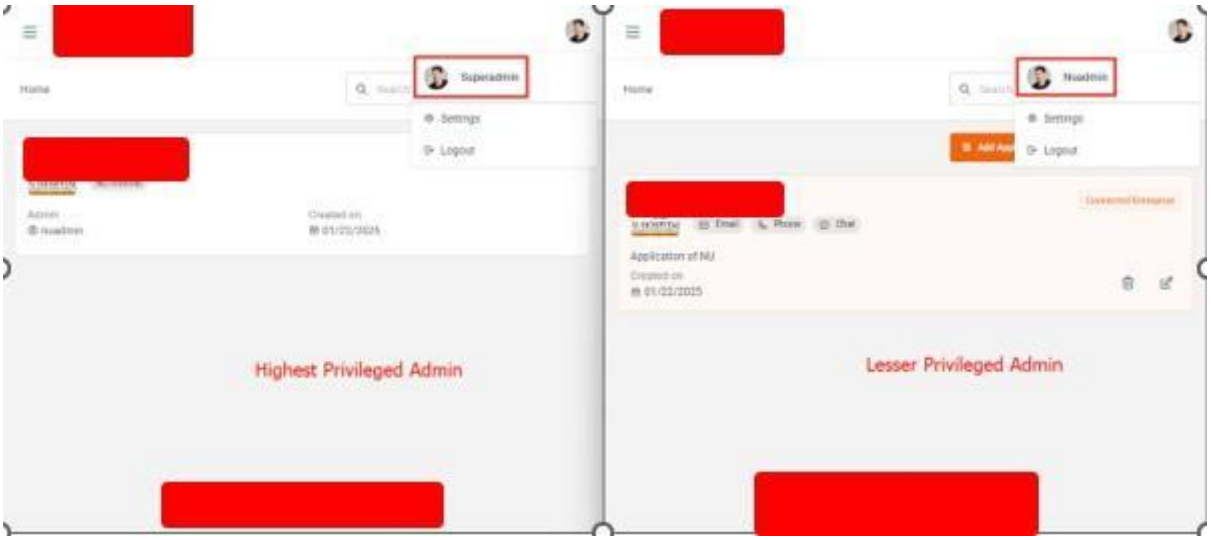
Implement Role-Based Access Controls (RBAC):

- Enforce **strict role validation** on sensitive endpoints.
- Implement **least privilege access** to minimize exposure.

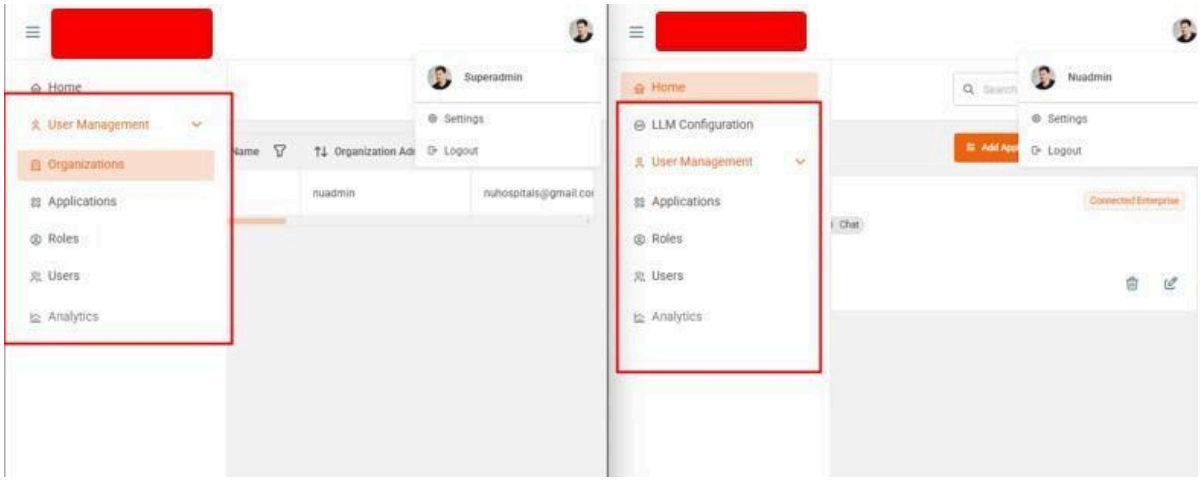
Session Monitoring & Anomaly Detection:

- Log session activities, including privilege escalations and user role changes.
- Use **multi-factor authentication (MFA)** for administrative access.

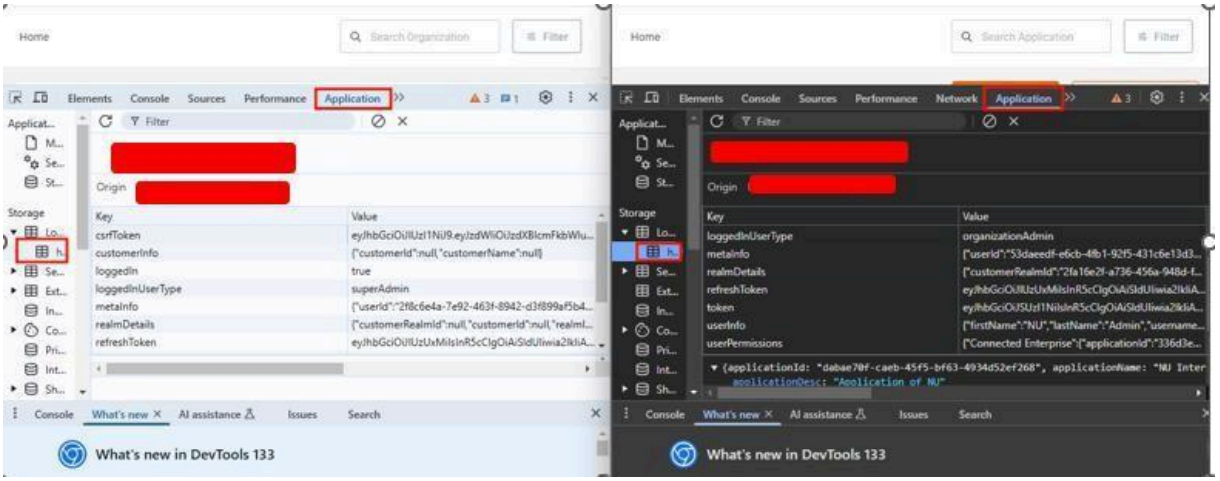
Evidence



Highest admin on left, lesser privileged admin on right

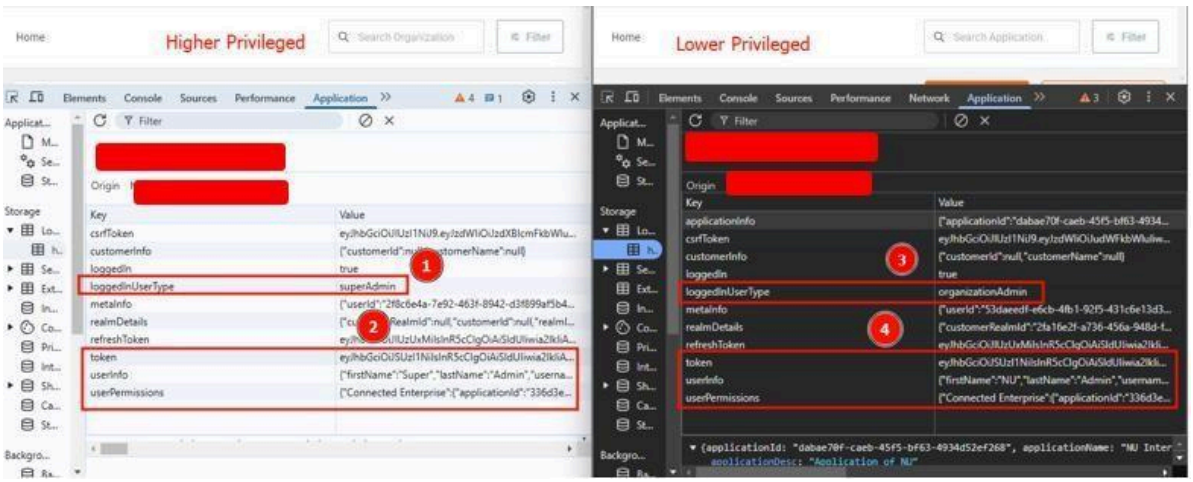


- In the above fig, we can see that the highest privileged admin has the access to add organisations and access it while the lesser privileged admin does not have organisation access.



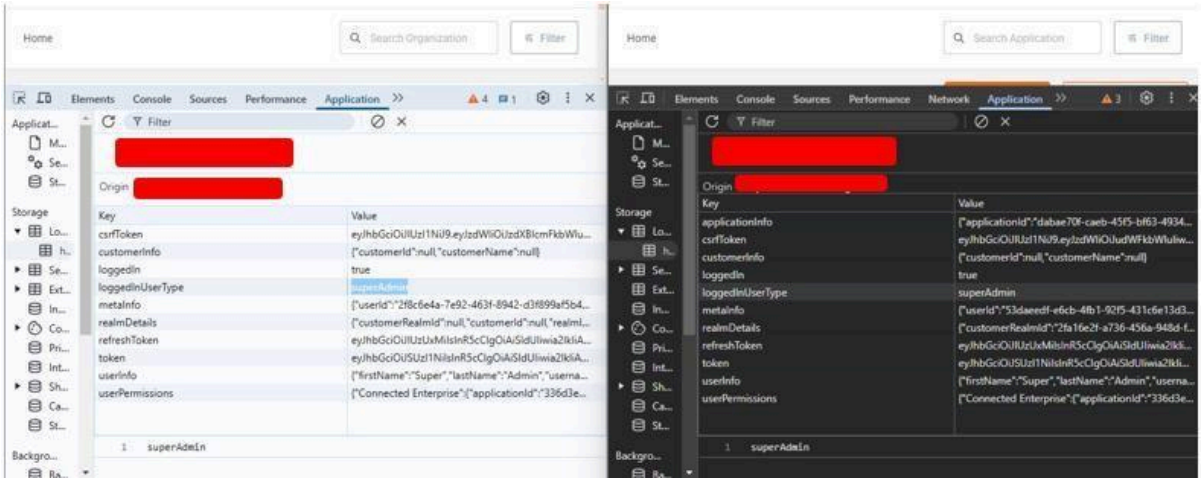
Local session storage

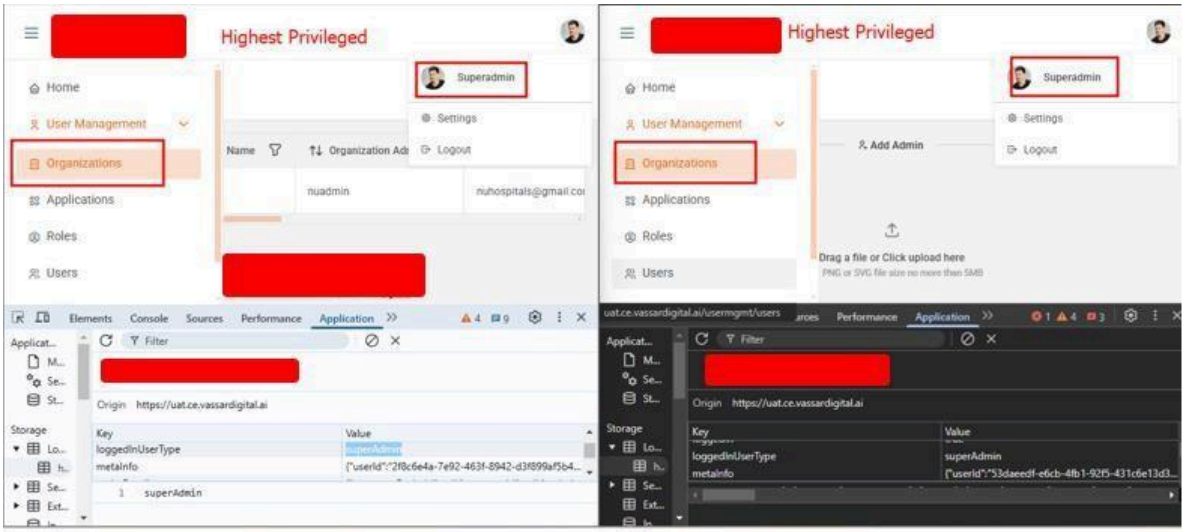
- Navigate to inspect element -> Application -> Local storage.



Replace the content in the box

- Replace the contents in the box such as loggedinUserType, token, userinfo and UserPermissions from higher privileged to lesser privileged admin.





Gained session of highest privileged admin

- We can see that now on the right-hand side, we have access to the organisation and can modify the organisation data.
- Hence, Session Hijacking is present in the application.

3.3 High Severity Findings

NONE

3.4 Medium Severity Findings

3.4.1 Vulnerability: Missing Security Headers

Description

Security headers are essential HTTP response headers that enhance web application security by mitigating various threats, including **Cross-Site Scripting (XSS), Clickjacking, MIME type sniffing, and insecure transport mechanisms**. The absence or misconfiguration of these headers can expose applications to **client- side attacks, unauthorized content embedding, and data leakage**.

Key security headers include:

- **Content-Security-Policy (CSP):** Restricts resource loading to prevent XSS and data injection attacks.
- **X-Frame-Options:** Mitigates Clickjacking by controlling frame embedding.
- **Strict-Transport-Security (HSTS):** Enforces secure HTTPS communication to prevent SSL stripping attacks.
- **X-Content-Type-Options:** Prevents MIME-type sniffing to block content-type spoofing attacks.
- **Referrer-Policy:** Regulates the amount of referrer data sent with requests to prevent information disclosure.

Impact

- **Increased Attack Surface**: Without proper security headers, web applications are more susceptible to a wide range of attacks such as XSS, clickjacking, and code injection.
- **Sensitive Data Exposure**: Lack of security headers like Strict-Transport-Security can lead to the interception of sensitive data.
- **Content Spoofing**: Missing headers can allow attackers to manipulate the MIME type of content, leading to content spoofing attacks.
- **Cross-Site Scripting (XSS)**: Absence of Content-Security-Policy increases the risk of XSSattacks.
- **Reputation Damage**: Successful exploits due to missing security headers can damage the reputation of the organization.

CVSS Score

- 4.3

Severity

- **Medium**

Mitigation

- **Implement Content Security Policy (CSP):**
  - Define a CSP to control resources the user agent is allowed to load.
  - Example: Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'
- **Use X-Frame-Options Header:**
  - Prevent clickjacking by restricting framing.
  - Example: X-Frame-Options: SAMEORIGIN
- **X-Content-Type-Options:**
  - Prevent MIME type sniffing.
  - Example: X-Content-Type-Options: nosniff
- **Referrer-Policy:**
  - Control the amount of referrer information sent with requests.
  - Example: Referrer-Policy: no-referrer-when-downgrade
- **Permissions-Policy:**
  - Control which features and APIs can be used in the browser.
  - Example: Permissions-Policy: geolocation=(self), microphone=()

Evidence

Headers missing

- Hence, Security headers are missing in **this application.**

3.4.2 Vulnerability: Browser Cache Weakness

Description

The application fails to properly manage session expiration and caching policies, allowing unauthorized access to protected pages. After logging in, if a user copies the URL and opens it in a new tab or even after closing and reopening the browser, the application directly loads the dashboard **without re- authentication**.

This issue arises due to the lack of proper **cache-control headers** and session expiration mechanisms, which results in sensitive pages being stored in the browser cache and served without verifying the active authentication state.

Impact

- **Unauthorized Access:** Users who should be logged out may still access restricted pages.
- **Session Persistence Risks:** If a user leaves a shared or public system without logging out, another person could access their session.
- **Data Leakage:** Sensitive information remains accessible even after the user session is closed.
- **Compliance Risks:** Violates security best practices such as OWASP A3:2021 - Sensitive Data Exposure and A7:2021 - Identification and Authentication Failures.

CVSS Score

- N/A

Severity

- **Medium**

Mitigation

- **Prevent Browser Caching of Sensitive Pages:**

  Configure the application to instruct browsers **not to store authenticated pages**. Set response headers that prevent caching of protected content.

- **Enforce Session Expiry and Reauthentication:**

  Ensure that each request to the dashboard **validates an active session on the server**.

  Implement session expiration policies and **force users to re-authenticate** after a certain period of inactivity.

- **Invalidate Sessions on Logout or Browser Closure:**

  Ensure that once a user logs out, the session is **immediately invalidated** on the server.

  Configure session cookies to **expire when the browser is closed**.

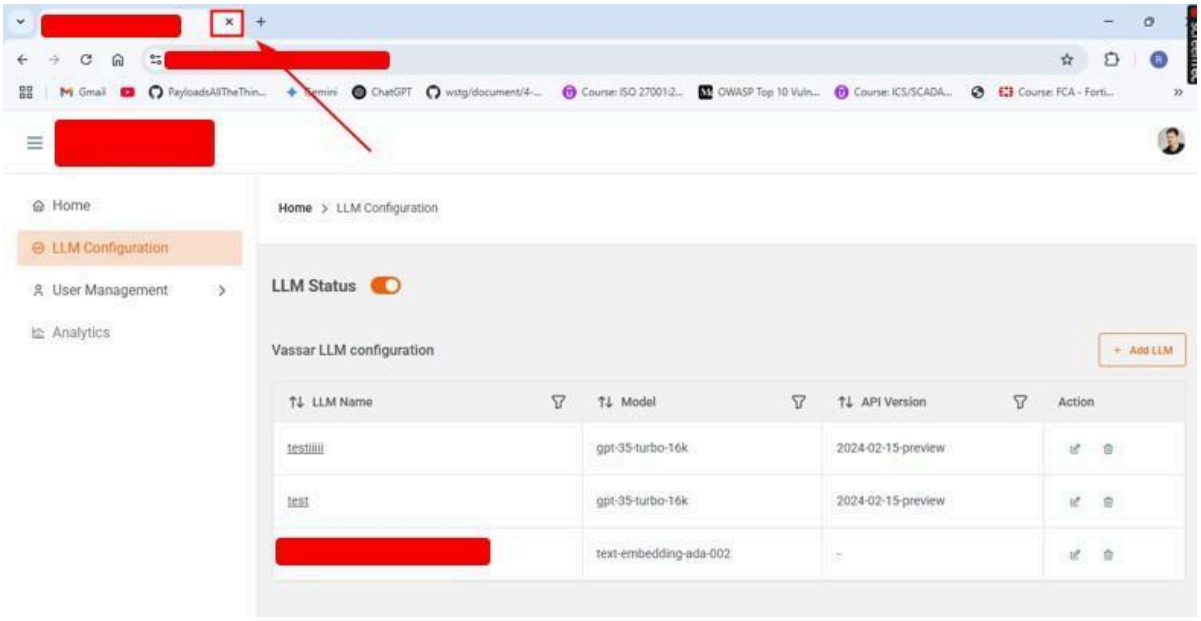
- **Implement Server-Side Authentication Checks:**

  Do not rely on cached authentication states; instead, enforce **session validation on every request**.

- **Monitor and Log Session Activities:**

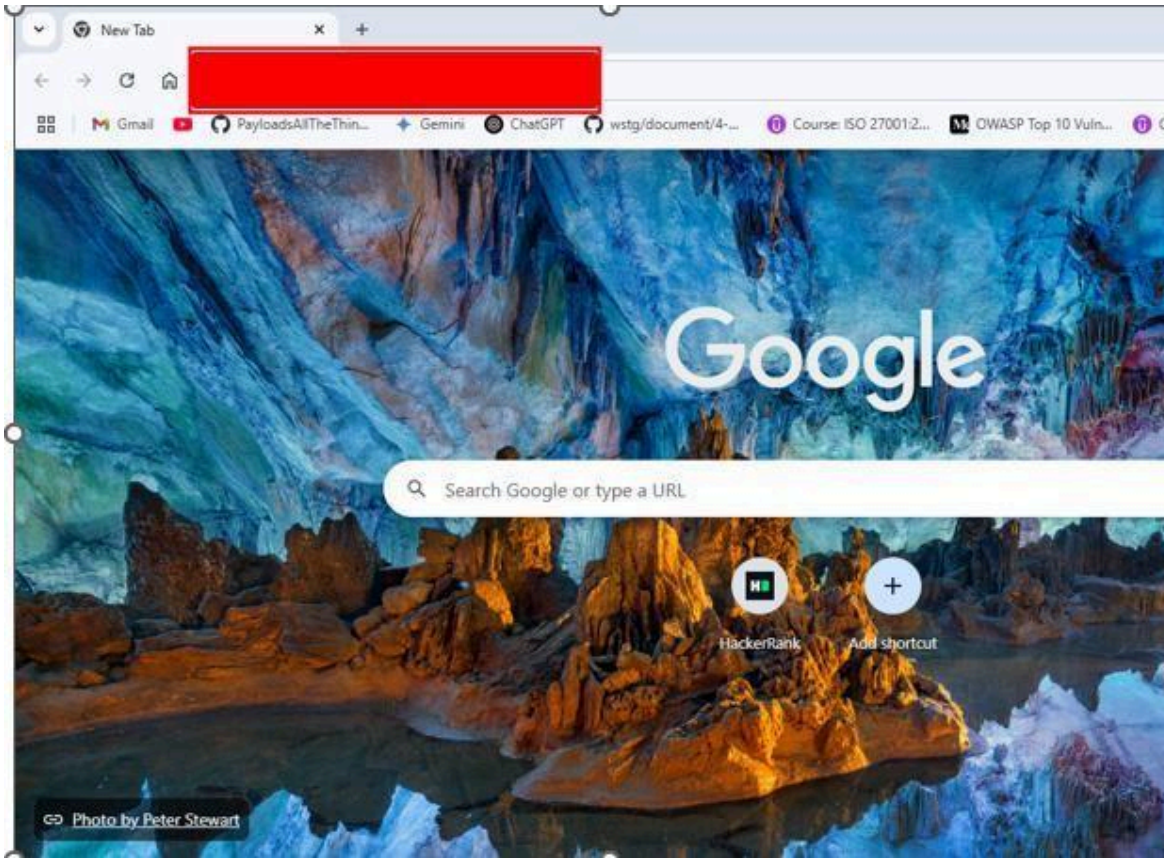
  Implement **session timeout policies** and log out inactive users. Track session activity to detect abnormal login behaviour.

Evidence

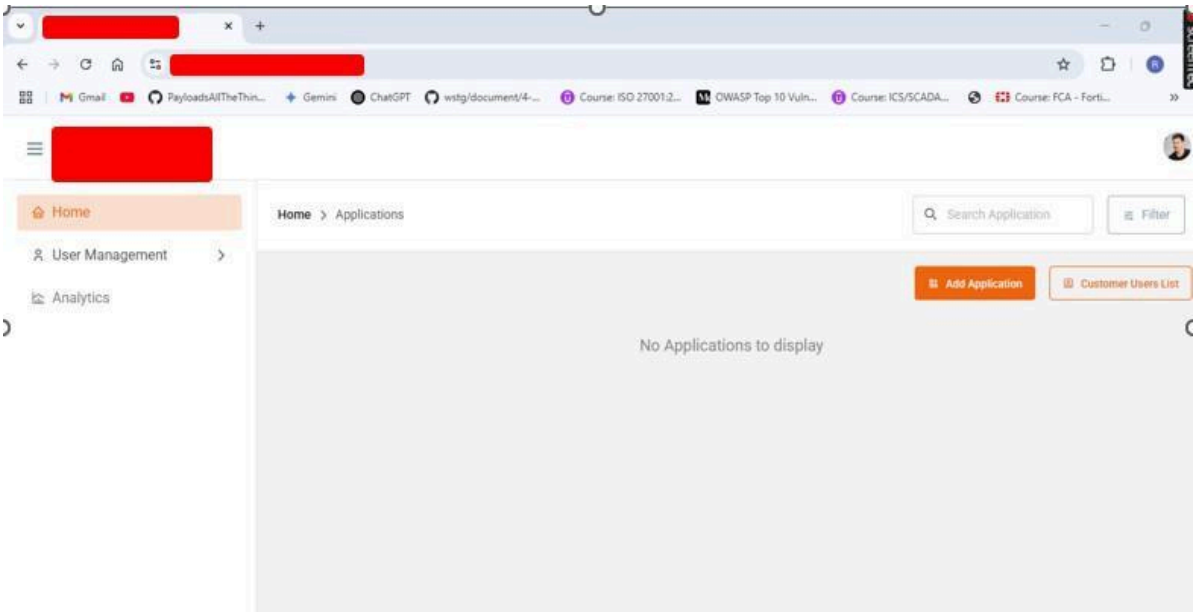


Dashboard view of the application

- After authentication, the above fig shows the dashboard view.
- Let's close the tab and open a new tab by copying the dashboard URL.



Paste the URL in a new tab



- In the above fig, we can see that the dashboard view has opened without any authentication in place.
- The URL should technically be redirected to the login page instead of directly opening the dashboard page.
- Hence, Browser cache weakness is present in the application.

3.4.3 Vulnerability: Improper Error Handling & SQL Validation Issues

Description

During penetration testing, it was observed that the application exposes **detailed SQL error messages** in the HTTP response. When an invalid or malicious payload was injected, the server returned a **500 Internal Server Error**, along with database query details, including table names, column names, and constraints.

This occurs due to **improper error handling** and **lack of input validation**, which allows attackers to gather sensitive database information. Additionally, the application does not sanitize SQL inputs, making it susceptible to **SQL injection attacks**.

Impact

- **Information Disclosure:** Attackers can gather critical database details (table names, column names, constraints).
- **SQL Injection Risks:** If inputs are not properly validated, attackers may exploit SQL injection to manipulate or exfiltrate data.
- **System Stability Issues:** Unhandled exceptions can lead to crashes or performance degradation.
- **Compliance Violations:** Violates security best practices such as OWASP A9:2021 - Security Logging and Monitoring Failures and A3:2021 - Injection Attacks.

CVSS Score

- Can vary(3.8 - 7.0)

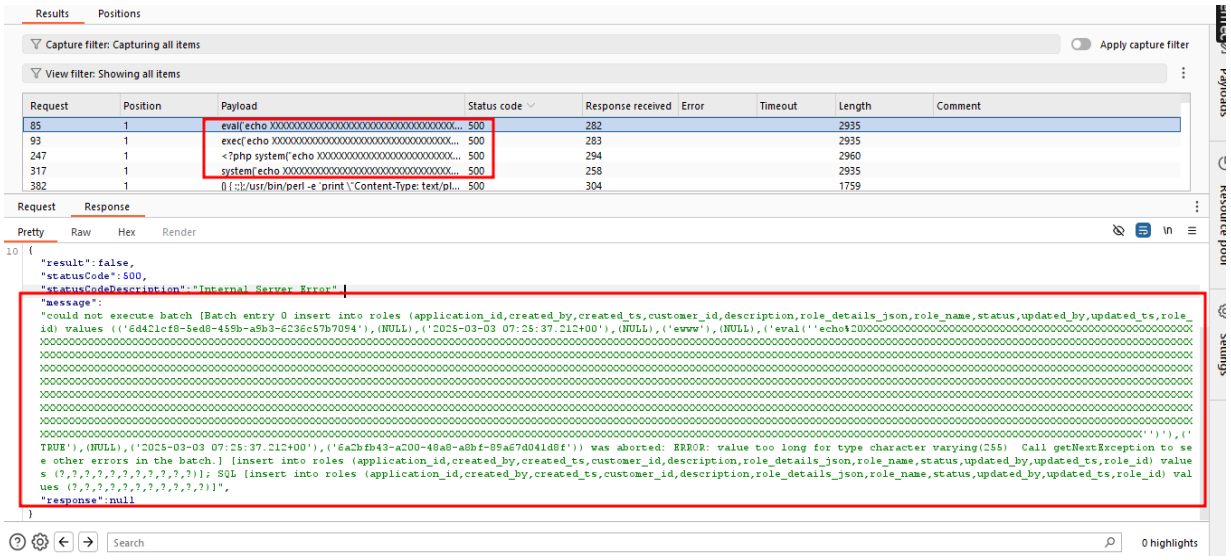
Severity

- **Low**

Mitigation

- Implement proper error handling to return generic messages instead of SQL errors.
- Use parameterized queries and input validation to prevent SQL injection.
- Suppress detailed database errors in production environments.
- Monitor logs for suspicious SQL queries and enforce security controls.

Evidence



500 error with SQL error disclosure

- The application exposes detailed SQL error messages, revealing database structure and internal query logic.
- The application fails to handle exceptions properly, leading to direct SQL error disclosure.

3.4.4 Vulnerability: Clickjacking

Description

Clickjacking, also known as UI redressing, is an attack where a malicious actor tricks a user into clicking on something different from what the user perceives, thereby performing actions without the user's intent. This is typically achieved by overlaying or embedding a transparent or opaque layer over a legitimate webpage element, causing the user to interact with the concealed element.

Impact

- **Compromised Security:** Clickjacking can lead to unintended actions such as changing user settings, initiating financial transactions, or downloading malicious software.
- **Information Disclosure:** Attackers can gain unauthorized access to sensitive information.
- **Session Hijacking:** Attackers can exploit user sessions, leading to unauthorized access to accounts.
- **Reputation Damage:** Users may lose trust in the affected website or service if they are tricked into performing unintended actions**.**

CVSS Score

- 5.4

Severity

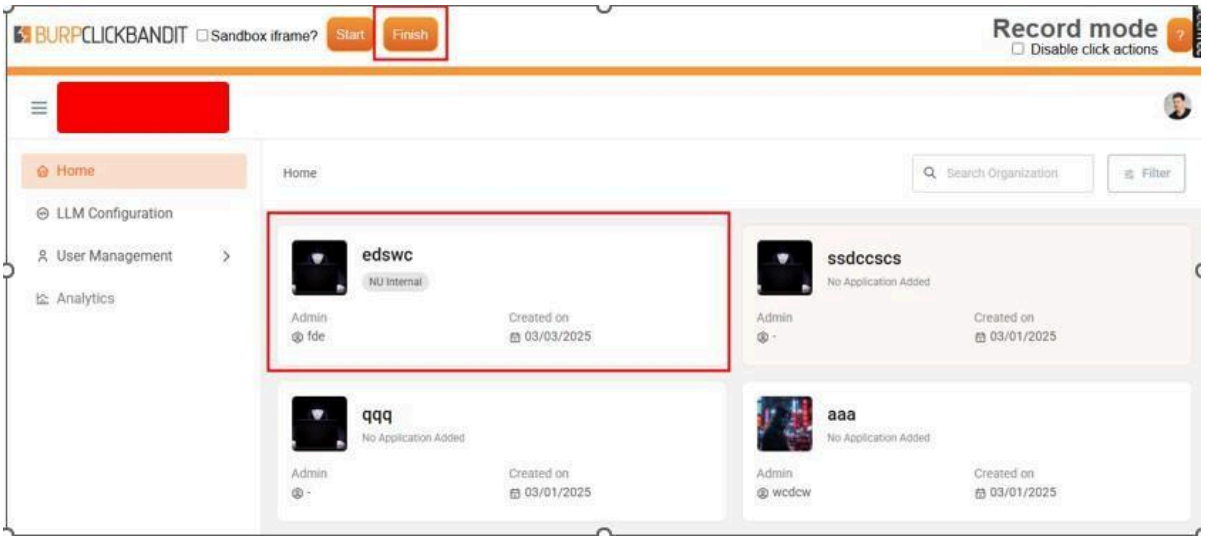
- **Medium**

Mitigation

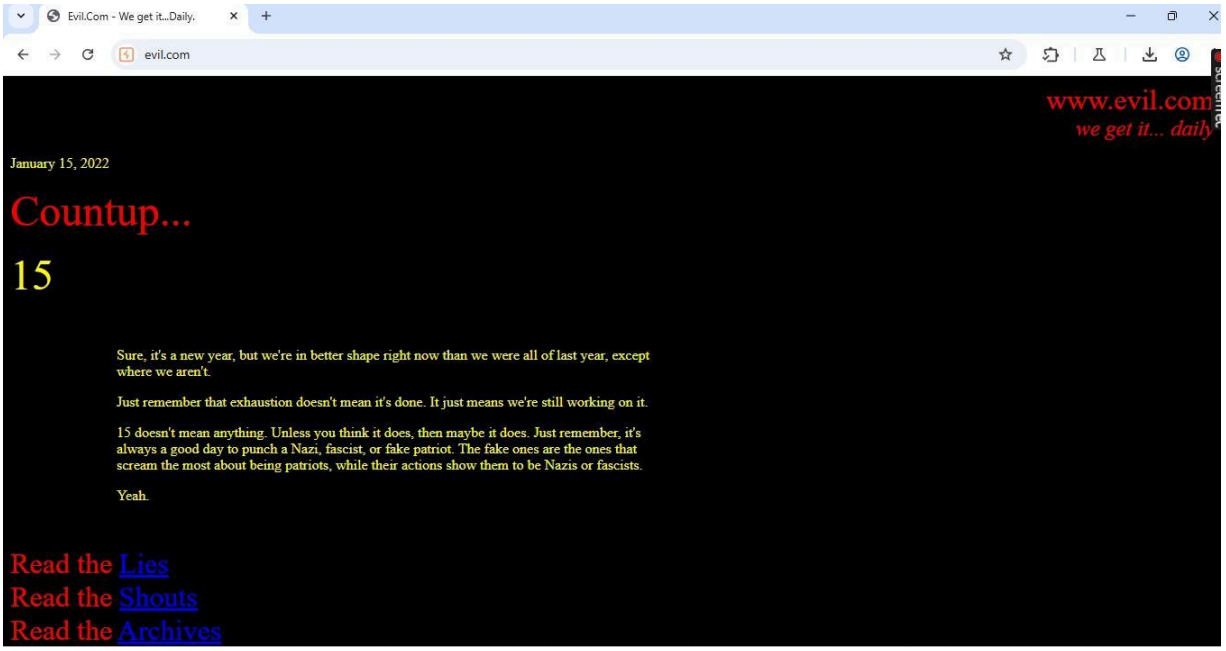
- **Set X-Frame-Options Header:** Use DENY or SAMEORIGIN to prevent unauthorized framing of the web page.
- **Implement Content Security Policy (CSP):** Use frame-ancestors 'self' to restrict iframe embedding to trusted domains.
- **Use JavaScript Frame Busting:** Detect if the page is loaded inside an iframe and force it to break out.
- **Perform Regular Security Testing:** Conduct penetration tests and security audits to detect potential clickjacking vulnerabilities.

Evidence

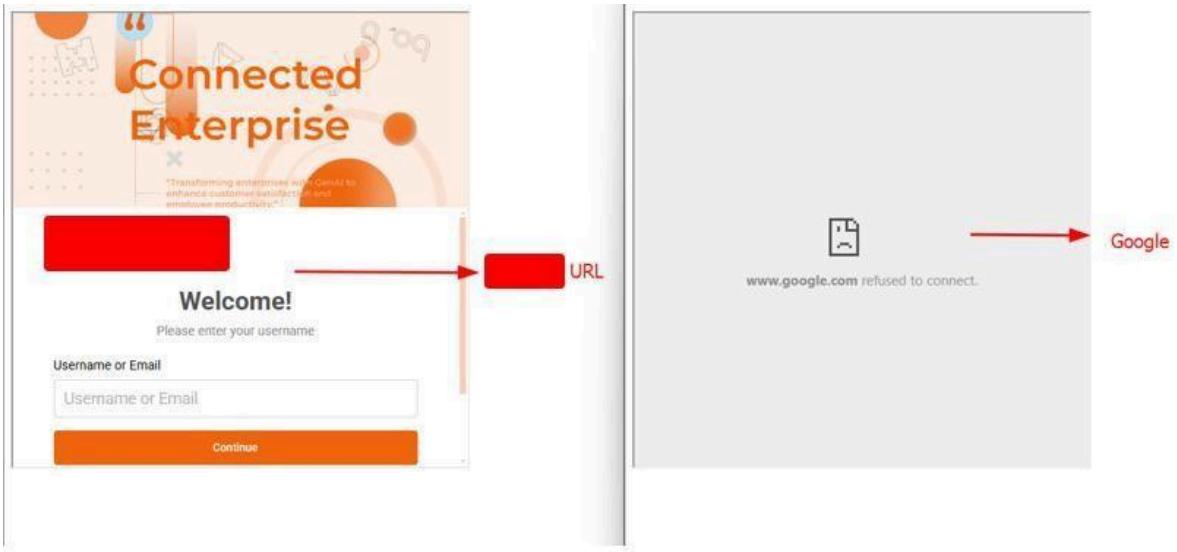
- Paste the code in the inspect elements console.



- Let’s click on the marked button in the above fig to observe the results.



- After clicking, the page has been re-directed to some other URL.



clickjacking not possible in google

- Hence, clickjacking is present in the application.

3.4.5 Vulnerability: Cross-Site Request Forgery Token Manipulation

Description

CSRF tokens are used to prevent unauthorized actions from being performed on behalf of authenticated users. If an attacker can modify, remove, or predict the CSRF token, they may be able to bypass CSRF protection and perform malicious actions.

Impact

- Unauthorized actions on behalf of a legitimate user, such as:
- Changing account details
- Performing transactions
- Modifying permissions
- Potential compromise of sensitive user data
- Account takeovers if combined with other vulnerabilities

CVSS Score

- Can Vary(3 - 8)

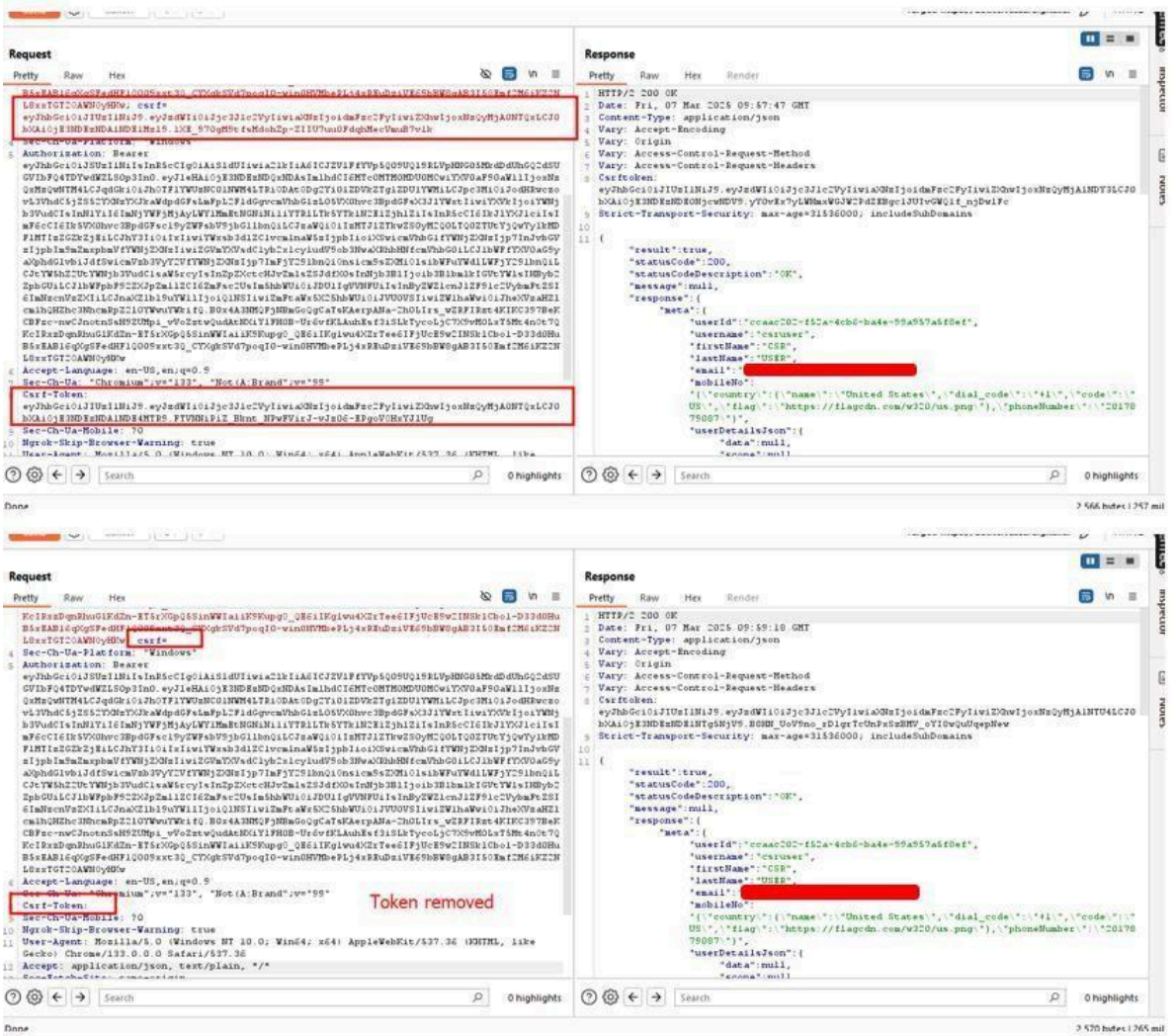
Severity

- **Medium**

Mitigation

- **Use Strong CSRF Tokens:** Generate cryptographically secure, random, and unique tokens for each session or request.
- **Enforce Server-Side Validation:** Ensure that every sensitive request includes a valid CSRF token and reject requests with missing or altered tokens.
- **Bind CSRF Tokens to User Sessions:** Associate tokens with user sessions to prevent reuse across different sessions.
- **Implement HTTP-Only and Secure Cookies:** Store CSRF tokens in HTTP-only cookies and send them via headers to prevent client-side manipulation.
- **Set the SameSite Cookie Attribute:** Use SameSite=Strict or SameSite=Lax to prevent unauthorized cross- site requests.

Evidence



- Here, we can modify the token or completely remove the token in the request page. The response will still be 200.

3.5 Low Severity Findings

3.5.1 Vulnerability: CORS Misconfiguration

Description

Cross-Origin Resource Sharing (CORS) is a security feature implemented by web browsers to prevent unauthorized cross-origin requests. A misconfigured CORS policy may allow unauthorized websites to interact with a web application's resources, leading to data theft, unauthorized actions, or other security risks.

Impact

- **Unauthorized API Access:** Attackers can make unauthorized requests on behalf of authenticated users.
- **Sensitive Data Exposure:** Leaking confidential information due to overly permissive CORS policies.
- **Account Takeover** (if combined with other attacks): Exploiting CORS misconfigurations along with session hijacking or CSRF to take over user accounts.
- **Client-Side Code Injection:** If Access-Control-Allow-Origin: * is set and combined with JSONP endpoints, attackers can execute malicious scripts**.**

CVSS Score

- Can Vary(3.5-7)

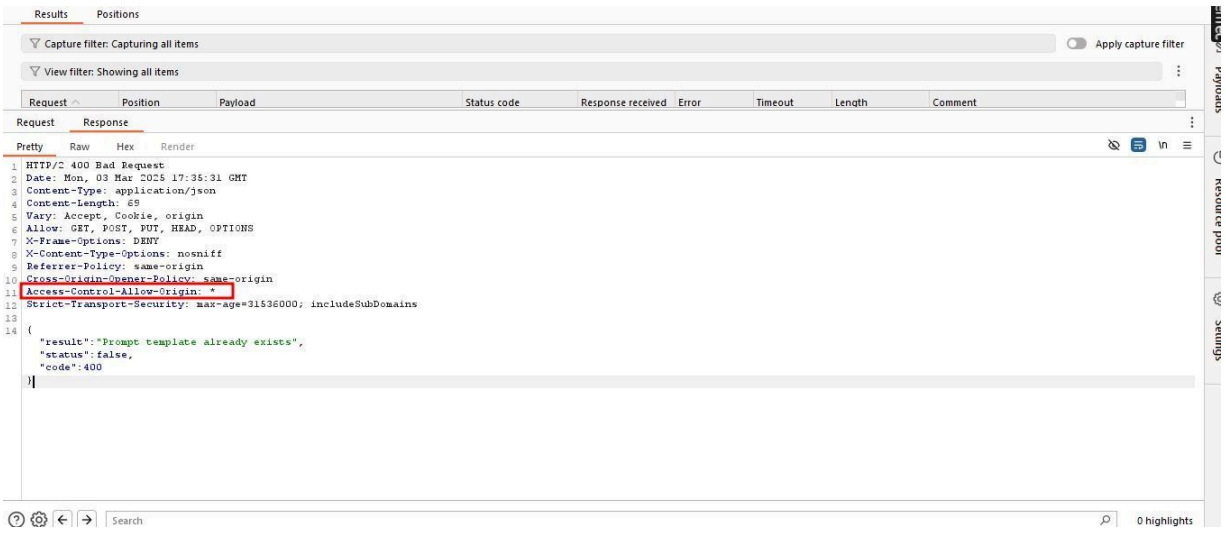
Severity

- **Low**

Mitigation

- **Avoid Using Access-Control-Allow-Origin: *:** Restrict allowed origins to trusted domains only.
- **Use a Proper Allowlist:** Define specific, trusted domains instead of allowing all origins.
- **Restrict HTTP Methods:** Allow only necessary HTTP methods (e.g., GET, POST) and avoid exposing sensitive operations.
- **Disable Credential Sharing:** Set Access-Control-Allow-Credentials: false unless necessary, preventing unauthorized websites from using user sessions.

Evidence



3.5.2 Vulnerability: TLS Cookie without Secure Flag Set

Description

Cookies are often used to store session tokens, authentication credentials, and other sensitive data. If a cookie is set without the Secure flag, it can be transmitted over unencrypted HTTP connections. This allows attackers to intercept the cookie using **Man-in-the-Middle (MitM)** attacks, potentially leading to session hijacking and unauthorized access.

Impact

- **Session Hijacking:** Attackers can steal authentication cookies and impersonate users.
- **Data Exposure:** Sensitive information stored in cookies can be accessed over unsecured connections.
- **Increased Risk of MITM Attacks:** An attacker can capture cookies on unsecured public networks (e.g., Wi-Fi hotspots).
- **Bypassing Authentication Protections:** Attackers may reuse stolen cookies to access user accounts without needing credentials.

CVSS Score

- N/A

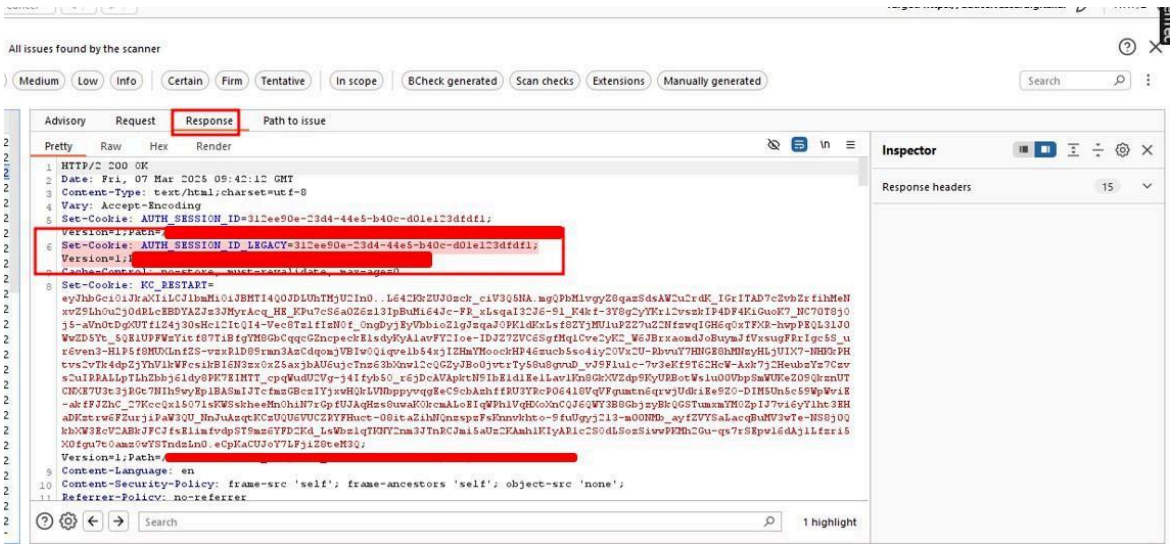
Severity

- **Low**

Mitigation

- Ensure all authentication and session cookies have the Secure attribute set, forcing them to be transmitted only over HTTPS.
- Ensure web servers (e.g., Apache, Nginx, IIS) are configured to enforce HTTPS and apply the Secure flag to cookies.
- Use SameSite=Strict or SameSite=Lax to restrict how cookies are sent with cross-site requests.

Evidence



Cookies and session id leaked in response

3.5.3 Vulnerability: Expired Token still valid

Description

Session tokens are used to authenticate users and maintain active sessions. If an application fails to properly invalidate expired session tokens, an attacker or legitimate user can reuse old, expired tokens to gain unauthorized access. This typically occurs due to improper session management, weak token expiration enforcement, or lack of server-side validation.

Impact

- **Unauthorized Access:** Attackers can reuse expired session tokens to log in as legitimate users.
- **Session Hijacking:** If an attacker captures an expired token, they can exploit it to maintain persistent access.
- **Bypassing Logout Mechanisms:** Users who log out expecting their session to be terminated may still be vulnerable if the expired token remains valid.

CVSS Score

- N/A

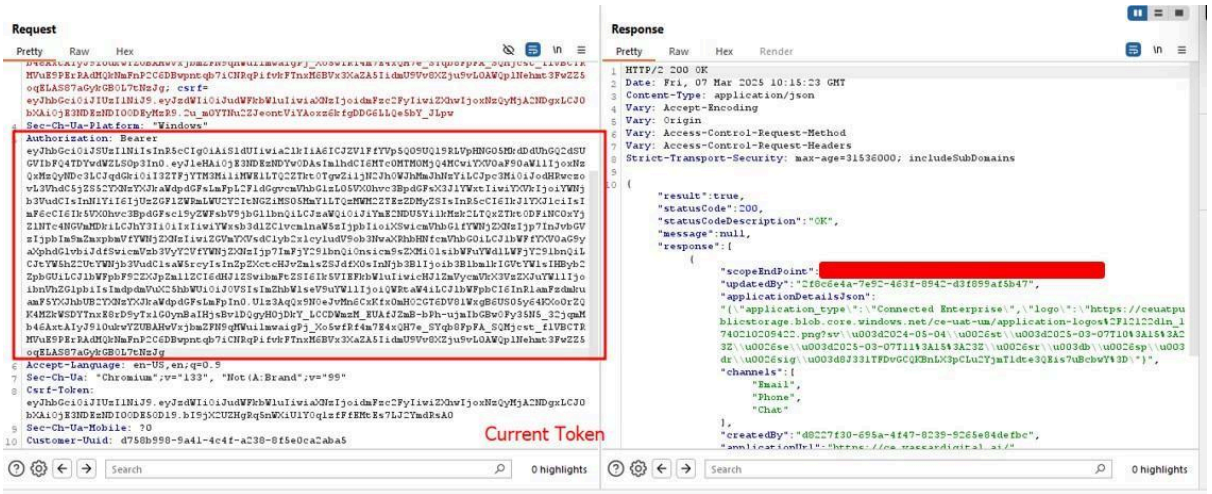
Severity

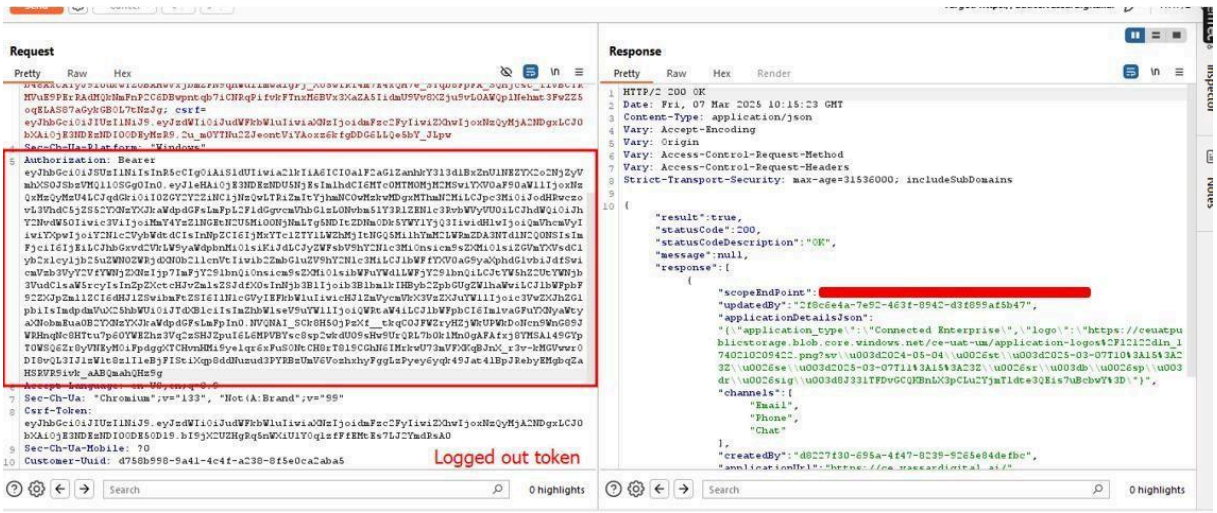
- **Low**

Mitigation

- Implement proper session expiration on the server side and reject expired tokens.
- Ensure that logging out completely destroys session tokens, both on the client and server.
- Issue short-lived access tokens and require reauthentication or refresh tokens to continue sessions.

Evidence





- In the above fig, the request was sent with an expired token and the response has been 200.
- The issue is categorized as a low-level severity because the time taken for the expired token to work is less

.

3.6 Security status according to OWASP Top 10

#	Vulnerability	Description	Status
A01	Broken Access Control	Access controls enforce policies so that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure or modification, destruction of data, or performing a business function outside the user's limits.	**Fail** **(Refer to 3.2.1)**
A02	Cryptographic Failures	Cryptographic Failures involve protecting data in transit and at rest. This includes passwords, credit card numbers, health records, personal information, and business secrets that require extra protection, especially if that data falls under privacy laws such as GDPR or regulations like PCI Data Security Standard (PCI DSS) for financial data.	**Pass**
A03	Injection	An application is at risk when user-supplied data is not validated, filtered, or sanitized by the application; dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter; hostile data is used within object-relational	**Pass**

		mapping (ORM) search parameters to extract additional, sensitive records; or when hostile data is directly used or concatenated.	
A04	Insecure Design	According to OWASP, “Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. Secure design requires a secure development lifecycle, some form of secure design pattern or paved road component library or tooling, and threat modelling.”	**Fail** **(Refer to 3.4.3)**
A05	Security Misconfiguration	This category includes such things as missing security hardening across any part of the application stack, improperly configured permissions on cloud services, any unnecessary features that are enabled or installed, and unchanged default accounts or passwords. The former category XML External Entities (XXE) is now included in Security Misconfiguration.	**Fail** **(Refer to 3.4.1,3.4.2,3.4.4,3.5.1)**
A06	Vulnerable and Outdated Components	This category includes any software that is vulnerable, unsupported, or out of date. If you do not know the versions of your components – including all direct and indirect dependencies – or you do not regularly scan and test your components, you are likely at risk.	**Pass**
A07	Identification and Authentication Failures	Security risk occurs when a user’s identity, authentication, or session management is not properly handled, allowing attackers to exploit passwords, keys, session tokens, or implementation flaws to assume users’ identities temporarily or permanently.	**Fail** **(Refer to 3.2.1,3.4.5,3.5.2,3.5.3)**
A08	Software and Data Integrity Failures	This includes software updates, critical data, and CI/CD pipelines that are implemented without verification. An example of this includes objects or data encoded or serialized into a structure that an attacker can modify. Another example is an application that relies upon plugins, libraries, or modules from untrusted sources. Insecure CI/CD pipelines that can introduce the potential for unauthorized access, malicious code, or system compromise also fit into this category. Lastly, applications with auto-update functionality, in which updates are downloaded without	**N/A**

		sufficient integrity verification and applied to a previously trusted application, are considered software and data integrity failures because attackers could infiltrate the supply chain to distribute their own malicious updates.	
A09	Security Logging and Monitoring Failures	This category includes errors in detecting, escalating, and responding to active breaches. Without logging and monitoring, breaches cannot be detected. Examples of insufficient logging, detection, and monitoring include not logging auditable events like logins or failed logins, warnings and errors that generate inadequate or unclear log messages, or logs that are only stored locally. Failures in this category impact visibility, incident alerting, and forensics.	**N/A**
A10	Server-Side Request Forgery	Server-Side Request Forgery occurs when a web application fetches a remote resource without validating the user-supplied URL. An attacker can coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network ACL. Though SSRF shows a relatively low incidence rate in the data OWASP reviewed, this category was added based on the industry survey results; users are concerned that SSRF attacks are becoming more prevalent and potentially more severe due to increased use of cloud services and the complexity of architectures	**Pass**

4. Tests Performed

Test name	Pass	Fail	N/A
Information Gathering			
Conduct Search Engine Discovery Reconnaissance for Information Leakage	Yes		
Fingerprint Web Server	Yes		
Review Webserver Metafiles for Information Leakage	Yes		
Enumerate Applications on Webserver	Yes		
Review Webpage Content for Information Leakage	Yes		
Identify Application Entry Points	Yes		
Map Execution Paths Through Application	Yes		
Fingerprint Web Application Framework	Yes		
Fingerprint Web Application	Yes		
Map Application Architecture			Yes
Configuration & Deployment Management Testing			
Test Network Infrastructure Configuration			Yes
Test Application Platform Configuration			Yes
Test File Extensions Handling for Sensitive Information	Yes		
Review Old Backup and Unreferenced Files for Sensitive Information			Yes
Enumerate Infrastructure and Application Admin Interfaces	Yes		
Test HTTP Methods	Yes		
Test HTTP Strict Transport Security	Yes		
Test File Permission	Yes		
Test Cloud Storage			Yes
Testing for Content Security Policy	Yes		

Test Path Confusion			Yes
Identity Management Testing			
Test Role Definitions	Yes		
Test User Registration Process			Yes
Test Account Provisioning Process			Yes
Testing for Account Enumeration and Guessable User Account	Yes		
Testing for Weak or Unenforced Username Policy	Yes		
Authentication Testing			
Testing for Default Credentials	Yes		
Testing for Weak Lock Out Mechanism	Yes		
Testing for Bypassing Authentication Schema		Yes	
Testing for Vulnerable Remember Password	Yes		
Testing for Browser Cache Weaknesses		Yes	
Testing for Weak Password Policy	Yes		
Testing for Weak Security Question Answer			Yes
Testing for Weak Password Change or Reset Functionalities			Yes
Testing for Weaker Authentication in Alternative Channel			Yes
Testing Multi-Factor Authentication (MFA)			Yes
Authorization Testing			
Testing Directory Traversal File Include		Yes	
Testing for Bypassing Authorization Schema	Yes		

Testing for Privilege Escalation	Yes		
Testing for Insecure Direct Object References	Yes		
Testing for OAuth Weaknesses			Yes
Testing for OAuth Authorization Server Weaknesses			Yes
Testing for OAuth Client Weaknesses			Yes
Session Management Testing			
Testing for Session Management Schema	Yes		
Testing for Cookies Attributes	Yes		
Testing for Session Fixation	Yes		
Testing for Exposed Session Variables	Yes		
Testing for Cross Site Request Forgery	Yes		
Testing for Logout Functionality	Yes		
Testing Session Timeout		Yes	
Testing for Session Puzzling		Yes	
Testing for Session Hijacking	Yes		
Testing JSON Web Tokens			Yes
Testing for Concurrent Sessions		Yes	
Input Validation Testing			
Testing for Reflected Cross Site Scripting	Yes		
Testing for Stored Cross Site Scripting	Yes		
Testing for HTTP Parameter Pollution	Yes		
Testing for SQL Injection	Yes		
Testing for Oracle			Yes
Testing for MySQL	Yes		
Testing for SQL Server	Yes		
Testing PostgreSQL	Yes		

Testing for MS Access			Yes
Testing for NoSQL Injection			Yes
Testing for ORM Injection			Yes
Testing for Client-side			Yes
Testing for LDAP Injection			Yes
Testing for XML Injection			Yes
Testing for SSI Injection	Yes		
Testing for XPath Injection	Yes		
Testing for IMAP SMTP Injection	Yes		
Testing for File Inclusion	Yes		
Testing for Command Injection	Yes		
Testing for Format String Injection	Yes		
Testing for Incubated Vulnerability	Yes		
Testing for HTTP Splitting (Protocol downgrade)	Yes		
Testing for HTTP Incoming Requests	Yes		
Testing for Host Header Injection	Yes		
Testing for Server-side Template Injection			Yes
Testing for Server-Side Request Forgery	Yes		
Testing for Mass Assignment	Yes		
Testing for Error Handling			
Testing for Improper Error Handling		Yes	
Testing for Weak Cryptography			
Testing for Weak Transport Layer Security			Yes
Testing for Padding Oracle			Yes
Testing for Sensitive Information Sent via Unencrypted Channels			Yes
Testing for Weak Encryption			Yes
Business Logic Testing			
Test Business Logic Data Validation	Yes		
Test Ability to Forge Requests	Yes		
Test Integrity Checks	Yes		

Test for Process Timing	Yes		
Test Number of Times a Function Can Be Used Limits	Yes		
Testing for the Circumvention of Work Flows	Yes		
Test Defences Against Application Misuse	Yes		
Test Upload of Unexpected File Types		Yes	
Test Upload of Malicious Files		Yes	
Test Payment Functionality			Yes
Client-Side Testing			
Testing for DOM-Based Cross Site Scripting	Yes		
Testing for Self-DOM Based Cross-Site Scripting	Yes		
Testing for JavaScript Execution	Yes		
Testing for HTML Injection	Yes		
Testing for Client-side URL Redirect	Yes		
Testing for CSS Injection	Yes		
Testing for Client-side Resource Manipulation	Yes		
Testing Cross Origin Resource Sharing	Yes		
Testing for Cross Site Flashing	Yes		
Testing for Clickjacking		Yes	
Testing WebSockets			
Testing Web Messaging			
Testing Browser Storage	Yes		
Testing for Reverse Tabnabbing	Yes		

5. Conclusion

During the penetration testing of Organization's web application, multiple security vulnerabilities were identified, ranging from **critical to low severity issues**. The most severe finding, **Session Hijacking via Insecure Session Management**, poses a significant risk as it allows attackers to hijack user sessions and gain unauthorized access. Additionally, **CSRF token manipulation** and **expired token reuse** indicate weaknesses in authentication mechanisms that could lead to session persistence and unauthorized actions. The presence of **SQL validation issues** and **improper error handling** further exposes the application to potential injection attacks and information disclosure.

Furthermore, medium to low-severity findings, such as **missing security headers, clickjacking, and CORS misconfigurations**, indicate areas where security best practices are not fully implemented. These vulnerabilities could be leveraged by attackers to exploit other weaknesses in the system. Addressing these issues promptly by implementing **secure session management, strict input validation, enforcing HTTP security headers, and improving error handling mechanisms** will significantly enhance the security posture of Organization's web application, ensuring the protection of sensitive patient and organizational data.