



Web Application Vulnerability

Scan Report

PREPARED BY **ZEROTHREAT AI**

Host : <https://www.parihaaram.in>

JANUARY 14, 2026

Comprehensive Security Coverage with ZeroThreat Scanner

How ZeroThreat Scan?

The security scan is executed using a dedicated, fully secured private container assigned exclusively to the target, in alignment with ZeroThreat's adherence to Zero Trust Architecture principles. It also integrates security best practices from frameworks such as the Microsoft Cloud Security Benchmark and the Well-Architected Framework. All scan data is securely handled, and critical parameters are encrypted using industry-standard cryptographic algorithms like AES, 3DES, and RSA.

The ZeroThreat Scanner follows the **OWASP Application Security Verification Standard (ASVS)** when testing applications. It prioritizes vulnerabilities based on recent trends and high-severity reports from sources like CVE and NIST, focusing on critical to medium-severity issues.

The scanner first crawls every page of the application, extracting even hidden form elements and separately categorizing API requests. It then initiates a security test similar to what a penetration tester would perform—identifying high-risk vulnerabilities such as session hijacking, information disclosure, input validation flaws, memory corruption, privilege escalation, and business logic errors.

Following this, it performs a series of injection attacks on every tracked request payload, including XSS, SQL Injection, Command Injection, Server-Side Template Injection, and more. In total, over 40,000 security tests are conducted to uncover vulnerabilities. The goal is to evaluate the application's entire security landscape rather than focusing on isolated components. Additional security tests performed by ZeroThreat Scanner include:

1. SSL Certificate Scan

The scanner validates key SSL parameters such as protocol support, key exchange, cipher strength, certificate chain, Diffie-Hellman parameter size, and known certificate vulnerabilities.

2. Vulnerable JavaScript Package Detection

The scanner analyzes all JavaScript packages used in the application, checks their versions, and compares them with an up-to-date CVE database. Currently, over 18,000 CVEs are mapped for JavaScript package vulnerability detection.

3. Server-Side Technology Detection

The scanner identifies the backend technology stack and checks whether known vulnerabilities affect the specific versions detected.

4. Server Open Ports Scan

One of the fastest scanners in the industry, ZeroThreat efficiently detects open server ports and highlights any that are critically exposed.

5. Mail Configuration Assessment

The scanner tests the target's email configuration, checking for vulnerabilities like email spoofing and other known email-related security flaws.

6. Sensitive Data Exposure Detection

Using AI, the scanner detects sensitive data values exposed in HTTP responses during the scan.

7. Secret Data Exposure Detection

The AI engine also identifies and reports exposed secrets from leading providers such as Amazon, Google, Azure, and API keys.

Additionally, the scanner assesses compliance requirements and generates reports accordingly.

Contents

1. Executive Summary	5
2. Technical Summary	9
3. Scan Information	15
3.1. Detected Vulnerabilities	16
3.2. Compliance Test Result	17
3.3. SSL Certificate Information	22
3.4. Server Ports Scan	23
3.5. Mail Configuration	24
3.6. Javascript Packages	25
4. Technical Details: Findings and Recommendations	26
4.1. ClickJacking : Improper Restriction of Rendered UI Layers or Frames	26
4.2. Content Security Policy Not Implemented	28
4.3. Sensitive Data Stored in Local Storage	30
4.4. Sensitive Data Stored in Session Storage	31
4.5. Strict Transport Security Not Enforced	32
4.6. X-Content-Type-Options Header Missing	35

1. Executive Summary

The Dynamic Application Security Testing (DAST) scan conducted on the application hosted at <https://www.parihaaram.in> between January 14, 2026, 17:09:38 UTC and January 14, 2026, 17:18:57 UTC encompassed an extensive evaluation of 84 URLs and delivered critical insights pertinent to the security posture of the web application. The primary objective was to identify exploitable vulnerabilities stemming from weaknesses in the application's configuration, implementation, and security controls at runtime. The findings elucidate the security landscape by highlighting deficiencies that affect the confidentiality, integrity, and availability of the application and its users, thus providing a comprehensive appraisal essential for informed risk mitigation decisions.

The security assessment revealed several medium and low severity vulnerabilities that collectively underscore engaging risks which, if left unaddressed, could be leveraged by malicious actors to compromise user trust, exfiltrate sensitive information, and execute attacks such as UI redressing and data theft. The report's vulnerability profile does not include any critical severity findings, yet the identified concerns are significant enough, especially those classified at medium severity, to warrant immediate attention and prioritized remediation efforts.

A principal area of concern is the absence and improper implementation of security policies that govern browser behavior and content handling. The lack of a robust Content Security Policy (CSP) represents one of the notable medium-severity risks identified. CSP is an essential security control designed to mitigate a broad spectrum of injection attacks, including Cross-Site Scripting (XSS) and data injection attacks, by restricting the sources from which a web page can load resources such as scripts, stylesheets, or iframes. By failing to implement an effective CSP, the application leaves open avenues through which attackers could inject malicious content, consequently jeopardizing end-user security, facilitating session hijacking, and compromising the privacy of sensitive interactions.

Equally significant is the vulnerability associated with clickjacking, another medium-severity issue that emerged prominently from the scan. Clickjacking attacks involve tricking users into clicking on concealed or disguised interface elements embedded within transparent or opaque layers. This manipulation can result in unauthorized actions such as changing settings, initiating transactions, or disclosing confidential information without the user's awareness or consent. The scan's identification of improper restrictions on user interface layers and frames signals a critical weakness in the application's defensive fencing against UI redressing exploits. This absence of appropriate frame options or frame boundary controls such as X-Frame-Options or frame-ancestors directives furthers the potential for adversaries to mount clickjacking attacks, thus compromising application integrity and user trust.

Another vital dimension detected during the scan concerns the handling and storage of sensitive data on the client side, specifically within local and session storage mechanisms of web browsers. The storage of sensitive or confidential information in local storage and session storage, which are accessible via client-side scripts, presents a medium level of risk. Although convenient for performance and user experience improvements, this practice exposes delicate data to theft through cross-site scripting and other client-side attacks. Such stored data may include authentication tokens, personal identifiable information (PII), or application state details which, if accessed by unauthorized actors, could translate into session impersonation, credential theft, and data breach incidents. The identification of these findings indicates a need to reassess client-side data management policies rigorously and incorporate secure handling paradigms aligned with best industry practices, such as using secure HTTP-only cookies or encrypted storage mechanisms where feasible.

Low-severity findings, though less immediately dangerous, represent systemic weaknesses often linked to the application's reliance on secure communication protocols and headers that enforce security by design. The scan detected six instances where Strict Transport Security (HSTS) policies were not enforced. HSTS is a protocol mechanism that forces web browsers to interact with websites only over secure HTTPS connections, thereby preventing protocol downgrade attacks and cookie hijacking via unsecured HTTP channels. The absence of enforced HSTS headers allows for the possibility that users may inadvertently access the site through unsecured connections, thus exposing them to man-in-the-middle interception or session manipulation.

Alongside HSTS omissions, seven instances of missing X-Content-Type-Options headers were found. This security header instructs browsers to adhere strictly to MIME types as declared by the server and prevents MIME sniffing. MIME sniffing is a technique sometimes leveraged by attackers to execute cross-site scripting attacks by tricking the browser into interpreting files as executable scripts. The omission of this header, therefore, subtly undermines the defense-in-depth strategy by potentially enabling content injection or execution attacks, further compounding the risk landscape.

Collectively, while none of the vulnerabilities surfaced represent outright critical risks, their cumulative presence paints a concerning picture of latent deficiencies in security best practices foundational to web application security. Their exploitation could impact system confidentiality, integrity, and availability, degrade the user experience, and invite regulatory scrutiny or breach notifications, especially in sectors handling sensitive personal or financial data. Moreover, these weaknesses transcend mere technical liabilities and touch upon broader organizational security governance and lifecycle management processes, encompassing secure coding practices, security configuration management, testing regimes, and escalation protocols.

This DAST scan displays the imperative necessity for the implementation of a coherent and layered security strategy. Immediate remedial action should focus on establishing strict content security policies that effectively curtail the ability of malicious actors to introduce unsafe or untrusted web content. The use of carefully crafted CSP directives tailored to the application's functional requirements is paramount, accompanied by continuous monitoring and tuning aligned with evolving threat landscapes.

Simultaneously, fortifying the application against UI-level attacks such as clickjacking demands enforcing relevant HTTP response headers like X-Frame-Options or frame-ancestor policies, thereby restricting framing of application content only to trusted origins or disallowing framing altogether where possible. This approach will rigorously defend against unauthorized UI redressing assaults that can manipulate user interactions.

Amending the approach towards client-side sensitive data management is equally critical. Revisiting stored data security policies with an emphasis on minimizing exposure within local and session storage is recommended. Alternative secure storage paradigms or encryption strategies must be explored and employed to reduce the risk of data leakage significantly. Furthermore, the principles of least privilege and data minimization should guide all related data handling decisions to ensure that confidential details are never unnecessarily exposed on the client side.

Implementation of HTTPS-related security mechanisms, primarily the strict enforcement of HSTS headers, is necessary to guarantee that no insecure communication channels exist that could derail the integrity of web sessions or compromise user data in transit. This measure is foundational and should be non-negotiable, supplemented by cryptographic best practices such as TLS 1.3 adoption, secure cipher suites, and proper certificate management.

In the same vein, introducing and diligently maintaining headers like X-Content-Type-Options bolsters defenses against subtle MIME-based attacks and enforces browsers' compliance with server-declared content types—a crucial step in preventing potential script injection or execution anomalies.

Beyond the technical corrections, the scan underscores the need for sustained security awareness and vigilance embedded into the development and deployment lifecycle processes. Regular automated and manual security testing must become integral activities to detect regressions or emergent risks swiftly. Developers and administrators alike should be continuously educated on secure coding standards, emerging threat vectors, and remedial strategies to solidify an organizational culture geared toward proactive security.

Equally, appropriate security policies, access controls, incident response plans, and governance frameworks must be articulated, maintained, and tested. Security is not solely a technical endeavor but an organizational commitment requiring coordination across teams, transparency in defect reporting, and prioritization of resources aligned with risk appetite and compliance requirements.

The findings from this DAST scan offer a transparent view of gaps that present realistic risks if exploited, providing a roadmap for elevating the security maturity of the application. By addressing these vulnerabilities systematically and comprehensively, the organization can significantly reduce its attack surface and enhance the resilience of <https://www.parihaaram.in> against diverse threats. This process not only protects end-users and preserves data integrity but also strengthens the overall trustworthiness and reputation of the service.

Ultimately, the report reinforces the notion that security is an ongoing, evolving challenge necessitating commitment, resources, and expertise. The detected medium and low severity vulnerabilities illustrate the common landscape of implementation oversights prevalent in modern web applications, yet they are entirely remediable through disciplined security engineering and management. Prioritizing these fixes will align the application with industry-recognized best practices and help safeguard both organizational assets and stakeholder interests in an increasingly hostile and dynamic cyber environment.

2. Technical Summary

The dynamic application security testing (DAST) scan of the web application at <https://www.parihaaram.in> was conducted beginning at 2026-01-14T17:09:38.600Z and concluded at 2026-01-14T17:18:57 GMT, spanning approximately 9 minutes and 19 seconds. During this period, a total of 84 distinct URLs within the application were thoroughly probed and analyzed. The objective was to identify security weaknesses, misconfigurations, and practices that could potentially be exploited by malicious actors, thereby safeguarding the confidentiality, integrity, and availability of the application and its users' data.

This comprehensive analysis revealed several vulnerabilities across the application, categorized as low and medium severity issues. The absence of information classified as informational vulnerabilities indicates that the scan primarily focused on concrete security risks rather than benign informational artifacts.

Throughout this summary, we will analyze the context and impact of these findings holistically without isolating individual vulnerabilities directly, drawing correlations between them to understand systemic issues inherent in the application's architecture and security posture. In addition, clear guidance will be provided for remediation priorities, empowering the development and security teams to effectively allocate resources to maximize mitigation efficacy.

A prominent theme emerging from the scan is the partial enforcement of fundamental HTTP security headers, which protect users from a broad spectrum of web-based attacks. It is evident that while some basic header configurations have been applied, critical headers such as Content Security Policy (CSP) have not yet been implemented. The absence of CSP introduces substantial risk, as it leaves the application vulnerable to Cross-Site Scripting (XSS) and data injection attacks by allowing arbitrary content execution, including scripts from untrusted sources. Without CSP, the browser is unable to reliably guard against injection-based attacks, thereby compromising user data confidentiality and session integrity.

Similarly, the header for HTTP Strict Transport Security (HSTS), which instructs browsers to automatically enforce HTTPS connections, has not been entirely implemented across all site pages. While the scan found six URLs where HSTS was missing or improperly configured, the inconsistent application of this policy weakens the overall security model. HSTS is a critical safeguard as it mitigates man-in-the-middle attacks, such as protocol downgrade and cookie hijacking, by ensuring encrypted communication channels. Its partial absence undermines the confidentiality and integrity of user sessions.

Additionally, the scan revealed that the X-Content-Type-Optionsheader, which prevents MIME sniffing vulnerabilities, is missing from seven URLs. Without this header, browsers may incorrectly interpret file types, potentially allowing malicious content to be executed inappropriately, leading to cross-site scripting or other script injection attacks. Its absence thus broadens the attack surface and can facilitate exploitation vectors against user agents.

Another important security mechanism impacted is framing protections. The application currently exhibits a lack of adequate clickjacking defenses across multiple pages. Clickjacking vulnerabilities, identified in 10 URLs, occur when attackers embed the application's UI within iframes on malicious sites, tricking users into performing unintended actions. The root cause is insufficient or missing frame-ancestors directives in the Content Security Policy or absent X-Frame-Options headers. Clickjacking can result in unauthorized state changes, data leakage, or transaction fraud, particularly in applications that handle sensitive data or financial interactions. Its identification in a considerable portion of the application indicates that frame embedding restrictions should be strengthened universally to prevent UI redress attacks.

Correlated with the missing CSP and clickjacking concerns, the lack of a robust Content Security Policy means the application fails to specify safe script sources, frame ancestors, and other critical parameters designed to isolate trusted content sources from untrusted ones. This deficiency leaves the site susceptible not only to unauthorized script execution but also to an expanded surface for framing and data exfiltration attacks. A well-constructed CSP also serves as a comprehensive control to enforce other security headers' policies, thus its absence is a serious concern requiring immediate remediation.

The scan also discovered two specific incidences of insecure client-side storage practices, where sensitive data has been improperly stored in local and session storage respectively. While these issues were identified in only one URL each, the implications are significant, as local and session storage mechanisms are inherently insecure for holding sensitive information such as authentication tokens, personally identifiable information (PII), or payment data. Unlike HTTP-only, secure cookies, client-side storage is accessible to all scripts running on the page, thereby exposing sensitive data to cross-site scripting exploits or malicious third-party scripts. This storage compromise escalates the risk of data theft, session hijacking, and unauthorized access to user accounts.

The presence of sensitive data in localStorage or sessionStorage suggests an architectural flaw where secure session management techniques are either misunderstood or bypassed in favor of convenience. This risk is compounded by the lack of CSP enforcement, which would otherwise help constrain script injection attacks. Therefore, mitigating these issues requires re-architecting storage strategies, favoring secure, server-side session management mechanisms or encrypted tokens stored in HTTP-only cookies—thus reducing attack vectors available to client-side script threats.

Overall, although the severity of vulnerabilities detected falls predominantly within low and medium categories, the verticals affected represent foundational security controls whose absence exposes the application and its users to easily avoidable yet impactful risks. The distribution of the detected issues signals a systemic need to enhance security hygiene and adopt industry best practices regarding HTTP header hygiene, security policy enforcement, storage of sensitive data, and framing protection. Addressing these will significantly bolster the defense-in-depth posture and reduce exposure to common web threats.

From a development perspective, the rectifications necessary are multifaceted. Immediate attention must focus on remediating issues that could lead to data compromise and script-based exploitations, namely the absence of a Content Security Policy and the improper handling of sensitive storage data. These represent direct avenues for attack and should be mitigated posthaste.

Other issues, such as enforcing HTTP Strict Transport Security and configuring X-Content-Type-Options headers, although also crucial, can be categorized within a priority fix bracket. These ensure secure transport and proper content handling, further securing the application's trust model, and should be resolved following the urgent vulnerabilities.

Lastly, the low severity issues, including missing headers that confer moderate security benefits and partial clickjacking protections, may be slated for the later fix phase. While they do not represent immediate threats, their remediation contributes positively to the comprehensive security posture and resilience of the application.

Immediate Fix

- Content Security Policy (CSP) not implemented (Medium severity): Instituting a strict CSP is paramount. Developers must define explicit script-src, frame-ancestors, style-src, and other directives to allow only trusted resources to execute. This will mitigate cross-site scripting and clickjacking risks. Crafting a CSP that balances security with application functionality requires detailed testing but is indispensable. Implementing nonce or hash-based policies for inline scripts can further strengthen defense.
- Sensitive data stored in local storage (Medium severity): Application code responsible for storing authentication tokens, personally identifiable information, or any sensitive data should be refactored to stop leveraging localStorage. Transitioning to secure cookies with flags HttpOnly and Secure dramatically reduces attack surface since client-side scripts cannot read these cookies. Encrypting sensitive data server-side rather than client-side must be a standard.
- Sensitive data stored in session storage (Medium severity): Similar to local storage, session storage usage for sensitive information must be eradicated. Instead, establish server-side managed sessions or JWTs stored securely in HTTP-only cookies. Validate that session expiration and renewal are properly handled to prevent session fixation and reuse.

Priority Fix

- Clickjacking: improper restriction of rendered UI layers or frames (Medium severity): Enforce frame-ancestors directives in CSP or implement X-Frame-Options headers (DENY or SAMEORIGIN) across all application pages. This will block unauthorized framing attempts on the application, mitigating clickjacking attacks that deceive users into interacting unintentionally. Rigorous testing is necessary to ensure legitimate embedded use cases are not disrupted.
- Strict Transport Security not enforced (Low severity): Enable HSTS with an appropriate max-age value across all web assets to compel browsers to use only HTTPS, eliminating plaintext HTTP downgrade vulnerabilities. Also consider including subdomains with the includeSubDomains directive where applicable for comprehensive coverage.
- X-Content-Type-Options header missing (Low severity): Add the X-Content-Type-Options header with a value of "nosniff" to all HTTP responses to prevent MIME sniffing by browsers. This limits execution of maliciously served content with incorrect or ambiguous MIME types.

Later Fix

- Strict Transport Security not enforced (Low severity): Though essential, implementation is less urgent than mitigating injection and data storage risks. Plan phased rollout to progressively enforce HSTS policy after ensuring all resources support HTTPS.
- X-Content-Type-Options header missing (Low severity): While improving security posture, lack of this header is less likely to be exploited compared to injection or clickjacking risks, so scheduling for later fixes is acceptable.
- Clickjacking vulnerabilities (Medium severity), given widespread occurrence, could be incrementally remediated but must be a medium-term priority after CSP implementation.

Achieving remediation requires a layered approach encompassing secure development lifecycle enhancements, improved configuration management, and continued security awareness. Developers should integrate automated pipeline scans and leverage security-focused code reviews to detect regressions or newly introduced weaknesses early. Additionally, security frameworks and libraries that facilitate CSP generation, secure cookie management, and standard header configurations should be incorporated into the technology stack.

Beyond fixing the revealed vulnerabilities, this scan underscores the need for an organizational security policy emphasizing the mandatory use of secure headers, avoidance of client-side sensitive data storage, and adherence to secure coding standards to prevent injection flaws. Developing a security checklist that includes verification of CSP presence and strict header application before production releases would help institutionalize these practices.

Training and knowledge-sharing among team members about the importance and mechanics of HTTP headers, CSP capabilities, and safe data storage patterns will enhance capability to prevent similar issues locally. The involvement of security architects and penetration testers prior to deployment accelerates identification and corrective actions.

Given the interconnected nature of these findings, resolving one without the others might yield partial remediation benefits. For instance, fixing data storage issues without implementing CSP leaves scripts free to execute malicious code that could directly access stored secrets. Likewise, applying CSP in isolation does not secure session data improperly held client-side unless cookie storage best practices are adopted. Therefore, remediation must be comprehensive, combining policy enforcement, secure storage, and transport layer protections harmoniously.

To summarize, the DAST scan of <https://www.parihaaram.in> reveals a web application that currently lacks several critical security controls, particularly in enforcing modern security headers, implementing comprehensive content controls, and securely managing client-side storage. Although individual vulnerabilities fall into medium and low severity classes, their cumulative effect presents a meaningful security risk that could lead to data exposure, session hijacking, scripting attacks, and user interaction-based fraud.

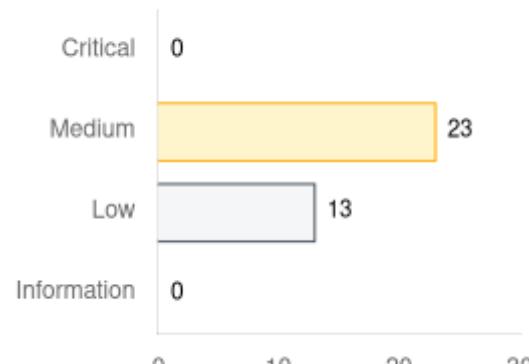
Prioritizing fixes should focus first on the absence of a Content Security Policy and storing sensitive data insecurely in client-side storage, which are the most directly exploitable issues. Next, robust enforcement of clickjacking defenses and strict transport security should be implemented to further harden the security stance. Lastly, the less severe but still relevant missing headers can be configured in subsequent releases, ensuring a continuous improvement lifecycle.

Combining these remediation efforts with organizational process improvements will enhance the maturity of the security posture and better protect both the application's users and its operational integrity against evolving cyber threats.

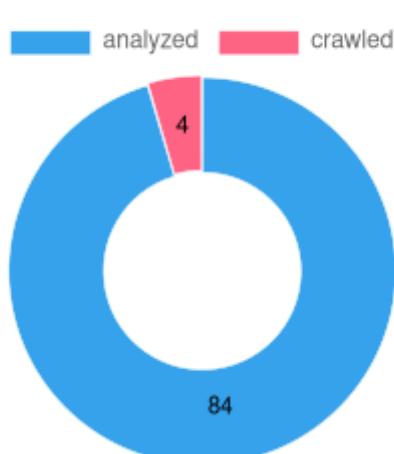
3. Scan Information

URL	https://www.parihaaram.in	
Risk Factor	Medium	
Detected Vulnerabilities	(23) Medium (13) Low	
Compliance Test Result	(4) OWASP Top 10 (3) HIPAA (0) GDPR (1) PCI DSS (3) ISO 27001-A	
SSL Information	Certificate Score:	A+
	Valid Until:	2026-04-03 03:43
	Supported Protocols:	TLS1.2, TLS1.3
Server Ports Scan	(2) Information	
Mail Configuration	(0) Pass (1) Fail	
Javascript Packages	(2) Success	
Server Side Technologies	No details available	
Crawled Information	(4) Crawled Request, (0) Forms, (0) API request	
Test Started On	Wednesday, January 14, 2026 at 05:09 PM UTC	
Time Taken	9.31 Min	
Scan Server	South India	
Scan Scope	Web App Scan	
Frontend	React	
Backend	Express.js	

Detected Vulnerabilities



Detections



3.1 Detected Vulnerabilities

No	Vulnerability	Risk Factor	Count
1	ClickJacking : Improper Restriction of Rendered UI Layers or Frames	Medium	10
2	Content Security Policy Not Implemented	Medium	11
3	Sensitive Data Stored in Local Storage	Medium	1
4	Sensitive Data Stored in Session Storage	Medium	1
5	Strict Transport Security Not Enforced	Low	6
6	X-Content-Type-Options Header Missing	Low	7

3.2 Compliance Test Result

Compliance Test Result Summary

No.	Compliance	Status
1	OWASP Top 10	Fail
2	HIPAA	Fail
3	GDPR	Pass
4	PCI DSS	Fail
5	ISO 27001-A	Fail

3.2.1. OWASP Top 10 Compliance Test Result

This report's findings are limited to a 'black-box' analysis of OWASP Top 10 compliance and may not encompass the full scope of OWASP Top 10 compliance requirements.

3.2.1.1. OWASP Top 10 Compliance rules summary

No.	Title	Status	Count
1	A1 Broken Access Control	Fail	1
2	A4 Insecure Design	Fail	2
3	A5 Security Misconfiguration	Fail	4
4	A7 Identification and Authentication Failure	Fail	1
5	A2 Cryptographic Failures	Pass	-
6	A3 Injection Flaws	Pass	-
7	A6 Vulnerable and Outdated Components	Pass	-
8	A8 Software and Data Integrity Failure	Pass	-
9	A10 Server-Side Request Forgery (SSRF)	Pass	-

3.2.1.2. OWASP Top 10 Compliance Vulnerabilities Information

No	Vulnerability	Risk Factor	Count
1	ClickJacking : Improper Restriction of Rendered UI Layers or Frames	Medium	10
2	Content Security Policy Not Implemented	Medium	11
3	Strict Transport Security Not Enforced	Low	6
4	X-Content-Type-Options Header Missing	Low	7

3.2.2. HIPAA Compliance Test Result

This report's findings are limited to a 'black-box' analysis of HIPAA compliance and may not encompass the full scope of HIPAA compliance requirements.

3.2.2.1. HIPAA Compliance rules summary

No.	Title	Status	Count
1	S.Rule - Part 164, Subpart A, 164.105 Protect Private Health Info	Fail	2
2	S.Rule - Part 164, Subpart C, 164.306(a)(1) Keep Info Safe and Available	Fail	3
3	S.Rule - Part 164, Subpart C, 164.306(a)(2) Protect Against Threats	Fail	2
4	S.Rule - Part 164, Subpart C, 164.306(a)(3) Stop Unauthorized Access	Fail	3
5	S.Rule - Part 164, Subpart C, 164.308(a)(1)(i) Prevent and Fix Problems	Fail	2
6	S.Rule - Part 164, Subpart C, 164.308(a)(1)(ii)(B) Lower Security Risks	Fail	2
7	S.Rule - Part 164, Subpart C, 164.312(e)(1) Protect Info Sent Online	Fail	1
8	S.Rule - Part 164, Subpart C, 164.312(e)(2)(i) Prevent Unauthorized Changes	Pass	-

3.2.2.2. HIPAA Compliance Vulnerabilities Information

No	Vulnerability	Risk Factor	Count
1	ClickJacking : Improper Restriction of Rendered UI Layers or Frames	Medium	10
2	Content Security Policy Not Implemented	Medium	11
3	Strict Transport Security Not Enforced	Low	6

3.2.3. GDPR Compliance Test Result

This report's findings are limited to a 'black-box' analysis of GDPR compliance and may not encompass the full scope of GDPR compliance requirements.

3.2.3.1. GDPR Compliance rules summary

No.	Title	Status	Count
1	A.10.3.2 System acceptance	Pass	-
2	A.11.4.4 Remote Diagnostic and Configuration Port Protection	Pass	-
3	A.11.6.1 Information Access Restriction	Pass	-
4	A.12.2.1 Input Data Validation	Pass	-
5	A.12.2.4 Output Data Validation	Pass	-
6	A.12.3.1 Policy on the Use of Cryptographic Controls	Pass	-
7	A.12.3.2 Key Management	Pass	-
8	A.12.5.4 Information Leakage	Pass	-
9	A.12.5.5 Outsourced Software Development	Pass	-

3.2.3.2. GDPR Compliance Vulnerabilities Information

No	Vulnerability	Risk Factor	Count
No vulnerabilities detected			

3.2.4. PCI DSS Compliance Test Result

This report's findings are limited to a 'black-box' analysis of PCI DSS compliance and may not encompass the full scope of PCI DSS compliance requirements.

3.2.4.1. PCI DSS Compliance rules summary

No.	Title	Status	Count
1	Requirement 4.1 Use Strong Cryptography for Cardholder Data Transmission	Fail	1
2	Requirement 2.2.2 Enable Only Necessary Services and Protocols	Pass	-
3	Requirement 2.2.3 Implement Additional Security Features for Insecure Services	Pass	-
4	Requirement 2.2.4 Configure System Security Parameters	Pass	-
5	Requirement 2.3 Encrypt Non-Console Administrative Access	Pass	-
6	Requirement 6.2 Install Vendor-Supplied Security Patches	Pass	-
7	Requirement 6.5.1 Prevent Injection Flaws in Applications	Pass	-
8	Requirement 6.5.3 Secure Cryptographic Storage	Pass	-
9	Requirement 6.5.4 Secure Communication Channels	Pass	-
10	Requirement 6.5.5 Proper Error Handling Practices	Pass	-
11	Requirement 6.5.7 Prevent Cross-Site Scripting (XSS) Vulnerabilities	Pass	-
12	Requirement 6.5.8 Prevent Improper Access Control Vulnerabilities	Pass	-
13	Requirement 6.5.9 Prevent Cross-Site Request Forgery (CSRF)	Pass	-
14	Requirement 6.5.10 Prevent Broken Authentication and Session Management	Pass	-

3.2.4.2. PCI DSS Compliance Vulnerabilities Information

No.	Vulnerability	Risk Factor	Count
1	Strict Transport Security Not Enforced	Low	6

3.2.5. ISO 27001-A Compliance Test Result

This report's findings are limited to a 'black-box' analysis of ISO 27001-A compliance and may not encompass the full scope of ISO 27001-A compliance requirements.

3.2.5.1. ISO 27001-A Compliance Vulnerabilities Information

No	Vulnerability	Risk Factor	Count
1	ClickJacking : Improper Restriction of Rendered UI Layers or Frames	Medium	10
2	Content Security Policy Not Implemented	Medium	11
3	Strict Transport Security Not Enforced	Low	6

3.3 SSL Certificate Information

IP Address	64.29.17.65
Signature Algorithm	SHA256 with RSA
Key	RSA 2048 bits (exponent is 65537)
Serial Number	060C47E3F5713DB159CC816F538F4B884514
Common Names	www.parihaaram.in
Alternative Names	www.parihaaram.in
Valid From	2026-01-03 03:43
Valid Until	2026-04-03 03:43
Revocation information	http://r12.c.lencr.org/21.crl
OCSP	--
OCSP Must Staple	No
DNS CAA	--
Certificate Transparency	yes (certificate extension)
Issuer	R12 (Let's Encrypt from US)
Certificate Score	A+
Supported Protocols	TLS1.2, TLS1.3

* For detailed information of ssl certificate, please visit:

<https://app.zerorthreat.ai/report/MKE9ZZC71093/ssl-certificate-examination>

3.4 Server Ports Scan

Port Number

- 443
- 80

* Port Severity is based on known vulnerabilities (CVEs) associated with the port.

* For detailed information of server ports, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/server-port-scan>

3.5 Mail Configuration

Name	Info
● Error	gethostbyname() argument 1 must be str, bytes or bytearray, not None

* For detailed information of mail configuration, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/mail-configuration>

3.6 Javascript Packages

Package Name	Version
● Node.js	20
● React-Dom	19.3.0-canary-f93b9fd 4-20251217

* For detailed information of javascript-packages, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/javascript-packages>

4. Technical Details: Findings and Recommendations

4.1. ClickJacking : Improper Restriction of Rendered UI Layers or Frames

Info

Title	Info
Risk Factor	Medium
Likelihood Of Exploit	5
CVSS v3.0 Base Score	5.4
CVSS Vector	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:L
CWE	CWE-1021: Improper Restriction of Rendered UI Layers or Frames
CVE References	CVE-2015-1241 CVE-2016-2496 CVE-2017-0492 CVE-2017-4015 CVE-2017-5697 CVE-2017-7440

Description

Clickjacking is a deceptive technique where malicious elements are hidden beneath benign overlays, tricking users into clicking and unwittingly performing actions that compromise their data and the integrity of websites. This tactic not only facilitates data theft but can also lead to unauthorized transactions and undermine the reputation of affected websites. To safeguard against clickjacking, several preventive measures can be employed. Implementing the X-Frame-Options header with directives like DENY or SAMEORIGIN helps control how web pages are displayed within frames or iframes, thereby mitigating the risk of clickjacking attacks. Additionally, adopting a robust Content Security Policy (CSP) allows web developers to specify trusted sources for content and restrict the execution of scripts, which aids in preventing malicious overlays. Employing frame-busting JavaScript code further enhances protection by ensuring that a web page cannot be embedded within another site's frame without explicit authorization. By implementing these proactive measures and regularly updating security protocols, website administrators can effectively mitigate the risks posed by clickjacking attacks and uphold the security and trustworthiness of their online platforms.

Findings

-  /
-  /dashboard
-  /chat

- ☒ /_next/image
- ☒ /admin
- ☒ /.well-known
- ☒ /.well-known/vercel
- ☒ /.well-known/vercel/security
- ☒ /.well-known/vercel/security/static
- ☒ /_next

* For detailed information of ClickJacking : Improper Restriction of Rendered UI Layers or Frames Vulnerabilities, please visit: <https://app.zerorthreat.ai/report/MKE9ZZC71093/vulnerabilities>

References

[CWE Improper Restriction of Rendered UI Layers or Frames](#)

[MDN Clickjacking](#)

[OWASP Clickjacking](#)

[OWASP Clickjacking Defense Cheat Sheet](#)

[OWASP Testing for Clickjacking](#)

4.2. Content Security Policy Not Implemented

Info

Title	Info
Risk Factor	Medium
Likelihood Of Exploit	5
CVSS v3.0 Base Score	3.1
CVSS Vector	CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N
CWE	CWE-693: Protection Mechanism Failure

Description

Content Security Policy (CSP) serves as a critical defense mechanism for websites, acting as a gatekeeper that regulates the loading of scripts and resources. In the absence of CSP, websites are susceptible to a range of security threats including Cross-Site Scripting (XSS), data injection attacks, and clickjacking. By implementing CSP, web developers can specify which sources are permitted to execute scripts, load resources, and frame the site, thereby mitigating the risk of unauthorized script execution and malicious content injection. CSP configurations enable organizations to enforce strict policies that restrict actions such as inline scripts, eval functions, and external resources from untrusted origins, thereby enhancing the overall security posture of their web applications. Embracing CSP not only bolsters protection against common web vulnerabilities but also fosters user trust by ensuring that only trusted content sources are permitted, thereby safeguarding sensitive user data and preserving the integrity of online interactions.

Findings

- 🔗 /
- 🔗 /.well-known/vercel/security/static/challenge.v2.min.js
- 🔗 /dashboard
- 🔗 /chat
- 🔗 /_next/image
- 🔗 /admin
- 🔗 /.well-known
- 🔗 /.well-known/vercel
- 🔗 /.well-known/vercel/security
- 🔗 /.well-known/vercel/security/static

 / _next

* For detailed information of Content Security Policy Not Implemented Vulnerabilities, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/vulnerabilities>

References

[MDN Content Security Policy \(CSP\)](#)

[OWASP Content Security Policy](#)

[OWASP Content Security Policy Cheat Sheet](#)

[OWASP Testing for Content Security Policy](#)

4.3. Sensitive Data Stored in Local Storage

Info

Title	Info
Risk Factor	Medium
Likelihood Of Exploit	5
CVSS v3.0 Base Score	4.6
CVSS Vector	CVSS:3.1/AV:L/AC:H/PR:N/UI:R/S:U/C:H/I:N/A:N
CWE	CWE-922: Insecure Storage of Sensitive Information

Description

Storing sensitive data such as authentication tokens, passwords, user credentials, personal information, or session details in browser Local Storage poses significant security risks. Local storage is persistently accessible through JavaScript, and data stored there remains available even after browser sessions end or tabs close. If an attacker successfully exploits vulnerabilities like Cross-Site Scripting (XSS), they can retrieve sensitive data directly from local storage, resulting in user impersonation, unauthorized access, identity theft, or compromise of sensitive information. Such vulnerabilities are often exacerbated by inadequate client-side data handling practices. Organizations must avoid storing sensitive or critical data in client-side storage. Recommended mitigation strategies include leveraging secure HTTP-only cookies for authentication purposes, using secure server-side session management, and applying strong Content Security Policies (CSP) to mitigate script-based attacks. Additionally, conducting regular code audits, implementing automated client-side storage analysis tools, and ensuring developer education on secure data handling practices are essential.

Findings

* For detailed information of Sensitive Data Stored in Local Storage Vulnerabilities, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/vulnerabilities>

References

[OWASP Session Management Cheat Sheet](#)

[OWASP Testing Browser Storage](#)

4.4. Sensitive Data Stored in Session Storage

Info

Title	Info
Risk Factor	Medium
Likelihood Of Exploit	5
CVSS v3.0 Base Score	4.6
CVSS Vector	CVSS:3.1/AV:L/AC:H/PR:N/UI:R/S:U/C:H/I:N/A:N
CWE	CWE-922: Insecure Storage of Sensitive Information

Description

The practice of storing sensitive data such as session identifiers, authentication tokens, user credentials, or other personal data in browser Session Storage introduces security risks. Although session storage data is cleared when a browser tab or window is closed, it remains accessible by JavaScript during an active session. Consequently, if an attacker leverages vulnerabilities such as Cross-Site Scripting (XSS), they can easily access and extract sensitive data stored in session storage, potentially enabling unauthorized access, identity theft, session hijacking, or data exposure. Due to the client-side nature of session storage, any sensitive data placed therein significantly increases the attack surface. Security best practices strongly advise against storing sensitive data in client-side storage mechanisms. Instead, applications should use secure HTTP-only cookies for session management and maintain sensitive session state securely on the server-side. Organizations should regularly perform client-side security reviews, employ strict Content Security Policies (CSP) to mitigate script-based attacks, and conduct developer training to ensure secure data handling practices are consistently followed.

Findings

* For detailed information of Sensitive Data Stored in Session Storage Vulnerabilities, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/vulnerabilities>

References

[OWASP Session Management Cheat Sheet](#)

[OWASP Testing Browser Storage](#)

4.5. Strict Transport Security Not Enforced

Info

Title	Info
Risk Factor	Low
Likelihood Of Exploit	5
CVSS v3.0 Base Score	6.5
CVSS Vector	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N
CWE	CWE-319: Cleartext Transmission of Sensitive Information
CVE References	CVE-2002-1949 CVE-2004-1852 CVE-2005-3140 CVE-2007-4786 CVE-2007-5626 CVE-2008-0374 CVE-2008-3289 CVE-2008-4122 CVE-2008-4390 CVE-2022-29519 CVE-2022-30312 CVE-2022-31204

Description

Websites that lack Strict Transport Security (HSTS) are susceptible to eavesdropping and data theft, as they do not enforce secure HTTPS connections. HSTS mandates HTTPS usage, effectively mitigating these risks. To enhance your website's security, enable HSTS with a sufficiently long max-age directive to ensure browsers continue enforcing HTTPS for extended periods. Including subdomains in your HSTS policy extends protection across all related domains, while considering preloading ensures browsers automatically enforce HSTS before initial contact, further bolstering security. Implementing HSTS not only strengthens your website's defenses against various cyber threats but also enhances user trust by safeguarding sensitive data and promoting a secure browsing experience.

Findings

- 🔗 /admin
- 🔗 /.well-known
- 🔗 /.well-known/vercel
- 🔗 /.well-known/vercel/security
- 🔗 /.well-known/vercel/security/static
- 🔗 /_next

* For detailed information of Strict Transport Security Not Enforced Vulnerabilities, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/vulnerabilities>

References

[MDN Strict-Transport-Security](#)

[OWASP HTTP Strict Transport Security Cheat Sheet](#)

[OWASP Secure Headers Project](#)

[OWASP Test HTTP Strict Transport Security](#)

4.6. X-Content-Type-Options Header Missing

Info

Title	Info
Risk Factor	Low
Likelihood Of Exploit	5
CVSS v3.0 Base Score	3.1
CVSS Vector	CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N
CWE	CWE-693: Protection Mechanism Failure

Description

The absence of the X-Content-Type-Options: nosniff header exposes websites to vulnerabilities where browsers may incorrectly interpret file types based on content, rather than relying on the server-provided MIME types. This discrepancy can be exploited by attackers to launch Cross-Site Scripting (XSS) attacks, disguising malicious scripts as legitimate files and deceiving users or compromising data integrity. By including the nosniff directive in the header, web developers instruct browsers not to infer the MIME type but to strictly adhere to the server-provided content type, thereby mitigating the risk of XSS attacks and other forms of data injection. This proactive security measure enhances the overall protection of web applications, ensuring that content is interpreted accurately and minimizing the potential for malicious exploitation through browser misinterpretation. Incorporating robust security headers like X-Content-Type-Options:nosniff is essential for fortifying web defenses and maintaining the integrity and trustworthiness of online platforms.

Findings

- 🔗 / .well-known/vercel/security/static/challenge.v2.min.js
- 🔗 / admin
- 🔗 / .well-known
- 🔗 / .well-known/vercel
- 🔗 / .well-known/vercel/security
- 🔗 / .well-known/vercel/security/static
- 🔗 / _next

* For detailed information of X-Content-Type-Options Header Missing Vulnerabilities, please visit:
<https://app.zerorthreat.ai/report/MKE9ZZC71093/vulnerabilities>

References

[MDN X-Content-Type-Options](#)

[OWASP HTTP Security Response Headers Cheat Sheet](#)

[OWASP Secure Headers Project](#)