iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VII **Month of publication:** July 2026

DOI: https://doi.org/10.22214/ijraset.2026.84363

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VII July 2026- Available at www.ijraset.com

VulnScanner Pro: A Rule-Based Automated Web Application Vulnerability Assessment and Risk-Scoring System

Athili Laxmi Lavanya[1], Mycharla Madhavkumar[2], Doma Gangadhar[3]
[1, 2]*Student, Master of Computer Applications (MCA), AQJ Center for PG Studies (affiliated to Andhra University), Visakhapatnam, Andhra Pradesh, India*
[3]*Associate Professor, Department of MCA, AQJ Center for PG Studies (affiliated to Andhra University), Visakhapatnam, Andhra Pradesh, India*

Abstract: ***Manual penetration testing needs specialist skills that most student developers and small teams simply don't have time to build, and the commercial DAST products that automate this work — Burp Suite Professional, Acunetix, Nessus — are priced for enterprise budgets rather than a college project or a solo developer's side application. Mature open-source alternatives such as OWASP ZAP close the cost gap but not the usability one: getting useful output still means learning how to configure a scan policy, spider a target, and read through a long list of raw findings. This paper describes VulnScanner Pro, a self-hosted assessment tool that tries to sit in the gap between those two extremes. It runs eight checks against a target URL — SQL injection, cross-site scripting, HTTP security-header analysis, SSL/TLS configuration, CSRF protection, WHOIS, IP geolocation, and Nmap-based port scanning — behind a FastAPI backend and a Next.js dashboard, storing every scan in SQLite so past results can be revisited later. A rule-based scoring function turns the combined module output into one 0–100 risk number and a CRITICAL/HIGH/MEDIUM/LOW label, and a matching recommendation engine looks up remediation text and reference links for whatever was found Active scanning of a system that you don't own can be a legal problem as much as a technical one. So the API by default won't start a scan of a system unless you explicitly pass in a consent flag and some text justification for why you are scanning the system. In a case study on a web application owned by the second author, the authors ran the full eight-module automated security assessment. It took about a minute to run the assessment and it reported a risk score of 20 out of 100 or LOW. The six checks for missing HTTP security headers were the only issues found. The checks for SSL/TLS, SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) all reported that they found no issues. Only the two expected web ports (80 and 443) were open on the web application. The risk breakdown for the web application is shown below along with four recommendations for fixing the six missing HTTP security headers. Each of the recommendations are linked to authoritative external references for additional information on how to fix each of the issues found..***

Keywords: Web Application Security, Vulnerability Scanning, OWASP Top 10, SQL Injection, Cross-Site Scripting, CSRF, Risk Scoring, FastAPI, Automated Security Assessment.

I. INTRODUCTION

When you release a web application fast, security review is something you typically cut first. A code review or even better a penetration test is still the best way to find bugs. But that needs a security expert and as a student, low budget open source developer or even small team, you can't always rely on that kind of expertise. There are also many DAST solutions on the market such as Burp Suite Professional, Acunetix or Nessus for example, but they are typically very expensive for a non-enterprise budget. There are also already quite mature Open Source DAST tools available such as the OWASP ZAP for example. These tools can be used for free and are even very comprehensive, but scan configuration, spidering, raw output and so forth is all targeted towards already experienced security testers. VulnScanner Pro is somewhere in the middle of these two scenarios. On one hand, a full DAST platform like Burp Suite Professional, Acunetix or Nessus is usually to expensive for a small project or even a single developer. On the other hand, mature open-source alternatives like OWASP ZAP are quite powerful, but their scan configuration, spidering, etc. are aimed at security-testing experts. As a developer, you just want to point the scanner at a single URL and get a few minutes later a risk score and a human-readable list of found vulnerabilities. For this purpose, VulnScanner Pro is the perfect tool as a first-pass triage scanner..

A. *Problem Definition*

The specific gaps motivating this work are: the absence of a low-cost, self-hosted scanner that covers the most common OWASP-aligned vulnerability classes in a single pass; the tendency of raw scanner output (logs, request/response dumps) to bury the handful of findings that actually matter; the lack, in many lightweight scanning scripts, of any enforced check that the operator is authorized to test the target; and the difficulty of getting a quick, prioritized signal — as opposed to a exhaustive but unranked report — when time is limited.

B. *Objectives*

1) Implement eight assessment modules — SQL injection, XSS, HTTP security headers, SSL/TLS configuration, CSRF protection, WHOIS, IP geolocation, and Nmap-based port scanning — covering vulnerability classes that recur across the OWASP Top 10 [1], [2].
2) Design a deterministic, rule-based weighted scoring engine that aggregates module output into a single 0–100 risk score and a CRITICAL/HIGH/MEDIUM/LOW severity band.
3) Enforce an explicit informed-consent flag at the API layer, so that a scan cannot be initiated without the operator affirming authorization to test the target.
4) Decouple scan execution from the HTTP request lifecycle using FastAPI background tasks, so that a multi-minute assessment does not block the client connection.
5) Persist every scan and its full result set in SQLite, with a history endpoint for retrieving past assessments.
6) Present results through a Next.js dashboard with per-category score breakdowns and human-readable findings, exportable as a standalone PDF report.

II. LITERATURE REVIEW

The OWASP Top 10 is the most widely referenced classification of web application security risks and is updated periodically through analysis of vulnerability data contributed by testing organizations [1]. In the 2021 revision, broken access control, cryptographic failures, and injection occupy the top three positions, with injection alone tested for in ninety-four percent of contributed applications and an average incidence rate across the underlying weakness categories of roughly three percent [1]. Security misconfiguration — the category that encompasses missing HTTP security headers and permissive TLS settings — was present in approximately ninety percent of tested applications [1]. These figures motivate why VulnScanner Pro treats header analysis and TLS configuration as first-class, separately weighted checks rather than folding them into a single generic "misconfiguration" flag.

Academic evaluation of automated scanners has a long history. Fonseca, Vieira, and Madeira benchmarked commercial web vulnerability scanners against SQL injection and XSS by injecting realistic software faults into test applications and measuring each tool's detection and false-positive rates, concluding that scanner effectiveness varies considerably by vulnerability class and cannot be assumed uniform across tools [7].

On the tooling side, OWASP ZAP (now maintained as ZAP by Checkmarx) remains the most widely used open-source dynamic scanner, combining an intercepting proxy, a crawler/spider, and both passive and active scan engines, and is explicitly documented as a tool for authorized use only [3]. VulnScanner Pro adopts the same consent principle at the architectural level, rejecting any scan request that does not carry an explicit consent flag, rather than relying on a documentation-level warning.

On the server side, REST remains the dominant architectural style for exposing scan and history endpoints, as formalized in Fielding's dissertation [4], and FastAPI was selected for its native async support and background-task primitive, which allows a scan to be queued and executed after the initiating HTTP request has already returned [5]. Nmap remains the reference tool for host and port discovery and is used here, with a raw-socket connect scan as a fallback path when the Nmap binary is unavailable on the host [6].

III. SYSTEM ARCHITECTURE AND METHODOLOGY

VulnScanner Pro follows a three-tier design. The presentation tier is a Next.js single-page dashboard through which the operator submits a target URL, sets an explicit consent flag and consent text, and optionally enables WHOIS and Nmap checks. The application tier is a FastAPI server exposing endpoints to create a scan, poll its status, retrieve a specific scan by ID, list scan history, and delete a scan record. The data tier is a SQLite database that stores each scan's target, consent metadata, status, and full JSON result payload.

On receiving a scan request, the API validates that the consent flag is set and that the target URL is well-formed, inserts a queued scan record, and immediately returns a scan identifier while the assessment itself is executed as a FastAPI background task. This separation means the client is not held open for the two-to-five-minute duration of a full scan and can instead poll the scan-status endpoint.

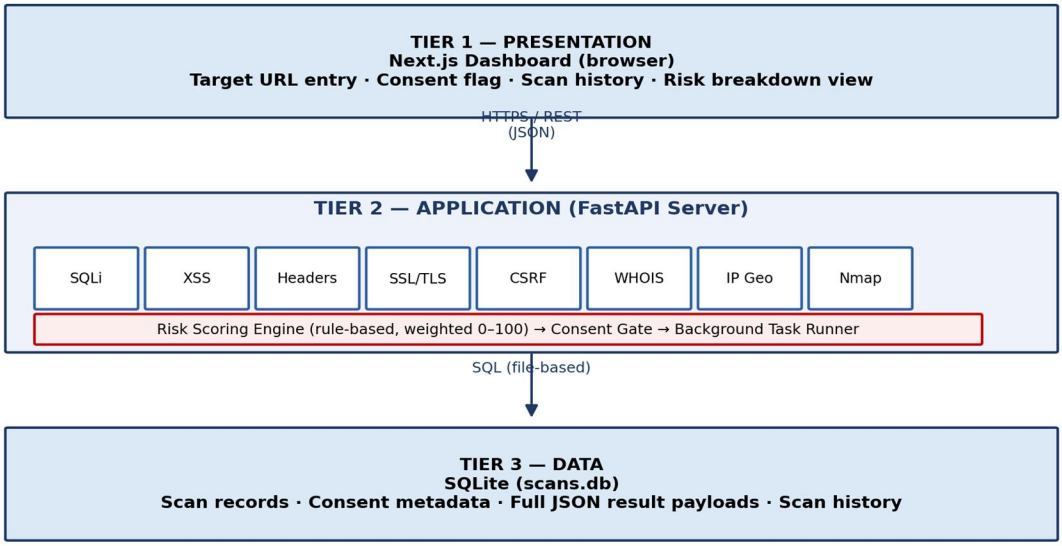


Fig. 1. Three-tier architecture of VulnScanner Pro: Next.js presentation tier, FastAPI application tier (eight assessment modules, risk-scoring engine, consent gate), and SQLite data tier.

A. Assessment Modules

Six of the eight modules — SQL injection, XSS, HTTP headers, SSL/TLS, CSRF, and IP geolocation — always run; WHOIS and the Nmap port scan are optional, operator-selected checks. Each module returns a common result structure containing a status (ok, warning, critical, or error), a list of human-readable findings, and a details object used for the dashboard's per-module view. Table I summarizes each module's technique and its contribution to the overall risk score.

Module	Technique	Max Risk Weight
SQL Injection	Reflected error-pattern matching against 7 payload variants across up to 5 query parameters	30
Cross-Site Scripting (XSS)	Reflected-payload detection across 5 XSS vectors; checks for Content-Security-Policy mitigation	25
HTTP Security Headers	Presence check for 7 standard headers (HSTS, CSP, X-Frame-Options, etc.) and detection of information-leaking headers	20
SSL/TLS Configuration	Certificate expiry, self-signed detection, cipher/TLS version inspection via a direct TLS handshake	15
CSRF	HTML form parsing for anti-CSRF tokens, SameSite cookie attribute, and X-Frame-Options	15
Open Port Scan	Nmap-based scan of 17 common ports, flagging services such as MySQL, Redis, MongoDB, RDP and Telnet as high-risk if exposed	10
WHOIS Lookup	Domain registrar, creation/expiry date and name-server retrieval (supplementary, not scored)	—
IP Geolocation	Resolves hosting IP, ISP, city/country and timezone via a third-party geolocation API (supplementary, not scored)	—

TABLE I. ASSESSMENT MODULES AND SCORING WEIGHTS

The SQL injection and XSS modules first attempt to substitute all of the URL's query parameters for a set of default names (e.g. id, q, search) if they are not already present. Then each parameter is in turn replaced with all of the attack strings and the resulting HTTP responses are inspected for certain features. The SQL injection module searches for a set of database engine error messages in the response body (for MySQL, PostgreSQL, SQLite, and Oracle). The XSS module looks for the injected code in the page's response body (reflected) as well as for a a number of indicators that the code would be unable to function, such as the presence of a Content-Security-Policy header.

The header check module can find 7 headers that are currently recommended by security web pages, such as the Strict-Transport-Security header, the Content-Security-Policy header, the X-Frame-Options header, the X-Content-Type-Options header, the Referrer-Policy header, the Permissions-Policy header and the X-XSS-Protection header. The module can also find other HTTP headers which may leak server technology details such as Server and X-Powered-By headers. The SSL/TLS check module is able to open a direct TLS connection to a given host, it then can check when the current SSL/TLS certificate will expire and it also checks the issuer and subject of the current SSL/TLS certificate for self-signed certificates. Additionally, the module is able to check the negotiated cipher and TLS protocol version of the current SSL/TLS connection. The CSRF check module can check all the HTML forms on a page for common anti-CSRF token field names, it also can check if the current website is using the SameSite attribute on the cookies and it also checks for the presence of a X-Frame-Options header on the current page which can also be used as a clickjacking control.

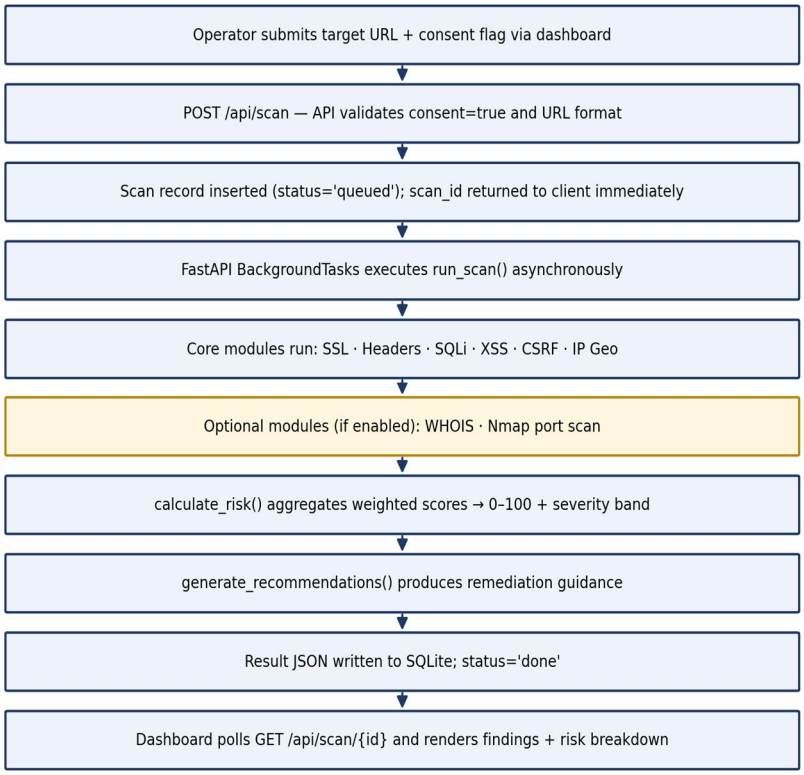


Fig. 2. End-to-end scan execution flow, from consent-gated submission through asynchronous module execution to risk scoring and dashboard display.

B. Risk Scoring Engine

Module findings are combined by a deterministic, rule-based scoring function rather than a trained machine-learning model. Each of six scoring categories carries a fixed maximum contribution — SQL injection 30 points, XSS 25 points, CSRF 15 points, missing headers up to 20 points, SSL/TLS issues up to 15 points, and risky open ports up to 10 points — for a maximum possible score of 100. Within the header category, each missing recommended header contributes 4 points up to the 20-point cap; within the port category, each open port on a predefined risky-service list (including database and remote-administration ports) contributes 5 points up to the 10-point cap. The aggregate score is computed as:

$$Risk = min(100, \Sigma\ score_i),\ \ i \in \{SQLi, XSS, CSRF, Headers, SSL, Ports\} \quad (1)$$

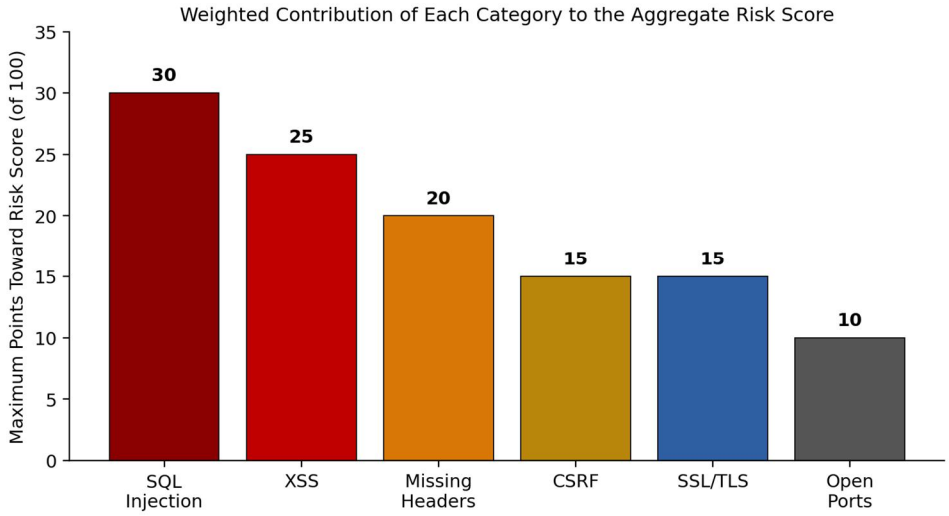


Fig. 3. Maximum point contribution of each scoring category toward the 0–100 aggregate risk score.

The resulting score is mapped to a severity band: CRITICAL at 75 or above, HIGH from 50 to 74, MEDIUM from 25 to 49, and LOW below 25. Because every rule and weight is fixed in code rather than learned from data, the scoring behavior is fully deterministic and auditable — a property that matters for a tool whose output may be used to justify remediation priorities.

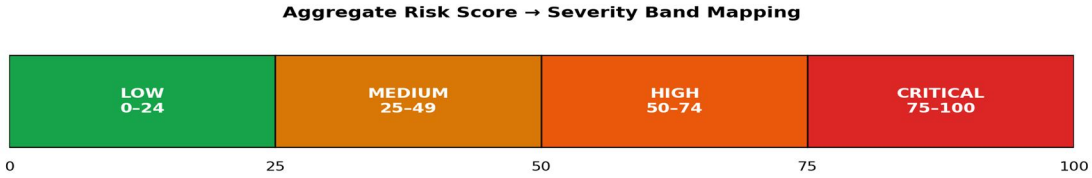


Fig. 4. Mapping of the aggregate risk score to the four severity bands used throughout the dashboard and reports.

C. Consent Enforcement

Unauthorized active scanning of a system — including SQL injection and XSS payload submission and port scanning — can constitute a computer-misuse offence in many jurisdictions, which is why tools such as OWASP ZAP explicitly restrict their use to authorized targets [3]. VulnScanner Pro enforces this at the API boundary: the scan-creation endpoint rejects any request whose consent flag is not set, and persists the operator-supplied consent text alongside the scan record for auditability. This does not substitute for a legal authorization process, but it removes the possibility of an operator triggering an active scan by accident through the dashboard.

D. Recommendation Generation

Alongside the numeric score, the dashboard surfaces a short list of remediation items labelled "AI Recommendations" in the interface. Despite the label, the underlying implementation is a deterministic lookup rather than a trained model: a fixed dictionary maps each missing header, each vulnerable module, and out-of-range SSL conditions to a title, a priority level, a one-line fix, and a reference URL (drawn mainly from MDN and OWASP documentation). This keeps the advice predictable and easy to verify, at the cost of not being able to generalize to findings the dictionary doesn't already cover — a fair trade-off for a tool whose main job is to point a developer at the right documentation quickly, rather than to reason about novel vulnerability classes.

E. Report Export

Every completed scan can be exported as a standalone PDF from the dashboard's "Download PDF Report" control, generated client-side rather than round-tripped through the server. The exported report mirrors the dashboard exactly — scan ID, start and finish timestamps, the operator's identity, the numeric risk score, every module's status and findings, and the generated recommendation list — with a fixed footer disclaimer ("Scan responsibly") reinforcing the authorized-use expectation set at scan creation.

Because the export is generated on demand rather than at scan completion, its own timestamp can trail the scan's finish time by minutes or hours, which is expected: it reflects when the report was pulled, not when the assessment ran.

IV. RESULTS AND DISCUSSION

VulnScanner Pro was evaluated in a case-study run against https://madhavkumar.qzz.io/, a web application owned and operated by the second author (Mycharla Madhavkumar) and hosted on Netlify. This is a single-target validation, not a large-scale benchmark, so the numbers below should be read as evidence that the pipeline works correctly end to end and returns a usable, prioritized result — not as a general statement about detection accuracy across arbitrary applications.

With all eight modules enabled and a working Nmap installation on the host, the scan (ID ad5744a5-db40-4400-8f31-dd91e162a63a) was started at 03:33 PM and marked done at 03:34 PM on 18 July 2026 — under two minutes for the complete assessment, including the two optional modules. Fig. 5 shows the scan-submission screen, where the target URL, consent checkbox, and optional WHOIS/Nmap toggles are set before a scan can begin.

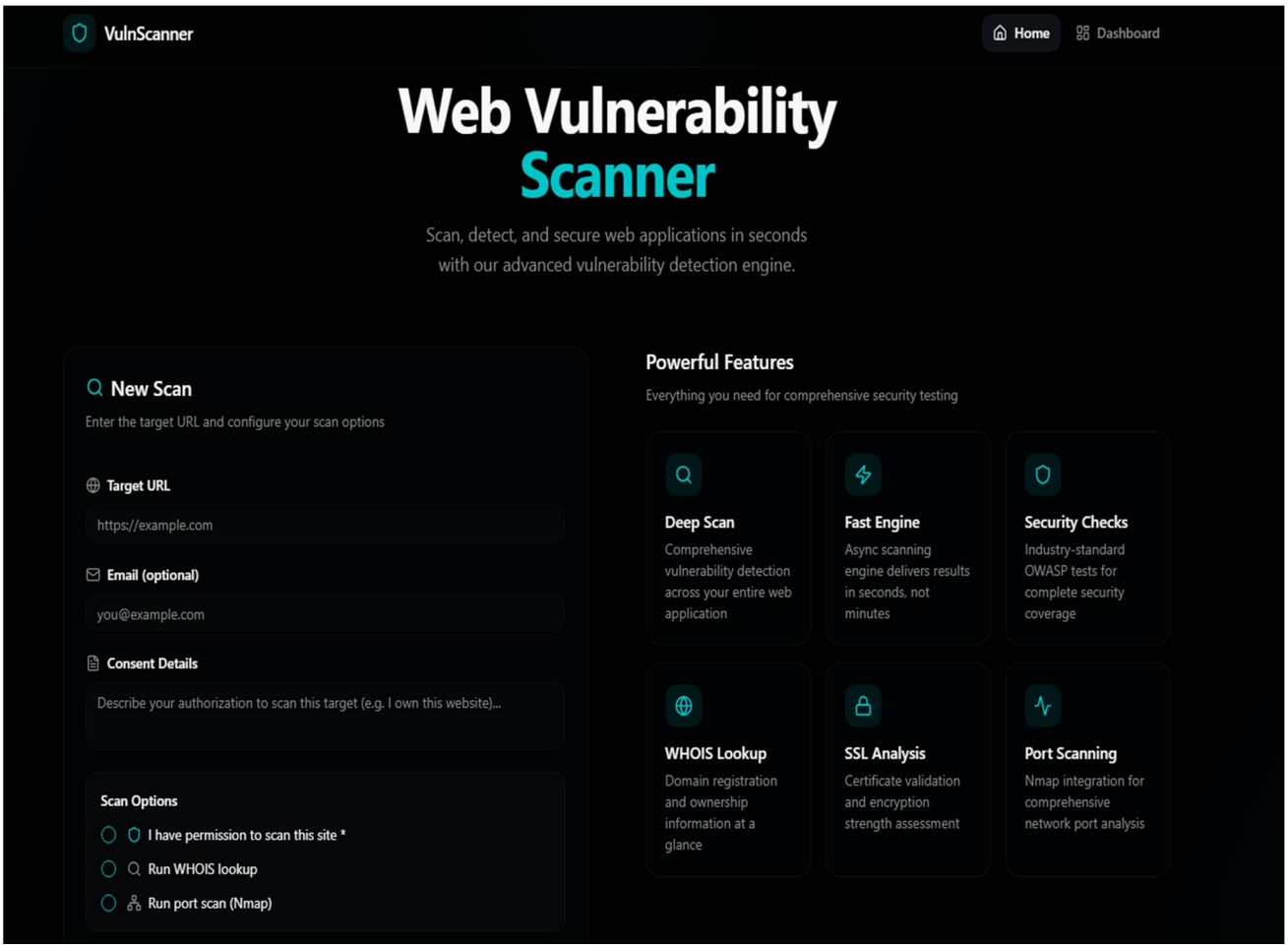


Fig. 5. VulnScanner Pro landing page and scan-submission form, showing the required consent checkbox and optional module toggles.

Table II summarizes the module-by-module outcome. The aggregate score came out to 20 out of 100, placing the target in the LOW band, and the breakdown makes clear that this was driven almost entirely by one category: HTTP headers scored the full 20 points because six of the seven recommended headers were absent, while every other module scored zero. SQL injection, XSS, and CSRF all returned clean on the tested parameters and forms. The SSL/TLS module found a valid Let's Encrypt certificate with 49 days left before renewal and a TLS 1.3 connection, so it contributed nothing to the score. Fig. 6 shows the results dashboard, with the circular risk gauge, the per-category bar breakdown, and the expanded SSL and header findings.

Module	Score	Result
SQL Injection	0 / 30	No error-pattern match on any of 4 tested parameters (id, q, search, page)
XSS	0 / 25	No reflected payload on any of 4 tested parameters (q, search, s, query)
CSRF	0 / 15	No HTML forms present on the page — nothing for the module to flag as unprotected
HTTP Headers	20 / 20	6 of 7 recommended headers missing; header score 1/7
SSL/TLS	0 / 15	Valid Let's Encrypt certificate, TLS 1.3, 49 days remaining
Open Ports	0 / 10	Only ports 80 and 443 open — neither on the risky-service list
WHOIS (supplementary)	—	Domain registered 4 May 2025 via GANDI SAS; 1,020 days to expiry
IP Geolocation (supplementary)	—	Resolves to 98.84.224.111, Ashburn, VA, US (Amazon.com, Inc.)
Total Risk Score	20 / 100	LOW

TABLE II. CASE-STUDY SCAN RESULTS FOR https://madhavkumar.qzz.io/

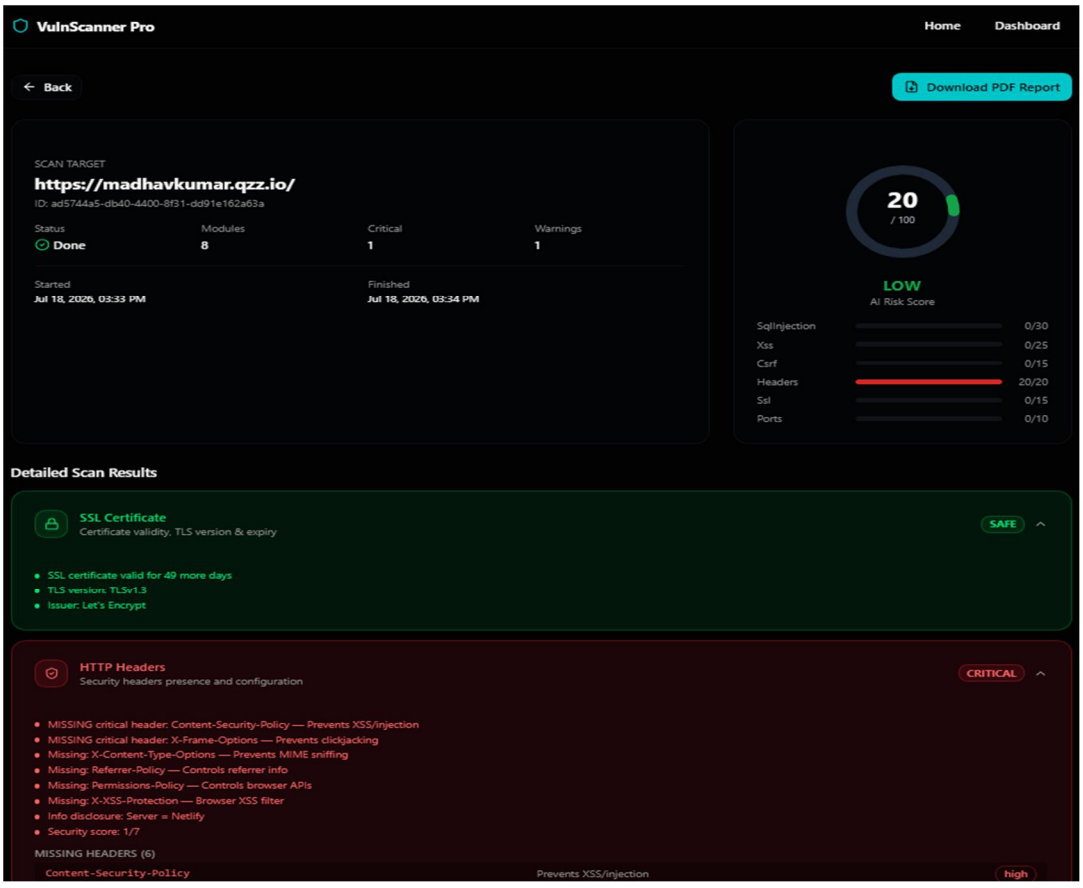


Fig. 6. Scan results dashboard showing the 20/100 LOW risk gauge, per-category score breakdown, and the SSL (safe) and HTTP header (critical) findings.

The header module's findings are the most actionable part of the run: it listed the six missing headers by name (Content-Security-Policy, X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Permissions-Policy and X-XSS-Protection), computed a 1-out-of-7 header security score, and also flagged that the Server response header discloses the underlying hosting platform (Netlify) — a minor information-disclosure finding in its own right. The port scan, run with a genuine Nmap installation rather than the socket fallback, found exactly two open ports — 80 and 443 — both expected for a public web application and neither on the risky-service list, which is why the module's status shows a WARNING badge (open ports were found) while still contributing zero points to the score (neither port is considered risky). Fig. 7 shows this port-scan card together with the four recommendations the system generated from the header findings, each carrying a priority tag and a link to the relevant MDN documentation.

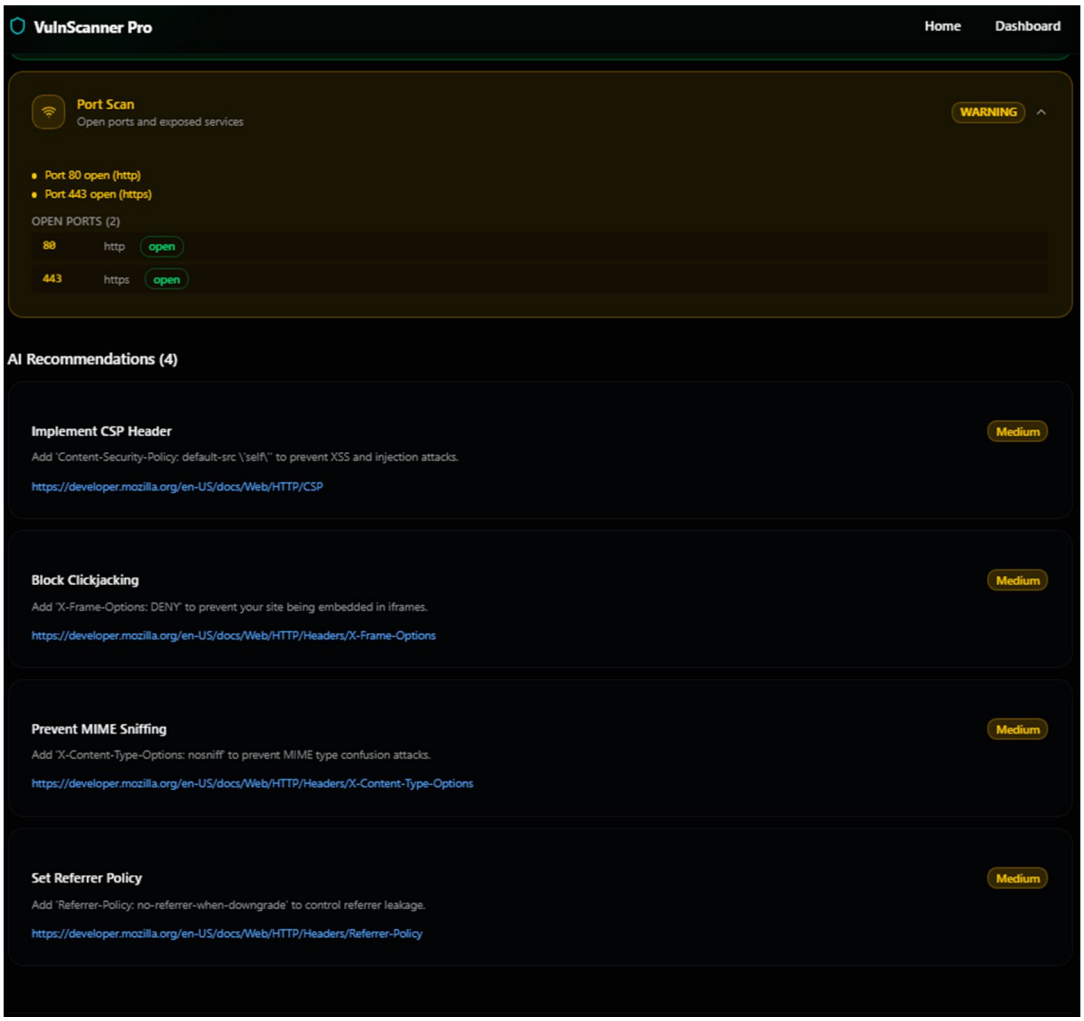


Fig. 7. Open-port findings (ports 80/443, WARNING status but zero risk contribution) and the four rule-based remediation recommendations generated from the missing-header findings.

Two of the modules produced a small but useful cross-check on each other. The header module flagged X-Frame-Options as one of the six missing headers, and the CSRF module independently reported the same header as missing while explaining why it matters for clickjacking protection specifically, even though the page carried no HTML forms for the CSRF module to test directly. Neither module knows about the other's output, so the fact that both converged on the same header from different code paths is a reasonable, if informal, indication that the finding is not a fluke of one particular check.

The two supplementary modules — WHOIS and IP geolocation — don't feed the score, but they filled in a detail worth noting: the Server header on the site's responses names Netlify, while the resolved IP address geolocates to an Amazon.com, Inc. block in Ashburn, Virginia. That is not a contradiction; Netlify's own edge infrastructure runs on AWS in several regions, so both facts can be true of the same request.

It is, however, a good illustration of why VulnScanner Pro treats WHOIS and geolocation output as context rather than as risk signals in their own right — the same underlying fact can be read two different ways depending on which layer of the hosting stack a given check happens to observe. The WHOIS lookup itself returned a domain registered on 4 May 2025 through GANDI SAS with roughly 1,020 days remaining before expiry, comfortably outside the 30-day expiry-warning threshold the module applies.

A few things stood out while reviewing this run. First, the gap between a module's status badge and its score contribution — the port scan shows WARNING but adds 0 points, because "open" and "risky" are deliberately different questions in the scoring logic — is useful in practice but could confuse a first-time user reading the dashboard without the underlying weighting in mind; a short explanatory tooltip would probably help. Second, because only one target was tested and it happened to have no SQL injection or XSS surface, this run says nothing about how well those two modules perform against an application that is actually vulnerable; a broader evaluation, ideally against a deliberately vulnerable benchmark application in addition to real targets, would be needed to characterize detection accuracy the way Fonseca et al. characterized commercial scanners [7]. Third, the reflected, single-request payload approach used for SQL injection and XSS does not attempt time-based blind SQL injection, stored or DOM-based XSS, or multi-step authenticated flows, all of which are common in production applications and represent a real gap against the fuller OWASP Top 10 injection category [1].

V. CONCLUSION

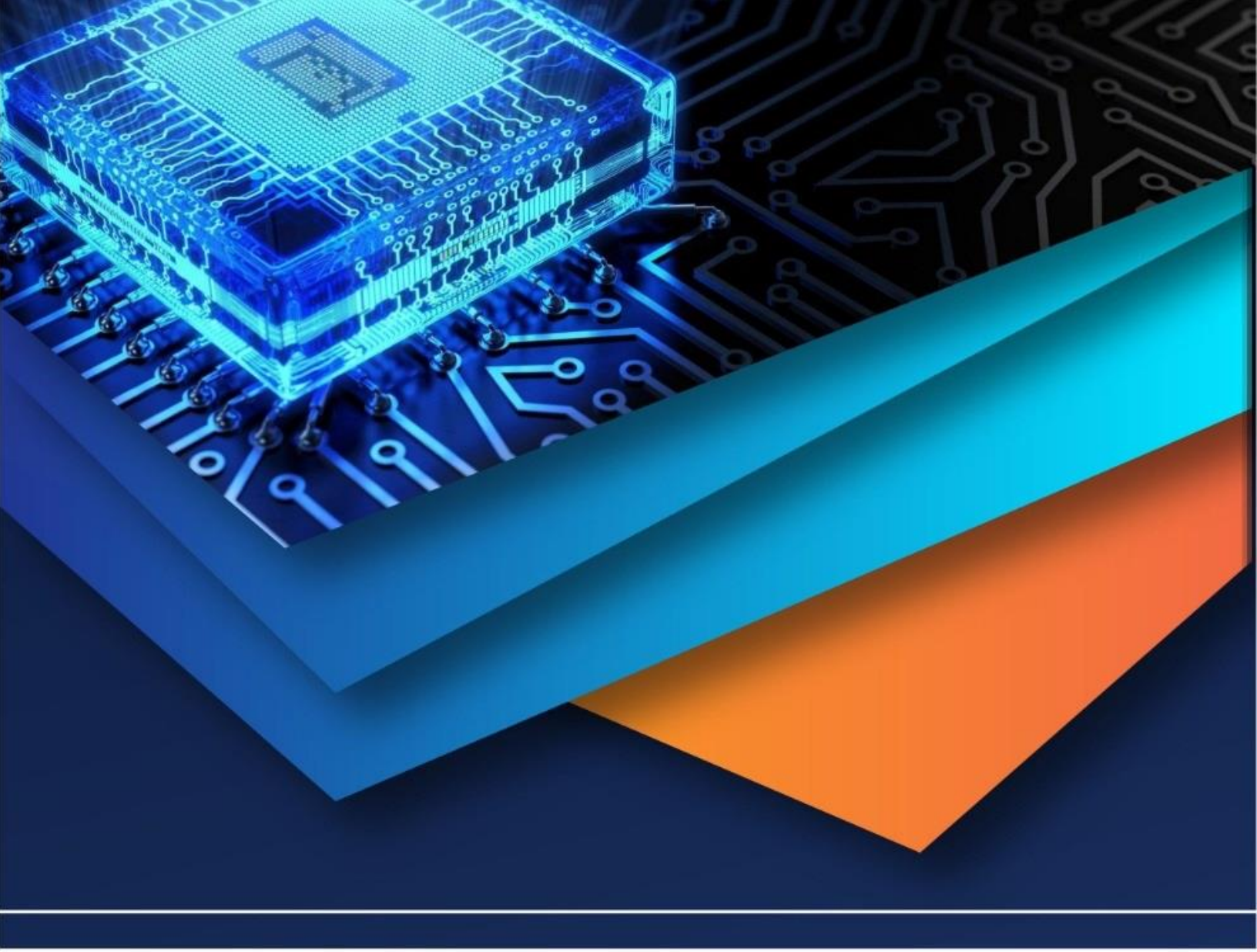
What this project shows, more than anything, is that you don't need a trained model or a heavyweight DAST platform to get a useful first read on a web application's security posture. A handful of well-understood checks — reflected SQL injection and XSS testing, header analysis, TLS inspection, CSRF form checking, and port scanning — combined through a simple weighted-scoring rule, already produce a result that tells an operator where to look first. The consent gate, the background-task execution model, and the SQLite-backed history are what turn that into something a student or a small team could actually run against their own project rather than a one-off script.

VI. FUTURE SCOPE

1) Multi-target and scheduled/recurring scanning, so the tool can track a target's risk score over time rather than as a single snapshot.
2) Broader validation against deliberately vulnerable benchmark applications to measure detection accuracy and false-positive rate, following the methodology used in prior scanner-comparison studies [7].
3) Support for authenticated scanning (session cookies or login flows) so that pages behind a login can be assessed.
4) Detection of stored and DOM-based XSS, and time-based blind SQL injection, in addition to the current reflected-payload checks.
5) CVE/CWE enrichment of findings using a public vulnerability database, to map detected issues to standardized identifiers.
6) PDF/HTML export of scan reports for audit and compliance submission.

REFERENCES

[1] OWASP Foundation, "OWASP Top 10:2021," 2021. [Online]. Available: https://owasp.org/Top10/2021/
[2] OWASP Foundation, "A03:2021 – Injection," OWASP Top Ten, 2021. [Online]. Available: https://owasp.org/Top10/2021/
[3] Zed Attack Proxy (ZAP) by Checkmarx, "ZAP – The World's Most Widely Used Web App Scanner," GitHub Repository, 2024. [Online]. Available: https://github.com/zaproxy/zaproxy
[4] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.
[5] FastAPI, "FastAPI Documentation," 2024. [Online]. Available: https://fastapi.tiangolo.com
[6] G. Lyon, "Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning," Insecure.Com LLC, 2009. [Online]. Available: https://nmap.org/book/
[7] J. Fonseca, M. Vieira, and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," in Proc. 13th IEEE Pacific Rim Int'l Symp. Dependable Computing (PRDC), Melbourne, Australia, 2007, pp. 365–372.
[8] MDN Web Docs, "HTTP Security Headers," Mozilla Developer Network, 2024.
[9] Ministry of Electronics and Information Technology, Government of India, "Digital Personal Data Protection Act, 2023," Gazette of India, 2023.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY