10.48047/jocaaa.2026.35.04.10

ThreatAtlas: A Unified Automated Threat Intelligence and Attack Surface Analysis Platform

Pagalla Bhavani Shankar[1], Dr. M Babu Reddy[2], Derangula Rohith[3*], K. Sudhir [4], D Ravindra[5], CH. Mano Gangadhar[6]

[1]Assistant Professor, Department of Computer Science and Engineering,
University College of Engineering and Technology,
Krishna University, Machilipatnam, Andhra Pradesh, India.
[2]Assistant Professor, Department of Computer Science,
University College of Arts & Science,
Krishna University, Machilipatnam, Andhra Pradesh, India.
[3,4,5,6]Student, Department of Computer Science and Engineering,
University College of Engineering and Technology,
Krishna University, Machilipatnam, Andhra Pradesh, India.

Abstract

In general, a detailed security scan of an accessibly networked target typically involves a tradeoff on a scale of magnitude of single-purpose tools and none of them is aware of the other ones. Their manual process of reconciling their outputs is a drag to the analysis and analysis lags well behind remediation. ThreatAtlas is able to address this gap by combining reconnaissance, vulnerability and threat reputation assessment within a single domain-driven platform. After being given a target domain, the system will also use three scanning engines simultaneously: Wapiti, which crawls the web application to identify vulnerabilities available [1]; a port enumeration module with the support of the Shodan API [5] to transparently switch to a socket-based multithreaded scanner when it cannot access the external service; and the VirusTotal API v3 [4], which also determines reputation and retrieves malware-specific data in the history. All the outputs are entered into a shared back-end store and an AI-based remediation module [2] transforms the raw detection data into ranked actionable fix instructions turning the platform much further into the detection only capability of a conventional scanner. The platform interface is provided through a Vite-based React single-page application [3], which is used to show the progress of live scans, enable temporal comparison of sequential scans to render surface drift, and show a Three.js globe of animated trajectories of threat arcs [6]. This paper provides the architectural reasons, scanning pipeline, component-level implementation and analysis feasible of the existing facilities of the platform and the constraints that constrained them.

Keywords— *threat intelligence: vulnerability scanning: attack surface analysis: port scanning: web application security: React: Three.js: Wapiti: VirusTotal: Shodan*

I. INTRODUCTION

Every person who has ever conducted a real-life security evaluation is familiar with the drill: open the web scanner, wait, export; open the port tool, wait, export; query the threat feed, copy the result, open a spreadsheet and begin the process of reconciliation. The process of moving between tools adds time and a manual correspondence process creates a chance in which one can miss something. Under time pressure, as the teams are almost always, the first thing to lose is the thoroughness of such reconciliation.

ThreatAtlas eradicates the transitions. Instead of posing as a one-purpose scanner, it is an orchestration layer itself, which contains three independent assessment engines web vulnerability analysis with Wapiti [1], network port enumeration with Shodan [5], and malware reputation intelligence with VirusTotal [4] and executes them in parallel against a given domain. What the analyst receives is not three distinct exports but one unified report, enhanced with an AI module [2] that has already read the results and produced ranked remediation actions to each of the issues that it has found.

10.48047/jocaaa.2026.35.04.10

The platform is structurally split in two levels. The scanning, intelligence querying, storing results and analysis based on AI all take place in a Python backend. The frontend is a Vite and React single-page application [3] which offers scan lifecycle management, a dashboard of temporal comparisons to understand the behavior of the attack surface of a target between assessments, and an interactive Three.js globe [6] that visualizes simulated threat paths in three-dimensional form.

The remaining part of this paper is structured in the following way. In section II, the architecture of both tiers is outlined and gives the justification of major structural decisions. Section III takes a tour of the scanning process and how data flows through engine output to consolidated report. The fourth section IV is implementation- technology and the functions of the main components. Sections V- VII appraise the functioning capacities of the platform, call out the shortcomings that are known to it, and suggest the most fruitful future development avenues. The final section of the paper is VIII which concludes.

II. SYSTEM ARCHITECTURE

A. Overall Architecture

The platform takes a classic two-level structure: a back-end that has all the computation, storage and intelligent logic and a front-end that has all the presentation and interaction with the user. They make their communication through the REST HTTP. To keep the scan progress display of the frontend (without the addition of a permanent connection) the backend transmits the HTTP status codes as a lightweight marker of state 202 when a scan is ongoing, 206 when some results have been stored and 200 when the whole assessment is complete. The frontend polling loop reads these codes and the transition of the UI occurs and hence WebSocket complexity is not necessary in this application [7].

At the frontend, there are three React Context providers that share the state responsibility. AuthContext is the owner of JWT-based session management which includes token refresh. ScanContext contains the results and status of the scan that is in progress. LayoutContext will be used to track the visibility of the shell of navigation. Such a break ensures that change in authentication state does not happen by chance and create a re-rendering of scan data, nor layout preferences be bound to either.

B. Backend Architecture

The submission of a domain is followed by the backend resolving the domain name to the network-layer identifier and makes a direct allocation of work to three autonomous scanning modules. A partial or complete failure of one module does not block the rest because all the modules do not share state during runtime nor block on each other, and the report is compiled based on all the results available to it, and the status code is used to reflect the degree of completion.

Wapiti carries out the scanning of the vulnerabilities on the web applications and the backend calls it a child process [1]. These invocation two parameters can be configured, that is, the depth of the crawl and the maximum number of links, the strength of the scan can be modified without altering the integration code itself. Wapiti identifies the vulnerabilities it is experiencing and records the runtime application behavior anomalies; the backend processes that generate and save the structured findings in the database, which may be accessed by the reporting layer and the AI remediation engine.

The enumeration of ports is done in two phases. The backend will query the Shodan API [5] first that might offer known open ports of indexed targets almost at once and without generating any traffic to the target itself. When the Shodan query fails because of any reason, an expired key, network error or an unindexed target, a socket-based fallback: a multi-threaded scanner scanning the standard TCP port set, defined as the codebase variable of COMMON_TCP_PORTS is used instead, and scans over a number of parallel threads. A determined ports are then matched with the entries of the local CVE and a model of cached service-description referred to as PortDescription to find out what software is likely to be available on the individual ports.

Threat intelligence requests are redirected into the VirusTotal API v3 [4]. The backend responds by selecting the endpoint as determined by the kind of entity being evaluated e.g. URL, domain or IP address and retrieving

10.48047/jocaaa.2026.35.04.10

reputation scores, malware family relationships and the history of prior scans. These outputs are inputted directly into the single structure of the findings as well as the vulnerability and port data.

C. Frontend Architecture

The codebase, the frontend code is written in terms of feature based code, not technology based code. There are four independent packages under features directory, which are concerned with authentication, the main dashboard, FindIntel scan interface and scan history. Components directory is common (result cards, data tables, charts and the visualization widgets) and it shall be shown in the top level. The Shell MainLayout and Sidebar and TopNavbar are not feature specific layout directory, but application wide infrastructure and can be found in a layout directory.

When the application is loaded, the HTML shell loads main.jsx that renders the App component. The entire application encases the application within three context provider nesting viz., AuthProvider, LayoutProvider and ScanProvider and the routing is allocated to React Router. MainLayout provides authenticated pages and provides rendering of the sidebar, top navigation and responsive behaviour: at larger viewports the sidebar is collapsed to icons whilst at smaller viewpoints the sidebar is an overlay of the whole screen, triggered by a menu control.

The whole of the HTTP communication is accomplished through the use of Axios and at the time of the building, the environment variable of VITE_API_BASE_URL would be considered by the backend origin. scanDataTransform.js an adapter module is between the network layer and the feature components, and standardizes the raw backend payloads to the internal format of the frontend. This is the implication of this boundary and that when the backend changes format in which it responds only the adapter needs to change and not the rest of the UI.

III. METHODOLOGY

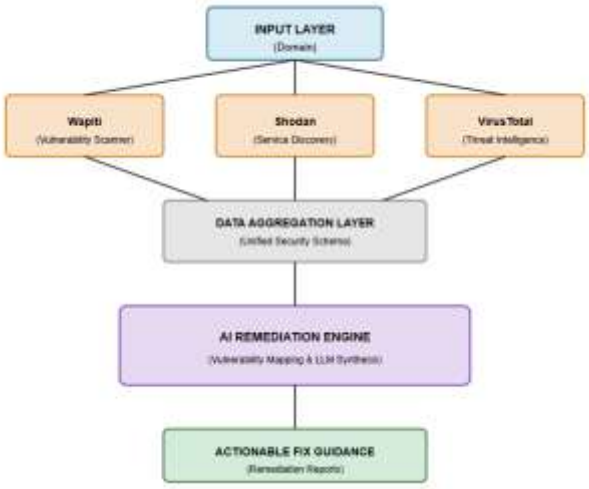

Figure 1: ThreatAtlas Unified Threat Intelligence and Remediation Pipeline

A. Scanning Workflow

The execution of Scan is initiated at FindIntel interface where a user enters a target domain. This request sends an HTTP GET request to /dashboard/FindIntelFullScan/ on the backend, and at that point, the frontend releases control of the process, and start polling. Back-end, the domain is resolved and the three scanning engines are engaged at the same time Wapiti initiates its web crawl, the port module connects with Shodan or pivots to the socket scanner and VirusTotal query requests are made on the resolved entity. The engines write to the common results store when they are done without considering the other ones.

As scan data is added to the database, the AI remediation engine starts to pass. It takes the vulnerability records of Wapiti, the service exposure inferences of the port analysis, and the threat indicators of VirusTotal and generates a remediation entry of each individual finding. Each entry has a similar format: a plain-language description of the detected thing, a risk severity label based on scanner output, a sequence of numbered corrective steps, and a citation to the secure coding or

10.48047/jocaaa.2026.35.04.10

configuration principle that it applies. This organization maintains a consistent remediation output irrespective of the scanner or engine that created the underlying finding [2].

B. Data Processing and Aggregation

All the output of the three scanning engines is stored in one unified system at the backend database. That is the consolidated store whereby the report as well as the AI engine get the source of record. The results centralization approach also provides the scan comparison functionality: when a user calls a delta view the backend fetches the latest scan and a specified previous scan of the same target, calculates a diff that encapsulates port additions or removals, severity changes, and new or resolved subdomains, and the results are sent back to the user in the /dashboard/findintel/compare/ endpoint in three form: current scan, previous scan, and diff

The ScanComparisonDashboard visualizes this payload making the diff the center of interest. Rather than showing two entire reports and letting the analyst identify the difference, it only shows what changed between scans, new ports that were revealed, previously closed vulnerabilities, and subdomain changes and severity changes. A review of this view by an analyst is able to evaluate whether a remediation effort was successful or whether the attack surface has increased since the last engagement, without having to sort through unchanged findings.

C. Frontend–Backend Interaction

The interactions of session management are aimed to three specific endpoints. Identification is done by /auth/signin or /auth/signup; and active session is ensured by regular requests to /auth/refresh access. Each authentication response includes role information and a flag signifying whether the account requires a second authentication factor or not and the frontend uses this information to send the user into the right direction when it comes to signing-in.

A ScanContext.jsx setInterval polling loop is then used immediately after active scan status. The frontend requests the approach to the current scan state every period and changes depending on the code of the HTTP response to choose the state to be shown [7]. TopoJSON files containing geographic boundary data are loaded on demand by the map and globe features which need to access geographic boundary data. The fact that those files are not part of the application bundle implies that the initial load will not be inflated and the geographic data will be updated automatically without necessarily having to update the build.

IV. IMPLEMENTATION

A. Technologies Used

Python is the core of the implementation of the backend. Wapiti is called using the subprocess module and the depth of the scan and the maximum number of links traversed are run time parameters [1]. The multithreaded socket scanner is based on the threading primitives of Python to scan a series of ports in parallel, which otherwise would have required a linear scan, which is reduced by a fraction of the time. VirusTotal communication and Shodan are regular HTTP client libraries [4][5]. The entire data of scan output and service description cache remains in a back end relational database. The AI remediation engine is an in-process operation, whereby it will query the database on scan results, generate structured text and write the results to the same store [2]. It does not come into contact with the target system and other infrastructure at all.

The frontend is based on Vite to bundle and do a hot-module replacement in development. React offers the model and rendering of the components [3]. React Router is used to handle client-side routing. Axios is used as the HTTP client. Three.js is used to render the three-dimensional threat globe [6], which creates a scene with a sphere geometry that represents Earth, a fragment shader that creates the appearance of atmospheric radiance, and country boundary meshes based on the TopoJSON data that is asynchronously loaded on startup. Presentation styling is based on plain CSS structured around a series of personal properties, defined in globals.css and variables.css; each of them has a scoped set of rules in its own co-located stylesheet.

B. Key Modules and Components

FindIntel.jsx is the point of entry of all the scanning in the frontend. It is in charge of the form receiving the target domain, sending the first scan request, running the polling loop, and providing a report about the verdicts of the engines at the end of the scan. Those analysts who require a more detailed perspective open TechnicalDeepDive.jsx, which presents the results

10.48047/jocaaa.2026.35.04.10

in individual tables of open ports, scanned subdomains, metadata of SSL certificates and the entire table of Wapiti-reported vulnerabilities.

ThreatGlobe.jsx is used to create the three dimensional threat visualization of the platform [6]. A THREE.Scene consists of a sphere with texture of the earth map around it and atmospheric glow effect which is created using a custom shader. The boundaries of countries are created using the coordinate information in TopoJSON that is projected to the sphere surface. Threat arcs, which are Botnet, DDoS, and Malware event types are displayed as QuadraticBezierCurve3 geometries, which curve between the source and target city coordinates. All arc instances are controlled by a four phase life cycle, including enter, where the arc fades in; travel, where it animates along the curve; exit, where it fades out; and destroy, where it is removed off the scene to release memory.

The three-object comparison payload on the /dashboard/findintel/compare/ endpoint is sent to ScanComparisonDashboard.jsx and displayed as the main UI surface, foregrounding severity changes, port changes and subdomain movement instead of displaying both scan reports in their entirety. Protected Route ProtectedRoute.jsx secures all the authenticated routes: it renders the route in a loading state until AuthContext can resolve the current session, after which it renders the route or redirects an unauthenticated request to /login. The combination of the MainLayout.jsx and Sidebar.jsx creates the chrome of the application, which is the bottom line navigation, the top bar, the content area and controls the switching between desktop and mobile layout modes.

Authentication HTTP calls are done using authApi.js which is an instance of Axios with the base URL, default headers and token injection and 401 refresh interceptors pre-configured. The calls on scans do not take advantage of this instance presently; they are built as bare Axios requests with options manually set, and thus not resolved by the interceptor chain. The scanDataTransform.js adapter translates the structure of the scan response of the backend into the standard internal schema on which feature components use, with any API-level modifications to that particular module.

V. DISCUSSION

ThreatAtlas is fundamentally a positional instead of technological value: it lies at the cross over of a number of assessment fields that are typically divided on the lines of tools. Each of the three types of web vulnerability information [1], open port intelligence [5], and threat reputation scores [4] provides a partial view of the risk of an internet-facing target. The difference between those three streams of data appearing in the same database, being processed by the same reporting layer, is that the gaps between them become noticeable. A function found within a non-standard port by the socket scanner may be cross-linked with VirusTotal indicators and Wapiti application-layer results in a single view, without the analyst having to develop that connection.

The AI remediation aspect changes the output of the platform towards prescriptive as opposed to observational [2]. A scanner which informs a team that there is an SQL injection vulnerability on a particular endpoint has done something useful; a system which also informs that user inputs are type-checked and that users should use parameterized queries and suggests the use of least-privilege database permissions has done more. Regardless of whether or not a practitioner concurs with each of the generated suggestions, having a structured starting point on remediation is quicker than constructing one ad hoc, especially with respect to findings that do not belong to the major area of expertise of one of the team members.

The resiliency of port scanning should be given particular consideration since external API dependency is one of the weak areas of such platforms. Shodan has the strength of well-indexed public targets, but cannot be used in locked-down environments, might not cover newly-provisioned infrastructure, and has authorization requirements that cannot always be met by any deployment [5]. The fact that ThreatAtlas automatically switches to a local socket scanner when Shodan is not reachable causes ThreatAtlas to slow gracefully instead of shutting down, and the port assessment is still successful at the expense of a bit of performance instead of an empty result.

The ThreatGlobe visualization and ScanComparisonDashboard have complementary functions at the interface level. The world is not presented in terms of the active scan; the threat arc animation is informative, which sets the context of the platform as a cyber intelligence tool instead of the viewer of utilitarian reports [6]. The comparison dashboard, in its turn, is strictly analytical, it derives signal out of the difference between two assessments, allowing the practitioners to follow whether the applied remediating was maintained, whether new exposure was made, and how the overall risk trend is going across engagements.

10.48047/jocaaa.2026.35.04.10

VI. LIMITATIONS

The evidence the remediation engine gets is only as comprehensive as the engine itself. Whatever the scanners do not reveal, be it because of coverage holes in the crawling logic of Wapiti [1], a port not covered under the definition of common TCP ports, or a threat indicator not found in the VirusTotal database [4] is unaddressed in the remediation output. The engine is advisorial too; it produces text outlining remedial action, but none of the platform implements the remedial action on the target environment. Teams that expect automated remediation to be integrated with advisory guidance will not be satisfied with the platform.

The navigation configuration within the frontend has old route definitions. The sidebar links to sections of the page, such as /settings, /reports, and /monitoring, to which there are no page elements. Any user who follows these links will hit dead ends and because the routes are announced but not populated will not create routing errors that will be manifested when testing is done during development. The deeper-seated problem is that the HTTP client architecture, here the authentication calls are passed through the authApi.js, which does token refresh and error normalization centrally, and scan-related requests bypass this client and go directly to Axios. Expiry of tokens in the middle of a scan will then be treated differently than the expiry of tokens in the middle of an authentication call, and will exhibit different recovery behaviour based on the time.

The ScanContext.jsx implementation of the polling is associated with a concurrency risk. Due to the fact that setInterval fires on a fixed clock irrespective of whether the last request has been fulfilled, sluggish scans or poor network connectivity could result in overlapping requests [7]. When a response to a superseded scan cycle is received following a more recent response, the resulting displayed results will be that of the stale response a silent data corruption that can be hard to recreate and troublesome to diagnose. The Scan History page is on a stub basis: it is fed with mocked local fixtures instead of being fed by the backend API, thus it cannot allow actual forensic investigation of past assessments. Some of the comparison measures of the ScanComparisonDashboard also show hardcoded strings instead of calculated deltas, which constrains the quantitative usefulness of the temporal comparison view.

VII. FUTURE WORK

The greatest short-term effect would have been two changes in the frontend. To start with, the synchronization of all outgoing HTTP requests through one centrally configured Axios instance would bridge the difference between the way of handling token renewal and token recovery of errors in authentication and scan requests. All requests would then go through the same interceptor chain and expiry handling is therefore deterministic and error logging is also uniform. Second, the setinterval polling loop should be replaced with a sequential pattern of asynchrony where one poll should have to wait until the prior response is received before scheduling the next poll and an AbortController flag should be used to signal the end of the loop when the network is slow [7].

On the state management tier, automatically caching, background revalidation, and standardized loading and error conditions would be applied to all data-fetches boundaries within the application by adopting React Query or a similar server-state library [3]. Scanning History option must be transferred out of mocked fixtures into live API calls in order that analysts can use it to do real retrospective exploration. These changes should be accompanied by route cleanup: deleting the dead sidebar links and ensuring that all the announced routes have a rendered destination to ensure that users do not get to empty states because of them.

Integration with patch management systems is the most impactful long-horizon feature to add on the backend to convert the output of the platform based on human instructions into inputs of automated remediation pipelines [2]. To maintain the coverage of the AI engine, that is, regularly updating it with new data,. The guidance would be significantly less generic with the context of infrastructure incorporated into remediation generation to operate system family, detected application stack, existing firewall posture. The adoption of ThreatAtlas testing into the CI/CD pipelines is also another step in maturation: to cease reactive point-in-time testing and to undertake sustained security validation as a part of the very process of delivering software.

VIII. CONCLUSION

ThreatAtlas shows that future lookups of threat intelligence, port enumeration, and web vulnerability scanning can all be combined under a common orchestration layer and be operationally and technically feasible. The platform eliminates the

10.48047/jocaaa.2026.35.04.10

manual handoffs across tools allowing practitioners to focus their time on analysis instead of assembling data. The AI remediation engine goes even further to make the deliverable a list of detections into a remediation roadmap that can be taken by team members who are not specialists and can be implemented without extra research.

Being honest regarding the limitations that the platform has at present is a part of defining what it is, at this point of its development. True constraints, including polling fragility, split HTTP client architecture, non-complete integration of Scan History, and stale navigation routes, are all actual constraints that impact reliability of production. None of them is architecturally unfeasible, the paths of development chosen in Section VII have a solution to each of the limitations. What ThreatAtlas is today is a working foundation that is coherent. The route between that base and a hardened, production grade security posture platform is obviously well-defined, and the rest of the journey is one of implementation priority as opposed to fundamental feasibility.

References

[1] N. Surribas, "Wapiti: Web application vulnerability scanner," wapiti-scanner.github.io. [Online]. Available: https://github.com/wapiti-scanner/wapiti. [Accessed: Apr. 2024].

[2] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in Proc. 44th IEEE Symp. Security and Privacy (SP), San Francisco, CA, USA, May 2023, pp. 2339–2356, doi: 10.1109/SP46215.2023.10179420.

[3] Meta Open Source, "React: The library for web and native user interfaces," react.dev. [Online]. Available: https://react.dev. [Accessed: Apr. 2024].

[4] VirusTotal, "VirusTotal API v3 overview," docs.virustotal.com. [Online]. Available: https://docs.virustotal.com/reference/overview. [Accessed: Apr. 2024].

[5] J. Matherly, The Complete Guide to Shodan: Collect. Analyze. Visualize. Make Internet Intelligence Work for You. Shodan LLC, 2016. [Online]. Available: https://leanpub.com/shodan.

[6] Three.js Authors, "Three.js: JavaScript 3D library," three.js.org. [Online]. Available: https://threejs.org. [Accessed: Apr. 2024].

[7] OWASP Foundation, "OWASP Web Security Testing Guide (WSTG) v4.2," owasp.org, 2021. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/. [Accessed: Apr. 2024].