Threat Intelligence Platform- Report

Haley Nilsen, Jack Keys, Montana Nicholson, Samuel Yohannes, and Toan Nguyen

University of Missouri - Kansas City

COMP_SCI 361: Intro to CyberSecurity

A S M Touhidul Hasan

04/27/2025

Abstract

This application delivers a full-stack cybersecurity threat intelligence system that integrates real-time OSINT (Open-Source Intelligence) data, Shodan API data, caching mechanisms, and risk analysis. It is designed to provide security teams with immediate, actionable insights into potential threats through a modern, responsive web dashboard. The system uses a Node.js/Express.js backend, a React.js frontend, a SQLite database for persistence, and Redis for fast caching. Through risk prioritization, threat analysis, and alert mechanisms, this platform improves threat visibility and aids in faster incident response. Extensive performance testing and optimization ensure the platform can scale under stress while maintaining rapid response times.

Introduction

The Threat Intelligence Monitoring System is designed to automate the gathering, analysis, and prioritization of cybersecurity threats. It combines open-source intelligence (OSINT) with live threat feeds to support real-time risk monitoring. The backend (Node.js with Express) handles API integrations, data persistence, caching, and alerting, while the frontend (React.js) provides a visually organized, easy-to-use dashboard for security professionals.

Key backend technologies include:

- **SQLite** for lightweight database storage
- **Redis** for caching threat data and reducing API load
- **Axios** for external API communications (e.g., Shodan API)

- **Node Schedule** for automated OSINT refresh jobs
- **OpenAI API** for predictive risk analysis

The React frontend uses:

- Styled threat dashboards
- Dynamic chart visualizations (using Chart.js)
- OSINT data displays
- Alert notifications and manual alert triggers
- Risk prioritization features

Objectives

- **Real-time Threat Intelligence:** Pull, cache, and display threat data for IPs using OSINT APIs.
- **Risk Prioritization:** Rank threats based on likelihood, impact, and contextual data.
- **Rapid Response:** Enable faster incident detection and handling through alerts.
- **Performance Optimization:** Stress-test the system and minimize latency using Redis caching.
- **User-Friendly Interface:** Present threat and risk data clearly through a React web dashboard.
- **Maintainability and Scalability:** Provide clean, modular code with documentation for future expansion.

Key Findings

Caching with Redis significantly improves API response times, particularly under high-load conditions, ensuring consistent system performance. Integrating OSINT and Shodan threat data provides broader threat visibility compared to relying on a single source. The card-based UI on the frontend enhances user clarity, allowing security teams to quickly prioritize urgent threats. Additionally, automated alerting and scheduled data refreshes strengthen system reliability by minimizing the need for manual intervention

System Architecture

Frontend (React.js): Displays real-time threat data, OSINT results, alerts, and TVA mappings

Backend (Node.js/Express.js): APIs for threats, risk analysis, prioritization, and alerting

Database (SQLite): Tables - threat_data, alerts, tva_mapping

Caching (Redis): Speeds up repeated threat info lookups

Flow:

1. User opens dashboard → frontend requests threat/alert/OSINT APIs
2. Backend checks Redis → SQLite → OSINT API
3. Results displayed in styled cards/charts in frontend

Implementation Details

…/backend/src → All backend API logic

- ai_threat_hunting.js
- alert_manager.js
- api_optimizer.js

- api_scheduler.js
- blue_team_defense.js
- cba_analysis.js
- data.js
- fetch_osint.js
- fix-imports.js
- incident_reponse.js
- mitigation_recommendaitons.js
- nist_mapper.js
- risk_analysis.js
- risk_prioitization.js
- server.js
- shodan_integration.js
- threat_data.db
- threat_management.db
- threat_mitigation.js
- tva_update.js

…/backend/src/database → initializes database and connects to db files

- db.js

…/frontend/src/components → dashboard react components

- AlertsPage.tsx
- App.tsx
- TVAView.tsx

- ThreatDashboard.tsx

OSINT Implementation/Risk Assessment Models

The implementation of OSINT allows for threat data to be pulled from the Shodan API, and a fallback is simulated if an IP is not found. The risk assessment models that were implemented analyze risks using OpenAI's GPT and prioritize risks by likelihood and impact.

Security Features & Blue Teaming Strategies

The system implements API input validation to prevent SQL injection attacks and protect the database from damage. CORS restrictions are enforced to allow only frontend-origin requests, preventing external users from directly accessing or abusing the API. Redis caching helps prevent database overloading by checking for and serving cached IP information when available. An integrated alert system notifies blue teams of high-risk threats, enabling faster incident response. Additionally, both manual and automated threat hunting are supported through the dashboard, allowing users to perform investigations on demand while the system also conducts scheduled OSINT updates automatically.

Testing and Performance Results

Security Testing

When testing the application, ZAP was used to perform a baseline scan of the system. The test was run multiple times to gather results, and all of the tests found no critical vulnerabilities, meaning the application is secure. Load testing was also performed on the system using Apache JMeter. Multiple tests were also performed on the system and in total 2000 concurrent users were tested. The results showed that the API response time was under 500 ms for most of the requests.

Cost-Benefit Analysis & Business Justification

Costs:

- Redis = free local, $5/month cloud option
- SQLite = free
- Shodan API = free tier, upgrade costs optional

By using these programs, the application is able to achieve faster threat identification, meaning a lower breach risk of information. There is also an immediate ROI (return-on-investment) from the reduced incident response times. The scalable platform allows for future growth without a high business cost.

Challenges Faced & Lessons Learned

During the creation process, there was difficulty with integrating the real Shodan data correctly. There was no fallback logic originally implemented, which was causing issues when no data was being fetched. There were also issues with dependency downloads, but it was easily fixed by implementing a packet.json file, that way users did not have to individually download all of the dependencies used within the project.

Lessons Learned:

- Implementing a packet.json file during the beginning of the design process
- Implementing a .env file to avoid multi-file issues
- Structuring the project better to avoid “no file found” errors

Future Enhancements & Recommendations

For future additions to the project, there are a few things that can improve the application. Adding authentication to the application can allow APIs to be protected, that way they are not easily accessed by anyone using the application. There also should be role-based dashboards implemented within the application. This can allow for different views depending on if the application is being accessed by an admin or a normal analyst. The final addition that may help the application improve would be to implement scheduled automated reports, that way users may see daily threat summaries emailed directly to them.