CTI Dashboard Project Report

Project Title: CTI Dashboard - Cyber Threat Intelligence Platform
Author: Aniket Pandey
Date: September 8, 2025

1. Executive Summary

The CTI Dashboard is a comprehensive Cyber Threat Intelligence (CTI) platform designed to provide security professionals with a centralized system for aggregating, analyzing, and managing threat indicators. In an increasingly complex digital landscape, the ability to quickly ingest, enrich, and act upon threat data is critical. This project addresses that need by offering a powerful, user-friendly solution that automates data collection, provides real-time enrichment, and facilitates in-depth analysis through an intuitive interface.
The platform is built on a modern full-stack architecture, featuring a robust Python Flask backend and a dynamic React frontend. It integrates with industry-standard threat intelligence sources like URLHaus, VirusTotal, and AbuseIPDB to deliver actionable insights. Key features include an intelligent threat scoring mechanism, advanced search and filtering capabilities, role-based access control, and a comprehensive administrative dashboard for system management. This document provides a detailed overview of the project's architecture, features, and implementation.

2. System Architecture

The CTI Dashboard employs a decoupled, service-oriented architecture, with a distinct separation between the backend API and the frontend client. This design choice promotes scalability, maintainability, and flexibility.

2.1. Backend Architecture

The backend is a powerful RESTful API developed with Python and the Flask micro-framework, designed for efficiency and extensibility.

- **Framework:** Built on **Flask**, a lightweight WSGI web application framework. **Flask-RESTX** is utilized to structure the API into logical namespaces and automatically generate comprehensive Swagger UI documentation.
- **Database: MongoDB Atlas** serves as the primary data store. The non-relational nature of MongoDB is well-suited for storing the varied and nested structures of threat intelligence data. The database.py module manages the connection and implements crucial indexes to optimize query performance, including a unique compound index on IOC (type, value) and TTL (Time-to-Live) indexes on lookups and exports collections for

automatic data cleanup.

- **Authentication & Authorization:** Security is managed through **JSON Web Tokens (JWT)** using the Flask-JWT-Extended library. The system issues both short-lived access tokens (15 minutes) and long-lived refresh tokens (30 days), providing a balance of security and user convenience. A custom decorator, @require_permission, enforces a granular **Role-Based Access Control (RBAC)** system across all protected endpoints.
- **External API Integration:** The external module contains clients for **VirusTotal (v3 API)** and **AbuseIPDB**. These clients include logic for rate limiting, exponential backoff retries, and caching to ensure reliable and efficient data enrichment without exceeding free-tier API limits.
- **Asynchronous Task Scheduling:** The APScheduler library is configured to run background jobs. Its primary function is to periodically trigger the ingestion of new threat data from the URLHaus feed every 30 minutes, ensuring the platform's data remains current.
- **Modularity:** The application is organized into distinct namespaces (auth, iocs, lookup, tags, metrics, admin), which promotes a clean and maintainable codebase.

2.2. Frontend Architecture

The frontend is a modern Single-Page Application (SPA) built with React and TypeScript, delivering a fast, responsive, and interactive user experience.

- **Framework & Language:** Developed with **React 18** and **TypeScript**, ensuring a robust, type-safe, and component-based structure. **Vite** is used as the build tool, providing a fast and efficient development experience.
- **Styling: Tailwind CSS** is employed for utility-first styling, enabling rapid development of a consistent and responsive UI. The application features a custom "glassmorphic" theme with support for both light and dark modes.
- **State Management:** Global application state, such as user authentication and theme preferences, is managed by **Zustand**. Its minimalistic and hook-based API provides a simple yet powerful solution for state management.
- **Server State Management: TanStack Query** is used extensively for all interactions with the backend API. It handles data fetching, caching, and synchronization, significantly improving performance and UI responsiveness by reducing redundant network requests and managing background refetching.
- **Routing:** Client-side navigation is handled by **React Router**. The application includes a ProtectedRoute component that integrates with the authentication state to protect routes based on login status and user permissions.
- **Data Visualization:** The dashboard utilizes the **Recharts** library to render interactive and insightful charts, such as pie charts for severity distribution and area charts for activity trends.

2.3. Database Schema

The MongoDB schema is designed for performance and scalability. Key collections and their indexes include:

- **indicators**: The core collection for storing IOCs.
    - **Fields:** type, value, score, severity, tags, sources, vt, abuseipdb, first_seen, last_seen.
    - **Indexes:** A unique index on (type, value) prevents duplicates. Indexes on last_seen, severity, tags, and score ensure fast querying for filtering and sorting.
- **users**: Stores user credentials and role information.
    - **Fields:** username, email, password_hash, role.
    - **Indexes:** Unique indexes on username and email enforce data integrity.
- **tags**: Manages user-defined tags for categorizing IOCs.
    - **Fields:** name, color, description, created_by.
- **lookups**, **exports**: These collections track user-initiated tasks. They utilize a **TTL index** on the created_at field, allowing MongoDB to automatically delete old records (7 days for exports, 30 days for lookups) to keep the database clean.

4. In-Depth Feature Analysis

The CTI Dashboard provides a rich set of features for comprehensive threat intelligence management.

4.1. Threat Intelligence Aggregation & Enrichment

- **Automated Ingestion:** The ingestion/urlhaus_fetcher.py module contains a robust fetcher that runs on a 30-minute schedule. It downloads the latest URLHaus CSV feed, parses it, and intelligently updates the database. For each record, it checks if the IOC already exists. If so, it updates the last_seen timestamp and adds "urlhaus" as a source; otherwise, it creates a new IOC entry.
- **Real-time Manual Lookup:** The Lookup feature, powered by lookup/service.py, provides on-demand enrichment. When a user submits an indicator, the service first checks the local database. If the indicator exists but its enrichment data is stale (older than 24 hours), it re-enriches it. If the indicator is new, it is created and then enriched. This process ensures that users always receive the most up-to-date intelligence.

4.2. Advanced Analysis and Visualization

- **Smart Threat Scoring:** The iocs/models.py module defines a calculate_score method that computes a threat score from 0-100. This score is a weighted aggregate of several factors:

    - **Source Diversity:** More sources reporting an IOC increase the score.

    - **Recency:** More recently seen IOCs are scored higher.
    - **VirusTotal Detections:** The ratio of positive detections on VirusTotal directly contributes to the score.
    - AbuseIPDB Confidence: For IP addresses, the abuse confidence score from AbuseIPDB is factored in.
      The final score is then mapped to a human-readable severity level (info, low, medium, high, critical) by the update_severity method.

- **Advanced Filtering:** The IOCs page (frontend/src/pages/IOCsPage.tsx) offers a multi-faceted filtering system, allowing users to drill down into the data with precision. Filters include full-text search, IOC type, severity, tags, date ranges, threat score ranges, and minimum VirusTotal detections.

- **Interactive Dashboard:** The main dashboard (frontend/src/pages/DashboardPage.tsx) serves as the central hub for situational awareness. It features key performance indicators (KPIs) like total IOCs and critical threats, a severity distribution pie chart, and an IOC activity trend chart, all of which are interactive and allow users to click through to filtered views of the data.

4.3. User and Security Management

- **Role-Based Access Control (RBAC):** The auth/models.py file defines three distinct roles:
    - **Admin:** Full access to all features, including user management and system operations.
    - **Analyst:** Can view, tag, and export data.
    - Viewer: Read-only access to view threat intelligence.
      These roles are enforced on the backend using the @require_permission decorator.
- **Comprehensive Admin Panel:** The admin section (frontend/src/pages/AdminPage.tsx and backend/admin/routes.py) provides powerful tools for system maintenance, including:
    - **Manual Task Triggers:** Manually start ingestion and enrichment jobs.
    - **System Health Monitoring:** View database statistics and collection counts.
    - **User Management:** Create, edit, and delete user accounts and assign roles.
- **Background Data Export:** Users can export filtered IOC data in CSV, JSON, or XLSX format. The export process is handled as a background job on the backend to prevent timeouts with large datasets.

5. API Endpoints

The API is organized into logical namespaces. All endpoints are prefixed with /api.

5.1. Authentication (/auth)

- POST /login: Authenticates a user and returns access/refresh tokens.
- POST /register: Registers a new user.
- POST /refresh: Uses a refresh token to obtain a new access token.
- GET /me: Retrieves the profile of the currently authenticated user.

5.2. IOCs (/iocs)

- GET /: Retrieves a paginated list of IOCs with extensive filtering options.
- POST /: Manually creates a new IOC.
- GET /{ioc_id}: Retrieves detailed information for a single IOC.
- PATCH /{ioc_id}: Adds or removes a tag from an IOC.
- POST /bulk/tags: Applies tags to multiple IOCs in a single operation.

5.3. Lookup (/lookup)

- POST /: Submits an indicator for real-time lookup and enrichment.
- GET /{lookup_id}: Retrieves the status and results of a lookup task.

5.4. Tags (/tags)

- GET /: Lists all available tags.
- POST /: Creates a new tag.
- DELETE /{tag_id}: Deletes a tag (Admin only).

5.5. Admin (/admin)

- GET /users: Lists all system users.
- POST /users: Creates a new user.
- PATCH /users/{user_id}: Updates a user's details or role.
- POST /ingest/run: Manually triggers a data ingestion job.
- POST /enrichment/run: Manually triggers a bulk enrichment job.
- GET /system/stats: Retrieves system and database statistics.

6. Deployment and Operations

The project is designed for a modern, serverless deployment model using **Vercel**, which simplifies infrastructure management and provides automatic scaling for both the frontend and backend.

- **Serverless Deployment:** By deploying to Vercel, the project leverages a serverless

architecture. The React frontend is served as a static site from Vercel's global Edge Network, ensuring minimal latency for users worldwide. The Python Flask backend is deployed as **Vercel Serverless Functions**, where each API route is converted into an independent function that scales on demand.
- **Vercel Configuration:** A vercel.json file in the project's root directory orchestrates the deployment. This file defines the build processes for both the frontend and backend and configures rewrites to correctly route API requests.
    - **Builds:** The configuration specifies two separate builds: one for the Vite-based React frontend (@vercel/static-build) and one for the Flask backend (@vercel/python).
    - **Rewrites:** A rewrite rule directs all incoming requests matching the /api/(.*) pattern to the corresponding serverless function (api/index.py), effectively unifying the frontend and backend under a single domain.
- **Environment Variables:** All sensitive information, including API keys and the MongoDB connection string, is managed as Environment Variables within the Vercel project settings. This provides a secure way to handle configuration for production, preview, and development environments without hardcoding secrets into the source code.
- **Continuous Integration and Deployment (CI/CD):** Vercel's native Git integration provides a seamless CI/CD pipeline. By connecting the project's Git repository (e.g., on GitHub), every git push to the main branch can automatically trigger a new deployment to production, while pushes to other branches can generate unique preview deployments for testing and review.

7. Conclusion and Future Work

The CTI Dashboard is a powerful, flexible, and feature-rich platform for managing and analyzing cyber threat intelligence. Its modern architecture, comprehensive feature set, and user-friendly interface make it an invaluable tool for security professionals. The project successfully meets its objectives of providing a centralized, enriched, and actionable view of the threat landscape.

7.1. Future Enhancements

The modular architecture of the CTI Dashboard allows for numerous potential enhancements:
- **Additional Threat Feeds:** Integrate with more threat intelligence sources (e.g., MISP, AlienVault OTX) to provide a more comprehensive view of the threat landscape.
- **Machine Learning:** Implement machine learning models for more advanced threat scoring, predictive analysis, and anomaly detection.
- **SIEM Integration:** Develop connectors to integrate with Security Information and Event Management (SIEM) systems like Splunk or Elastic Stack, enabling seamless workflows for security operations.
- **Advanced Reporting:** Add more sophisticated reporting and data export options,

including support for standardized formats like STIX/TAXII.
- **WebSocket Integration:** Implement WebSockets for real-time push updates to the frontend, providing an even more dynamic user experience.