**Final Project Detailed Report - Real-Time Threat Intelligence and Risk Management System**

**1. Executive Summary**

The Real-Time Threat Intelligence and Risk Management System is a full-stack cybersecurity web platform designed for e-commerce companies to monitor live threats and vulnerabilities across assets, assess risk using AI-based scoring, and visualize critical security insights. The platform integrates OSINT APIs (Shodan, VirusTotal, HaveIBeenPwned), processes data in a Flask backend, stores it in PostgreSQL, and displays analysis and dashboards through a React.js frontend.

**Key Highlights:**

* Manage and display 40+ assets with live vulnerability, threat, and risk score data.
* Integrate OSINT APIs for automatic threat intelligence ingestion.
* Calculate and store AI-generated risk scores for assets.
* Interactive dashboards and asset views for security analysis.
* Dockerized architecture for easy deployment.

**2. Project Objective**

* Provide live threat monitoring for company assets.
* Automate risk assessment using OSINT data and AI scoring.
* Visualize risks, threats, and vulnerabilities with modern dashboards.
* Simplify security operations by centralizing intelligence data.

**3. System Architecture Overview**

[ User Browser (React Dashboard) ]

|

v

[ Flask Backend API ] <--> [ PostgreSQL Database ]

| |

| |---> [ OSINT APIs (Shodan, VT, HIBP) ]

|

|---> [ LLM GPT-4 Risk Score Module ]

**Flow Description:**

1. User accesses dashboard frontend.
2. Frontend fetches data from backend API.
3. Backend queries PostgreSQL database for asset, threat, vulnerability data.
4. On-demand scans trigger OSINT API calls and AI risk scoring.
5. New results are stored back in database and visualized.

**4. Database Design (ER Diagram in PNG to be generated)**

**Tables:**

**assets**

* id, name, type, department, location, ... , risk\_score

**vulnerabilities**

* id, cve\_id, description, cvss\_score, cvss\_severity

**threats**

* id, source, name, description, severity

**asset\_vulnerabilities** (M:M mapping)

* id, asset\_id, vulnerability\_id, date\_detected

**asset\_threats** (M:M mapping)

* id, asset\_id, threat\_id, date\_detected

**asset\_risk\_scores**

* id, asset\_id, risk\_score, updated\_at

**ER Diagram:**  
(Will be provided as PNG)

**5. Backend Application (Flask)**

**Technologies:** Flask, SQLAlchemy, OSINT API integration, GPT-4 scoring

**Modules:**

* /api/assets: CRUD and retrieval of asset information
* /api/scan: triggers live threat and vulnerability check + risk score
* /api/risk\_score: fetches stored risk score for asset
* /api/vulnerabilities, /api/threats: lists threats and vulnerabilities

**Risk Scoring Module:**

* Generates AI risk score by sending asset’s threat and vulnerability summary to GPT-4

**Error Handling:**

* All API and DB operations wrapped in try/except
* Validation of asset IDs
* Logs for OSINT and scoring failures

**6. Frontend Application (React.js)**

**Technologies:** React.js, Axios, Recharts, Tailwind CSS, shadcn/ui

**Main Pages:**

* Assets List (with search, filter, sorting)
* Asset Details Page (risk score, threats, vulnerabilities)
* Dashboard (live counts, graphs, top risky assets)
* Threat and Vulnerability Explorer

**Components:**

* Risk score cards
* Vulnerability and threat lists
* Pie charts, bar graphs (for severity distribution)

**User Actions Supported:**

* Generate risk score
* Filter/search assets and threats
* View live dashboard stats

**7. External Integrations (OSINT APIs)**

* **Shodan:** IP/Domain lookup → Threat data
* **VirusTotal:** Hash/URL/IP lookup → Threat data
* **HaveIBeenPwned:** Email/domain pwned data → Breach information

Data from these APIs is aggregated, processed and stored in PostgreSQL.

**8. AI Risk Scoring**

* GPT-4 is used to generate the risk\_score per asset based on vulnerabilities and threats.
* The prompt sends a structured summary and asks AI to return a numeric risk (0–100).
* This risk score is then stored and displayed in the frontend.

**9. Example API Workflow**

User → React → /api/scan → Flask → OSINT APIs → AI Risk Score → PostgreSQL → React (dashboard update)

**10. Testing & Validation**

**Unit Tests:**

* Flask service modules (pytest)

**Integration Tests:**

* API → OSINT → Risk Score → DB (mocked external APIs)

**UI Testing:**

* Manual testing via browser
* Verification of asset data, risk scores, charts

**11. Deployment**

**Dockerized Infrastructure:**

* Flask backend
* PostgreSQL database
* React frontend

**Production Environment:**

* AWS EC2 (Ubuntu)
* NGINX reverse proxy
* Docker Compose to manage all containers

**12. Challenges & Resolutions**

| **Challenge** | **Solution** |
| --- | --- |
| Dynamic data mapping from OSINT APIs | Built modular parsers for each API |
| Risk score calculation logic | Integrated GPT-4 scoring prompt |
| Slow response from LLM | Implemented background tasks and retries |
| Frontend API synchronization | Axios and useEffect optimized |

**13. Future Scope**

* Role-based user authentication
* Automated scheduled scans (via Celery Beat)
* Advanced reporting and email alerts
* Integration with Slack/SIEM tools

**14. Screenshots (Insert Here)**

* Asset Dashboard (Risk Score Overview)
* Asset Details (Threat and Vulnerability Listings)
* Threat Intelligence Dashboard (Severity/Threat Types)
* PostgreSQL Database (ER Diagram / Schema)

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**15. Appendix - Installation**

# Backend

pip install -r requirements.txt

flask run

# Frontend

npm install

npm run build

serve -s build

**Environment Variables:**

* NVD\_API\_KEY
* OTX\_API\_KEY
* VT\_API\_KEY
* HUGGINGFACE\_TOKEN

NVD Api: 883c2296-bb61-4131-9c25-6961a40eaadf

OTX : d2435e058f11fbd4c49ae3a8dd9dd8cc709088727a1a9c13774544281074180f

Virus Total: 5884a7b7d11fb8cc2f2fdceb26b0df091835389117c69a8c843bf0f131a5b12e

Hugging face api : hf\_xgpbcvHzXerVPczRMLDOOPVsQmxTDBJmaL