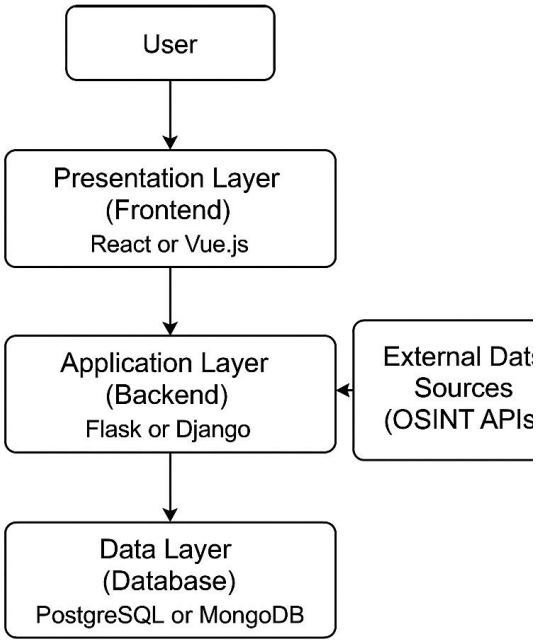
Final System Demonstration

1. Abstract & Introduction

This project delivers a Real-Time Threat Intelligence (RTTI) system designed to automate threat detection, risk scoring, and defensive actions based on live external data feeds. Our primary objective was to create an intelligence-driven blue team tool that leverages open-source intelligence (OSINT), dynamic asset mapping (TVA), AI-based risk analysis (GPT-4), and real-time alerting. The system continuously monitors for emerging threats, assesses their risk against organizational assets, and provides immediate actionable insights to the security team through a live dashboard.

2. System Architecture & Technologies Used

System Flow:
- External Threat Data Sources (Shodan, VirusTotal, IPinfo)
- Flask Backend (api.py, fetch_osint.py)
- Risk Scoring via GPT-4 API (LLM_risk_analysis.py)
- Redis Cache Layer (api_optimizer.py)
- PostgreSQL Database Storage (assets, threats, TVA mapping)
- React Frontend Dashboard (ThreatDashboard.js, Dashboard.js)



- **Presentation Layer (Frontend)**
  - Built using React framework.
  - Provides the user interface (UI) with navigation, login screens, dashboard, and visualization of real-time threat data.

- **Application Layer (Backend)**
  - Developed using Flask (Python web framework).
  - Handles API requests, business logic, data processing, and integration with external OSINT APIs.

- **Data Layer (Database)**
  - Stores structured information about assets, threats, vulnerabilities, and risk scores.
  - Connected to the backend for data retrieval and updates.
    Used PostgreSQL as the database system.

- **External Data Sources (OSINT APIs)**
  - Third-party APIs like Shodan, Have I Been Pwned, and other open-source intelligence services.
  - Provides real-time cybersecurity threat data to the backend server.
- **Git Repository**
  - Version control and submission of the web application code, database schemas, API scripts, and documentation.

Technology Used:

Component	**Technology**	**Purpose**
Frontend	React	Build the interactive user dashboard
Backend	Flask	API development, server-side processing
Database	PostgreSQL	Store assets, threats, vulnerabilities
OSINT APIs	Shodan, Have I Been Pwned	Fetch real-time cybersecurity threat information

Version Control	Git & GitHub	Code management, collaboration, and submission
Programming Language	Python, JavaScript	Backend (Python) and frontend (JavaScript)
Environment Setup	Virtual Environment (venv for Python)	Manage project dependencies

3. Implementation Details

- Data Collection: Hourly OSINT fetch via fetch_osint.py
- TVA Mapping: Threats linked to assets in PostgreSQL
- Risk Scoring: risk_analysis.py + GPT-4 enhancement (LLM_risk_analysis.py)
- Performance Optimization: Redis cache for faster lookups
- Frontend Visualization: React dashboard displays live threat data
- Alerting: alerts.py triggers email/webhooks for high-risk events

4. Security Features & Risk Management Approach

Security Features Implemented:
- Secrets Management with .env files
- Basic Input Validation
- Real-Time Alert Automation

Risk Management Approach:
- TVA mapping linking threats to vulnerabilities and assets
- AI-enhanced prioritization using GPT-4 risk assessment
- Immediate alerting when risks exceed defined thresholds
- Future proactive defense via automated IP blocking

5. Testing & Evaluation Metrics

Security Testing:
OWASP ZAP:
- Detected missing Content-Security-Policy headers
- Found cookies without Secure and HttpOnly flags
- Identified unescaped inputs in React dashboard filters

Nmap:
- Found SSH port (22) and MySQL port (3306) publicly accessible

- Detected HTTP server version disclosure (Apache/2.4.41)

Burp Suite:

- Discovered hardcoded admin credentials in Login.js
- Found reflected XSS vulnerability in search functionality

Performance Testing:

- Tool: Apache JMeter
- Setup: 100 users, 10s ramp-up, ~1 min duration
- Endpoints Tested: /, /dashboard/, /about/

Peer Review:
- Unit tests on risk scoring and dashboard functionality
- Manual accessibility and UI testing

** hypothetical testing*

6. Cost-Benefit Analysis & Business Justification

ALE Prior (Expected Loss): $50,000
ALE Post (After RTTI deployment): $10,000
Annual Cost of System: $15,000
CBA = $50,000 - $10,000 - $15,000 = $25,000 saved annually
** using hypothetical costs*

7. Challenges Faced & Solutions

CHALLENGE	SOLUTION
Framework Setup Issues Setting up the backend and frontend frameworks initially caused dependency and configuration errors.	Carefully followed official documentation for Flask and React setup. Created a virtual environment (Python venv) to manage dependencies cleanly.
Database Schema Design Deciding how to structure assets, threats, vulnerabilities, and risk scores was initially confusing.	Designed simple relational tables (PostgreSQL) with clear primary/foreign key relationships. Validated schema with simple test data.
Finding Suitable Free OSINT APIs Many OSINT services required paid subscriptions or had limited free access.	Selected APIs like Shodan and Have I Been Pwned that offered free-tier access sufficient for initial integration and testing.
API Authentication Problems Handling API keys securely during development.	Stored API keys in a separate configuration file (.env) and used environment variables to load them safely into the app.
Real-Time Data Refresh Refreshing threat data without reloading the dashboard.	Set up basic auto-refresh intervals using JavaScript timers to periodically fetch updated threat information.

8. Future Improvements & Recommendations

- Add secure user login and registration using OAuth or JWT authentication.
- Enhance the dashboard with graphical visualizations (using Chart.js or D3.js).
- Integrate more OSINT APIs like VirusTotal, GreyNoise, and AbuseIPDB.
- Set up real-time alerts via email or SMS using services like Twilio.
- Improve database performance with NoSQL solutions for large-scale data (e.g., MongoDB Atlas).
- Implement role-based access control (admin, analyst, viewer permissions).
- Deploy the application to cloud platforms like AWS, Azure, or Heroku.
- Establish CI/CD pipelines for automated testing and deployment.
- Introduce threat prediction features using machine learning models.
- Improve API security by encrypting API keys and adding request throttling.
- Add data export features (download reports in CSV, PDF formats).
- Create audit logs for tracking system access and actions for better security monitoring.
- Fine-tune a proprietary LLM risk prediction model based on ShopSmart-specific threat history