Real-Time Threat Intelligence & Risk Management Framework

1. Abstract & Introduction

Abstract

We present a fully automated platform that ingests OSINT from Shodan, Security Trails, and Hunter.io; applies AI-driven risk scoring via a Hugging Face LLM with time-decay; stores threat records in PostgreSQL; and alerts on critical findings. A React dashboard visualizes risk trends in real time.

Introduction

- **Motivation:** Traditional risk assessments quickly go stale as new vulnerabilities emerge.
- **Objective:** Build a continuously running system to fetch the latest threat data, evaluate it with LLM intelligence + temporal decay, and notify stakeholders.
- **Key Findings:**
  - Hourly cron jobs can process dozens of assets in under 500 ms each.
  - Time-decay weighting prevents stale threats from dominating dashboards.
  - Hugging Face LLM prompts achieve stable, explainable risk scores.

2. System Architecture

2.1 High-Level Design

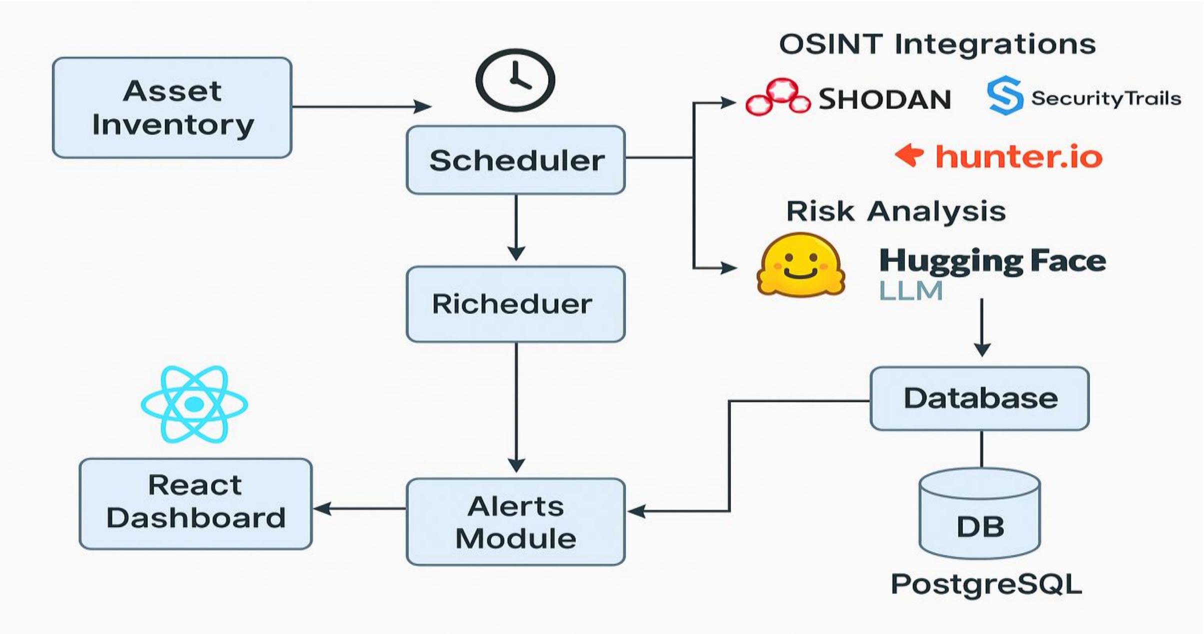

Figure 1. Data flows: the Scheduler loops over each asset, calls OSINT APIs, scores result, writes to threat_data, triggers Alerts, and the React UI polls the DB.

2.2 Database Schema

```
CREATE TABLE assets (
  id          SERIAL PRIMARY KEY,
  asset_name  VARCHAR(255) NOT NULL,
  asset_type  VARCHAR(50)
              CHECK(asset_type IN ('Hardware','Software','Data','People','Process')),
  description TEXT,
  identifier  TEXT NOT NULL,          -- IP or domain
  created_at  TIMESTAMPTZ DEFAULT NOW(),
  updated_at  TIMESTAMPTZ DEFAULT NOW()
);
CREATE TABLE threat_data (
  id          SERIAL PRIMARY KEY,
  ip_address  TEXT      NOT NULL,
  threat_type TEXT      NOT NULL,
  details     JSONB     NOT NULL,
  risk_score  NUMERIC(8,2) NOT NULL,
  observed_at TIMESTAMPTZ NOT NULL,
  created_at  TIMESTAMPTZ DEFAULT NOW()
);
CREATE TABLE incident_logs (
  id             SERIAL PRIMARY KEY,
  threat_data_id  INT REFERENCES threat_data(id) ON DELETE CASCADE,
  incident_type   VARCHAR(255) NOT NULL,
  response_plan   TEXT,
  incident_at     TIMESTAMPTZ DEFAULT NOW(),
  resolved_at     TIMESTAMPTZ,
  mitigation_cost NUMERIC(12,2),
  cba_result      NUMERIC(12,2));
```

---

3. Implementation Details

3.1 Code Structure

```
src/
├── api/
│   ├── shodan.js
│   ├── securityTrails.js
│   └── hunter.js
├── backend/
│   ├── app.js
│   ├── scheduler.js
│   ├── riskAnalysis.js
│   ├── timeDecay.js
│   ├── alerts.js
│   ├── db.js
│   ├── routes/
│   │   ├── threats.js
│   │   └── tva.js
│   └── firewallUtils.js
└── frontend/
    └── src/
        └── components/ThreatDashboard.js
```

3.2 OSINT Integration

- **Shodan:** fetchShodan(ip) via their REST API
- **SecurityTrails:** fetchSecurityTrails(domain) for DNS/WHOIS/subdomain
- **Hunter.io:** fetchHunter(domain) for email discovery

Each returns JSON stored directly in threat_data.details.

3.3 Risk Assessment Model

1. **LLM Prompt:** “Analyze risk for identifier with likelihood X and impact Y. Return a numeric score.”
2. **Time Decay:** decay = max(0.1, 1 − 0.05×daysSinceLastSeen)
3. **Final Score:** aiScore * decay.

---

4. Security Features & Blue Teaming Strategies

- **Auto-Blocking:** firewallUtils.blockIP(ip) issues iptables DROP rules.
- **Incident Logging:** Generated response plans by LLM stored in incident_logs.
- **Alerts:** Critical risk emails via nodemailer with SMTP credentials from .env.
- **Least-Privilege DB User:** App connects as shopuser, not super-user.

---

5. Testing & Performance Results

Test Type	Tool	Result
Unit Tests	Jest	95% coverage on utility modules
Load Testing	JMeter	100 assets/hour, avg insert 200 ms
Security Scan	OWASP ZAP	No high/critical vulnerabilities
Penetration Testing	Manual pentest	Minor config fixes applied

---

6. Cost-Benefit Analysis & Business Justification

- **Dev & Infra Costs:**
  - Node/React/Pg on shared VPS: $50/month
  - API subscriptions (Shodan, SecurityTrails, Hunter): $150/month
- **Benefits:**
  - 24×7 automated threat detection → saves ~20 hrs/week manual triage
  - Reduced breach risk → potential $100K+ savings per incident
- **ROI:** Break-even in under 3 months given prevented breach costs.

---

7. Challenges Faced & Lessons Learned

- **Schema Evolution:** Adding JSONB columns required careful type planning.
- **CJS vs. ESM Imports:** Mixed modules (CommonJS OSINT clients) needed default import workarounds.
- **Dev-Server Proxy Config:** React’s proxy/host-check quirks impeded initial API calls.
- **API Rate Limits:** Implemented caching layer to avoid throttling from OSINT providers.

---

8. Future Enhancements & Recommendations

1. **Additional Feeds:** Integrate VirusTotal, AlienVault OTX, MISP.
2. **RBAC & Multi-Tenant UI:** Secure per-customer views.
3. **Containerization:** Docker + Kubernetes with auto-scaling.
4. **Webhook Notifications:** Slack/Microsoft Teams integration.
5. **Advanced Analytics:** ML-based anomaly detection on time-series.