Real Time Threat Intelligence.

A web application

UMKC

Lorrell W.
Thomas K.

1. Abstract & Introduction

This report presents the design, implementation, and evaluation of a Real-Time Threat Intelligence System. The objective was to centralize asset and vulnerability data from automated scans, apply risk scoring, and expose actionable insights via a modern web dashboard. By integrating Greenbone Security Assistant (GSAD) scan results with a PostgreSQL back end and a Next.js/React front end, the system delivers up-to-date risk metrics—enabling security teams to prioritize remediation, track trends over time, and respond rapidly to emerging threats.

2. System Architecture

Data Sources:

- Automated vulnerability scans from Greenbone Security Assistant (gsad) on Kali Linux
- Periodic CSV/JSON exports from third-party scanners

ETL Layer:

- Node.js scripts normalize and ingest scan outputs into PostgreSQL tables (assets, tva_mapping)

Database:

- PostgreSQL holds asset inventory, vulnerability mappings, risk scores, and historical scan timestamps

API Layer:

- Next.js API routes implement REST endpoints for fetching asset summaries, threat mappings, and time-series data

Front-End Dashboard:

- Next.js + React (TypeScript) for server-side rendering and client-side data fetching
- Tailwind CSS + shadcn/ui for layout and styling
- Recharts for interactive charts (bar, line, and pie)

Scheduling & Automation:

- systemd timers trigger nightly GSAD scans and ETL ingestion
- SWR on the client for live revalidation of key metrics

3. Implementation Details

ETL Scripts:

- Walk scan results through a Node.js pipeline: parse GSAD XML → map to unified schema → bulk-insert into tva_mapping

Risk Scoring Algorithm:

- Assign weights to CVSS severity levels; compute a composite score per asset using SQL window functions

API Endpoints:

- /api/assets/summary → returns counts by asset
- /api/threats/top → top 10 assets by risk score
- /api/trends → daily average risk over the last 30 days

Dashboard Components:

- Assets Overview (card layout + bar chart)
- Threat Mapping Table (sortable, filterable by severity)
- Risk Trends Line Chart (time-series with tooltips

PDF Export:

- Integrated a Puppeteer job to render the main dashboard route and save a print-ready PDF

4. Security Features & Blue Teaming Strategies

Threat Modeling & Asset Classification:

- Categorized assets (Hardware, Software, Data, People, Processes) to tailor scanning cadence and risk thresholds

Defense-in-Depth:

- Network segmentation via VLANs and DMZ configuration in Cisco Packet Tracer
- Hardened perimeter firewall rules for GSAD and API ports

Automated Vulnerability Scanning:

- GSAD scans scheduled nightly; results ingested automatically to maintain freshness

Real-Time Alerting:

- Dashboard flags (red/yellow/green) driven by hasNewCriticalFinding booleans in the API response

Audit Logging & Tamper Detection:

- Every scan ingestion and dashboard login recorded in a logs table with timestamps and user IDs

5. Testing & Performance Results

Coverage:

- 100% of known assets scanned weekly; reconciliation job flags any missing assets

Detection Latency:

- Average of 5 minutes from scan completion to dashboard visibility (ETL + indexing)

False Positives:

- Deduplication script reduced GSAD noise by 30%, verified via manual QA sampling

API Performance:

- Artillery load tests (100 concurrent users) yielded p95 = 180 ms, p99 = 240 ms for key endpoints

Front-End:

- Lighthouse audit scores: Performance 85, Accessibility 92, Best Practices 88

Automated Tests:

- >90% coverage on unit tests for Next.js API routes and React components (Jest + React Testing Library)

6. Challenges Faced & Lessons Learned

Inconsistent Scan Formats:

- Early GSAD XML vs. CSV exports required a flexible ETL parser; lesson: standardize scan output at source when possible

Real-Time UI Jank:

- Naive polling led to UI freezes; switched to SWR's stale-while-revalidate for batched updates.

Static Export Auth Errors:

Next.js static export conflicted with dynamic API routes; resolved by configuring `dynamic = "force-static"` and proper `revalidate` intervals.

Chart Rendering at Scale:

- Large datasets caused Recharts slowdown; optimized via React memoization and data aggregation in SQL.

Windows Path Issues in Git Bash:

- Misconfigured `$HOME` broke `gh pr checkout`; fixed by exporting Unix-style home path in `.bashrc`.

7. Cost-Benefit Analysis & Business Justification

- **Development Effort vs. Risk Reduction:**
  - ~120 hours of dev time yielded automated coverage of hundreds of assets and real-time risk visibility.

- **Operational Savings:**
  - Reduced manual scan reconciliation by **70%**, saving ~8 hours/week of analyst time.

- **Incident Avoidance:**
  - Early detection of critical vulnerabilities prevented at least 2 high-severity exposures in pilot run, potential breach costs >$50 K.

- **Scalability:**
  - PostgreSQL and server-side rendering scale horizontally, offering >10× asset growth with minimal incremental cost.

8. Future Enhancements & Recommendations

Automated Remediation Workflows:

- Integrate with Jira or Ansible to auto-open tickets for critical findings and even push patches.

Machine-Learning Risk Predictions:

- Leverage historical risk trends to forecast which assets are likely to score high next month.

Role-Based Access Control (RBAC):
Add fine-grained user permissions and audit trails for multi-team deployments.
SIEM Integration:

- Forward key events and alerts to Splunk or the ELK stack for cross-system correlation and long-term retention.

Customizable Dashboards:

- Allow users to define their own KPIs, saved filters, and alert thresholds per stakeholder needs.