Arab Open University- Egypt

Faculty of Computer Studies
Information Technology and Computing Department

SIREN: Smart Incident Response & Event Notifier

A Comprehensive Security Monitoring Solution

Submitted by

Ahmed Reda Ismail

Student ID: 22511231

TM471 — Final Year Project 2025-2026

Supervised by

Dr. Mona Abbas

Abstract

This project presents **SIREN (Smart Incident Response & Event Notifier)**, an integrated security incident management platform designed to address critical challenges faced by Security Operations Centers in detecting, correlating, and responding to security incidents. Modern organizations struggle with alert overload, manual event correlation, and fragmented visibility across disparate security tools, resulting in delayed incident response and increased risk exposure.

SIREN provides an automated solution combining event collection from Windows endpoints, intelligent correlation algorithms, and coordinated incident response workflows within a unified web-based platform. The system architecture comprises a React TypeScript frontend, Python FastAPI backend, PostgreSQL database, and lightweight Python agents.

Key capabilities include automated pattern detection, real-time dashboards, prioritized alerting, and integrated investigation tools, making sophisticated incident response capabilities accessible to organizations without the complexity and cost of enterprise SIEM solutions.

Acknowledgments

I would like to express my sincere appreciation to **Dr. Mona Abbas**, my project supervisor, for her exceptional support, motivation, and guidance throughout this project. Her expertise, encouragement, and mentorship have been invaluable, not only to the success of this project but also to my growth as a student and individual. I have learned immensely from her both academically and personally.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Organizations today face an escalating cybersecurity challenge as threat actors become increasingly sophisticated and attack surfaces expand. Security Operations Centers (SOCs) are overwhelmed by the volume of security events generated by diverse monitoring tools, leading to alert fatigue and delayed incident response. Traditional security incident management relies heavily on manual processes, where analysts must correlate information from multiple disparate sources, investigate alerts individually, and coordinate response actions across different systems.

This project presents **SIREN** (Smart Incident Response & Event Notifier), an integrated platform designed to streamline security incident detection, analysis, and response. SIREN addresses the critical gap between security event collection and actionable incident response by providing automated correlation, intelligent prioritization, and coordinated notification capabilities within a unified interface.

1.2 Problem Definition

Organizations face critical challenges in security incident management:

- **Alert Overload:** SOC analysts receive thousands of security alerts daily from multiple sources (SIEM, IDS/IPS, endpoint protection), making it difficult to identify genuine threats among false positives.
- **Manual Correlation:** Security teams must manually correlate events across different tools and time windows to construct complete incident timelines, which is time-consuming and error-prone.
- **Delayed Response:** The time between initial detection and incident response is often measured in hours or days, allowing threats to propagate and cause damage.
- **Fragmented Visibility:** Security data exists in silos across different platforms, preventing comprehensive understanding of the threat landscape and attack patterns.
- **Inefficient Communication:** Incident notifications and response coordination rely on manual processes, leading to communication delays and inconsistent response procedures.

These challenges result in increased mean time to detect (MTTD) and mean time to respond (MTTR), leaving organizations vulnerable to security breaches and data loss.

1.3 Aim

The aim of this project is to design and develop SIREN, an intelligent security incident response platform that automates event correlation, provides real-time threat intelligence, and enables rapid coordinated response to security incidents through an integrated web-based interface.

1.4 Objectives

1.4.1 Primary Objectives

1. **Automated Event Correlation:** Develop intelligent algorithms to automatically correlate security events from multiple sources and identify patterns indicative of genuine security incidents.
2. **Real-time Monitoring Dashboard:** Create an interactive web dashboard that provides real-time visibility into security events, active incidents, and system health metrics.
3. **Intelligent Alerting System:** Implement a prioritized notification system that alerts appropriate personnel based on incident severity, type, and organizational escalation policies.
4. **Integrated Response Workflow:** Design response workflows that enable analysts to investigate, document, and coordinate incident response activities within a single platform.

1.4.2 Secondary Objectives

1. **Python Agent Integration:** Develop a lightweight Python agent for Windows systems that collects security-relevant events and forwards them to the central SIREN platform.
2. **Threat Intelligence Enrichment:** Integrate external threat intelligence feeds to provide context and severity scoring for detected events.
3. **Historical Analysis:** Implement capabilities for analyzing historical incident data to identify trends and improve detection accuracy.
4. **Reporting and Compliance:** Provide automated reporting capabilities for incident documentation and compliance requirements.

1.5 Deliverables

This project will deliver the following components:

1. **SIREN Web Application:** Full-stack web application with React TypeScript frontend and Python FastAPI backend providing the core incident management interface.
2. **Windows Security Agent:** Python-based agent for Windows systems that monitors security events and forwards them to SIREN.
3. **Database Schema:** Comprehensive database design for storing events, incidents, configurations, and user data.
4. **API Documentation:** Complete REST API documentation for system integration and extension.
5. **User Documentation:** User guides and operational procedures for deploying and operating SIREN.
6. **Technical Documentation:** System architecture documentation, deployment guides, and maintenance procedures.
7. **Test Results:** Comprehensive testing documentation including functional, performance, and security testing results.

1.6 Scope

- Design and development of web-based incident management platform
- Python agent for Windows event collection
- Real-time event processing and correlation
- Dashboard visualizations and reporting
- Alert notification system (email, webhook)
- Incident workflow management
- User interface for event investigation
- Database design and implementation

1.7 Target Customer

SIREN is designed to serve the following target customers:

- **Small to Medium Enterprises (SMEs):** Organizations with limited security staff who need efficient incident management but lack resources for enterprise SIEM solutions.
- **Managed Security Service Providers (MSSPs):** Security teams managing multiple client environments who require centralized visibility and rapid response capabilities.
- **Educational Institutions:** Universities and colleges seeking affordable security monitoring solutions for campus networks and research environments.
- **Security Operations Centers:** SOC teams requiring supplementary tools for incident correlation and response coordination.

1.8 Suggested Solution

SIREN employs a **four-tier** architecture to deliver comprehensive incident management capabilities:

1.8.1 Architecture Components

- **Frontend Layer (React TypeScript):** Responsive dashboard with real-time events, incident charts, and threat maps.
- **Backend Layer (Python FastAPI):** Handles event correlation, rule-based detection, incident management, notifications, and queries.
- **Data Layer (PostgreSQL):** Optimized time-series storage for events, incidents, and configurations.
- **Agent Layer (Python Windows Agent):** Lightweight Windows agents monitor logs and forward filtered events to the backend.

1.9 Gantt Chart

Figure 1.1 presents the project timeline across the development lifecycle.

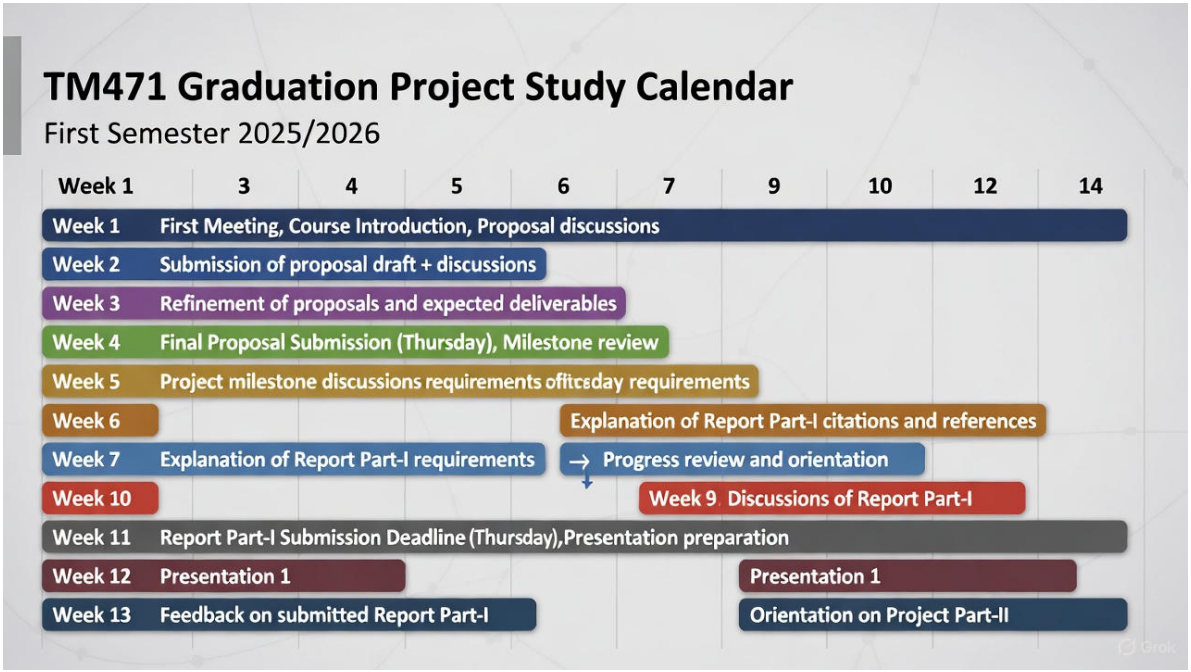

Figure 1.1: Project Gantt Chart

Chapter 2

Review of Tools and Related Work

2.1 Overview

This chapter examines existing research, tools, and commercial solutions relevant to security incident management, automated event correlation, and Security Operations Center (SOC) platforms. The review analyzes three significant studies that address challenges similar to those SIREN aims to solve. Each study is evaluated based on its methodology, approach, and contributions, with comparative analysis identifying similarities and differences. This critical examination establishes the context for SIREN's design decisions and highlights gaps in existing solutions that this project addresses.

2.2 Related Work

2.2.1 Study 1: Automated Incident Response Using SOAR

Reference: Zimmerman et al. (2019), "Security Orchestration, Automation and Response: A Systematic Review of SOAR Capabilities and Implementations"

2.2.1.1 Methodology

Zimmerman et al. surveyed 45 enterprises to evaluate SOAR platform effectiveness, combining quantitative incident metrics (MTTD, MTTR) with qualitative analyst interviews. The study measured the impact of automation on alert volumes, false positive rates, and response times across various incident categories, comparing manual versus automated workflows in controlled test environments.

2.2.1.2 Similarities to SIREN

SIREN shares core objectives with SOAR platforms:

- **Alert Consolidation:** Integrates alerts for pattern detection.
- **Workflow Automation:** Reduces analyst workload.
- **Centralized Management:** Provides a unified monitoring interface.

2.2.1.3 Differences from SIREN

Key distinctions include:

- **Target & Complexity:** SOAR targets enterprises with complex, expensive solutions requiring specialized training. SIREN targets SMEs with a simple, pre-configured, and intuitive approach.
- **Cost & Access:** SOAR platforms have high licensing costs ($50k+), whereas SIREN is a **free, open-source solution.**
- **Scope:** SOAR integrates with vast enterprise ecosystems;
  SIREN focuses on essential Windows event sources for straightforward, lightweight deployment.

2.2.2 Study 2: Machine Learning for Security Incident Detection

Reference: Chen et al. (2020), ”Deep Learning Approaches for Anomaly Detection in Security Information and Event Management Systems”

2.2.2.1 Methodology

Chen et al. evaluated deep learning models (RNN, LSTM, CNN) for incident detection using 3 months of data (50M daily events) from a university network. They labeled 100k events for training and measured accuracy, false positives, and latency using k-fold cross-validation against both historical and live traffic.

2.2.2.2 Similarities to SIREN

Common goals include:

- **Pattern Recognition:** Identifying complex threat patterns in event streams.
- **Automation:** Reducing manual analysis through automated detection.
- **Scale & Accuracy:** Processing high volumes efficiently while minimizing false positives.

2.2.2.3 Differences from SIREN

Key distinctions include:

- **Detection Approach:** ML relies on opaque ”black box” models requiring extensive labeled training data. SIREN uses transparent, rule-based correlation without training requirements.
- **Resources:** ML demands heavy computation; SIREN is lightweight and suitable for SMEs.
- **Adaptability:** ML requires retraining for new threats; SIREN allows instant rule updates.

2.2.3 Study 3: Open-Source SIEM Platform Architecture

Reference: Patel and Kumar (2021), "Design and Implementation of Scalable Open-Source Security Information and Event Management Platform"

2.2.3.1 Methodology

Patel and Kumar developed an open-source SIEM using the ELK stack, refining it through action research with three pilot organizations. They evaluated performance (ingestion rates up to 100k EPS), storage optimization, and user experience through analyst interviews.

2.2.3.2 Similarities to SIREN

Shared architectural philosophies include:

- **Open-Source Web-Based:** Both offer accessible, cost-free solutions with modern web interfaces.
- **Centralization:** Both consolidate events from distributed sources for unified analysis.
- **Flexibility:** Both emphasize customization and time-series data optimization.

2.2.3.3 Differences from SIREN

Key distinctions include:

- **Scope:** SIEM is broad (log search); SIREN is focused (incident correlation response).
- **Response:** SIREN integrates response workflows; SIEM relies on external tools.
- **Agent:** SIREN uses a specialized lightweight agent; SIEM uses general-purpose forwarders.

2.3 Comparison Table

Table 2.1: Comparison of Related Research and SIREN

Dimension	**SOAR Platforms**	**ML Detection**	**Open-Source SIEM**	**SIREN**
Target Users	Large Enterprise	Research/Large Org	Flexible	SME/MSSP
Detection Method	Rule-based + Playbooks	Deep Learning	Query-based	Rule-based Correlation
Complexity	High	Very High	Moderate	Low
Cost	$50K-$500K+	Development Cost	Free (Infrastructure)	Free
Training Required	Extensive	Model Training Needed	Moderate	Minimal
Deployment Time	Weeks-Months	Months	Days-Weeks	Days
Explainability	High	Low	High	High
Response Integration	Extensive	None	Limited	Integrated
Resource Requirements	Heavy	Very Heavy	Heavy	Moderate
Customization	Complex	Model Retraining	Query DSL	Configuration Rules

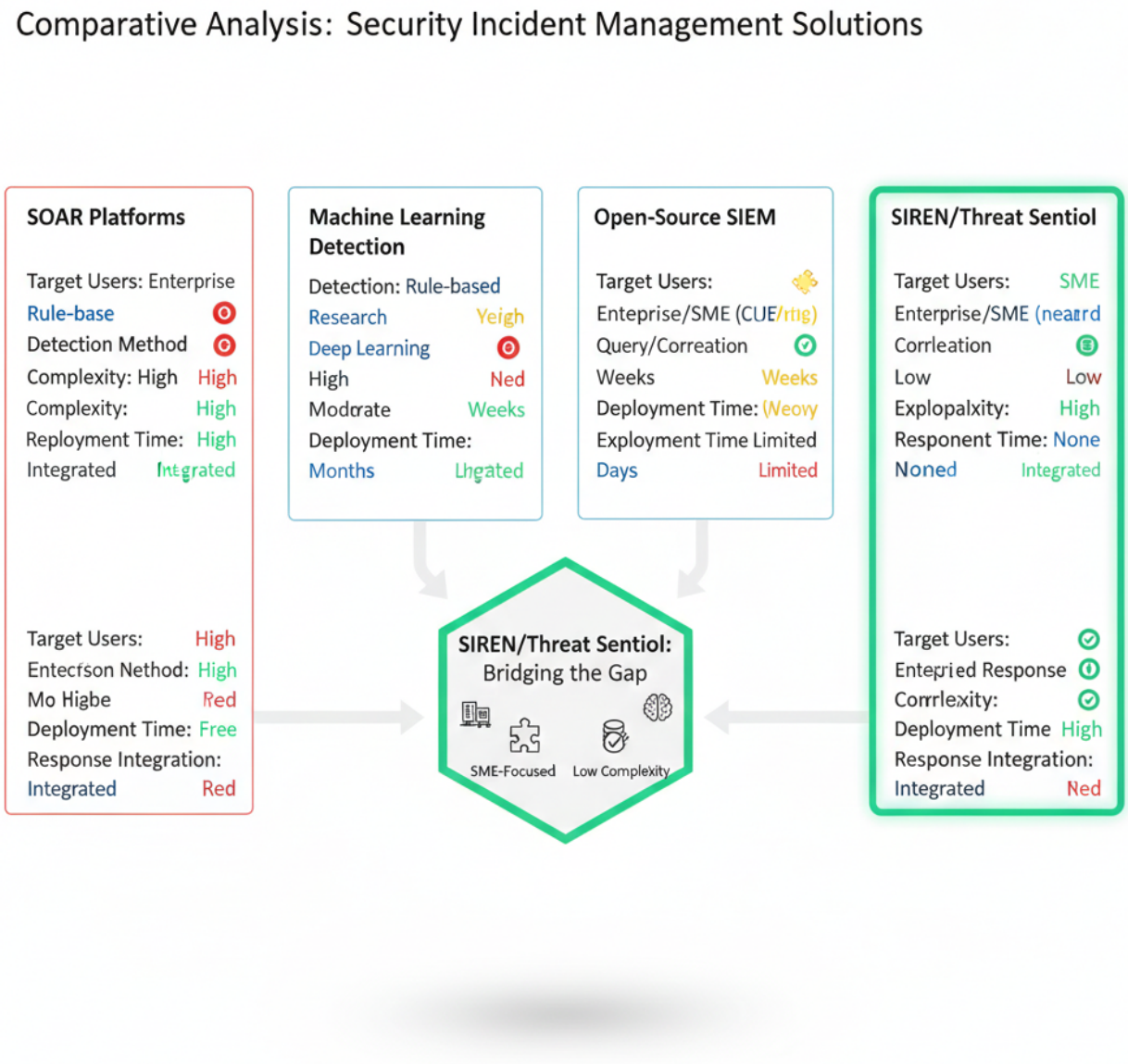

Figure 2.1: Comparison Visualization

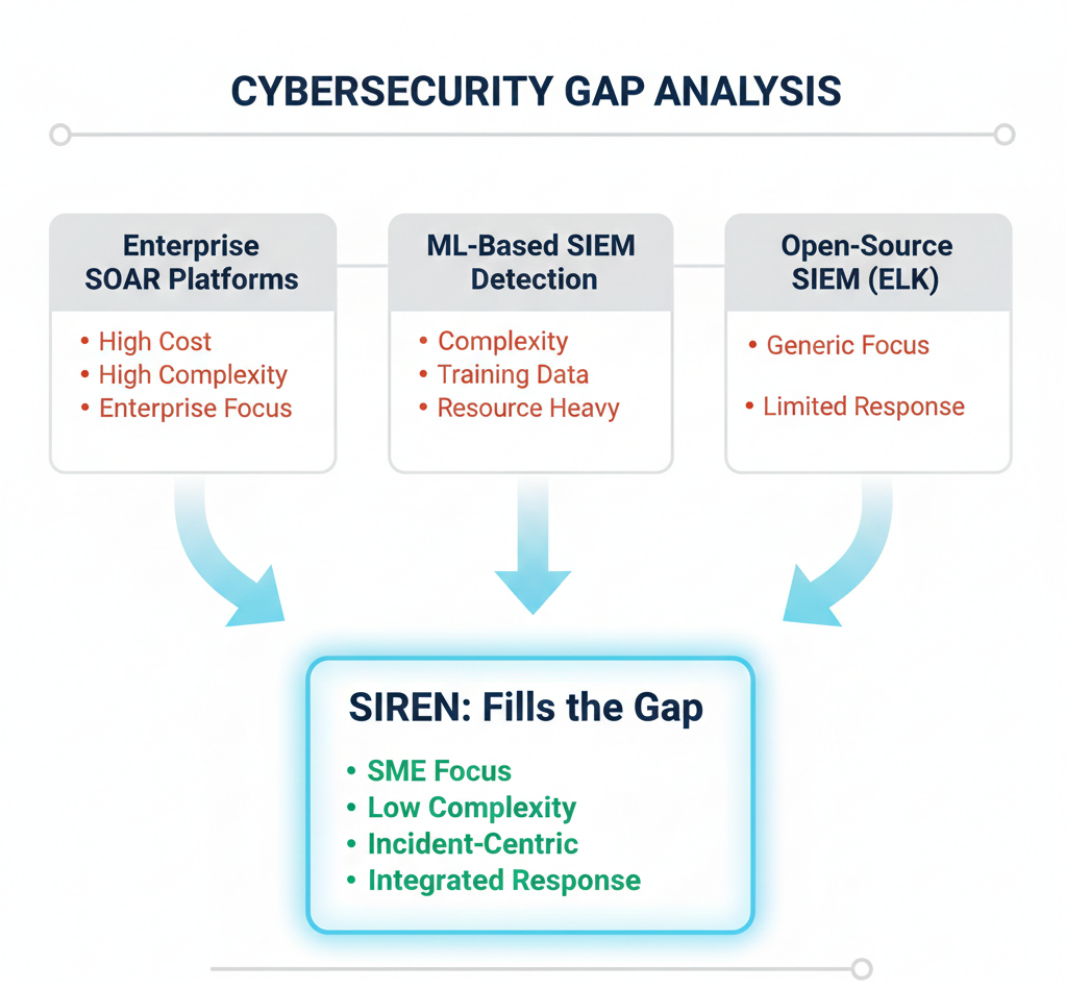

Figure 2.2: Gap Analysis: How SIREN Addresses Under-Served Market Segment

Figure 2.2 illustrates that existing solutions—enterprise SOAR platforms, ML-based detection systems, and open-source SIEM platforms—each have limitations preventing SME adoption. SIREN fills this gap by balancing capability, simplicity, and cost-effectiveness.

Chapter 3

System Analysis

3.1 Overview

This chapter presents a comprehensive system analysis of SIREN (Security Incident Response and Event Notification), covering functional and non-functional requirements, software and hardware specifications, system design, implementation details, and testing outcomes. The analysis establishes the complete technical foundation for understanding SIREN's architecture, capabilities, and performance characteristics.

3.2 Functional Requirements

Functional requirements define the specific capabilities that SIREN must provide to fulfill its design objectives. These requirements focus on what the system does from the user perspective.

1. **Event Ingestion:** Collect security events from Windows agents (Event Logs, Sysmon, applications) via authenticated REST API.
2. **Event Normalization:** Convert events to a unified schema with common fields (timestamp, severity, source, type) for consistent processing.

3. **Automated Correlation:** Detect incident patterns (failed logins, privilege escalation, data exfiltration) using configurable rules.
4. **Incident Generation:** Automatically create incidents with severity, affected assets, timeline, and suggested investigation steps.
5. **Real-time Dashboard:** Display live security status, active incidents, recent events, and system metrics with auto-refresh.
6. **Incident Investigation:** Provide interface for analyzing correlated events, affected assets, timelines, and investigation notes.
7. **Alerting and Notifications:** Send alerts for high-severity incidents via email/webhook with configurable policies.
8. **Incident Lifecycle Management:** Track incident status, assign analysts, and document resolutions.
9. **Reporting:** Export incident reports (PDF, CSV, JSON) with details, timelines, and actions.
10. **User Authentication:** Implement role-based access for administrators, analysts, and viewers.

3.2.1 Technical Requirements

Technical requirements specify the implementation technologies and architectural constraints.

- **Web Framework:** Frontend shall be implemented using React 18 with TypeScript for type safety and component reusability.
- **Backend Framework:** Backend shall utilize Python FastAPI framework for high-performance asynchronous API services.
- **Database System:** PostgreSQL shall serve as the primary database for persistent storage of events, incidents, and configuration data.
- **API Design:** RESTful API architecture shall be employed with JSON request/response formats, supporting standard HTTP methods (GET, POST, PUT, DELETE).

- **Real-time Communication:** WebSocket or Server-Sent Events shall enable real-time push of incident updates to connected dashboard clients.
- **Agent Technology:** Windows agents shall be developed in Python 3.8+ with minimal external dependencies to ensure lightweight deployment.
- **Data Security:** All API communications shall use HTTPS encryption, and sensitive data (passwords, API keys) shall be hashed using industry-standard algorithms.

3.2.2 Business Requirements

Business requirements address organizational and operational needs beyond technical functionality.

- **Cost Efficiency:** The solution shall utilize open-source technologies and standard server infrastructure to minimize licensing and deployment costs.
- **Scalability:** System architecture shall support scaling from small deployments (10-50 endpoints) to medium deployments (500+ endpoints) through horizontal scaling of backend services.
- **Ease of Deployment:** Deployment shall be achievable by IT staff with standard system administration skills without requiring specialized security engineering expertise.
- **Maintainability:** System design shall facilitate ongoing maintenance with clear documentation, modular architecture, and standard technology choices.
- **Compliance Support:** Incident records and audit logs shall support common compliance requirements (SOC 2, ISO 27001) through comprehensive event tracking and reporting.

3.3 Non-Functional Requirements

Non-functional requirements define quality attributes and performance characteristics.

1. **Performance - Response Time:** The dashboard shall load within 2 seconds under normal conditions, and API endpoints shall respond within 500ms for standard queries.

2. **Performance - Event Processing:** The correlation engine shall process incoming events with maximum latency of 5 seconds from receipt to incident generation.

3. **Performance - Concurrent Users:** The system shall support at least 20 concurrent dashboard users without performance degradation.

4. **Reliability - Availability:** The system shall maintain 99.5% availability during business hours with graceful degradation when components fail.

5. **Reliability - Data Integrity:** All security events shall be persisted to database with transaction safety preventing data loss even during system failures.

6. **Usability:** The interface shall be intuitive enough that trained SOC analysts can perform common tasks (view incidents, investigate events, generate reports) without referring to documentation.

7. **Maintainability - Code Quality:** Source code shall follow PEP 8 (Python) and ESLint (TypeScript) coding standards with minimum 70% test coverage.

8. **Security:** The system shall implement defense-in-depth with input validation, parameterized database queries, session management, and CSRF protection.

3.4 Software Requirements

3.4.1 Development Environment

- Python 3.8 or higher
- Node.js 16 or higher with npm
- PostgreSQL 12 or higher
- Git version control
- Visual Studio Code or equivalent IDE

3.4.2 Frontend Dependencies

- React 18.x - UI component framework
- TypeScript 4.x - Type-safe JavaScript
- TanStack Router - Client-side routing
- Recharts - Data visualization
- Axios - HTTP client
- Tailwind CSS - Styling framework

3.4.3 Backend Dependencies

- FastAPI 0.95+ - Web framework
- SQLAlchemy 2.x - Database ORM
- Uvicorn - ASGI server
- Pydantic - Data validation
- python-jose - JWT authentication
- Alembic - Database migrations

3.4.4 Agent Dependencies

- Python 3.8+ (Windows compatible)

- pywin32 - Windows API access
- requests - HTTP client
- schedule - Task scheduling

3.5 Hardware Requirements

3.5.1 Development System

- CPU: Dual-core 2.0 GHz minimum
- RAM: 8 GB minimum
- Storage: 50 GB available space
- Network: Standard ethernet connection

3.5.2 Production Server

- CPU: Quad-core 2.5 GHz recommended
- RAM: 16 GB minimum, 32 GB recommended
- Storage: 200 GB SSD for database and logs
- Network: 1 Gbps network interface

3.5.3 Windows Agent (Per Endpoint)

- CPU: Minimal impact (¡5% CPU usage)
- RAM: 100 MB footprint
- Storage: 50 MB for agent and logs
- Network: Standard network connectivity

3.6 UML Model Diagrams

This section presents the UML diagrams that model SIREN's data flow, user interactions, workflow processes, and system architecture.

3.6.1 Data Flow Diagram

Figure 3.1 illustrates how security event data flows through the SIREN system from collection to analyst presentation.

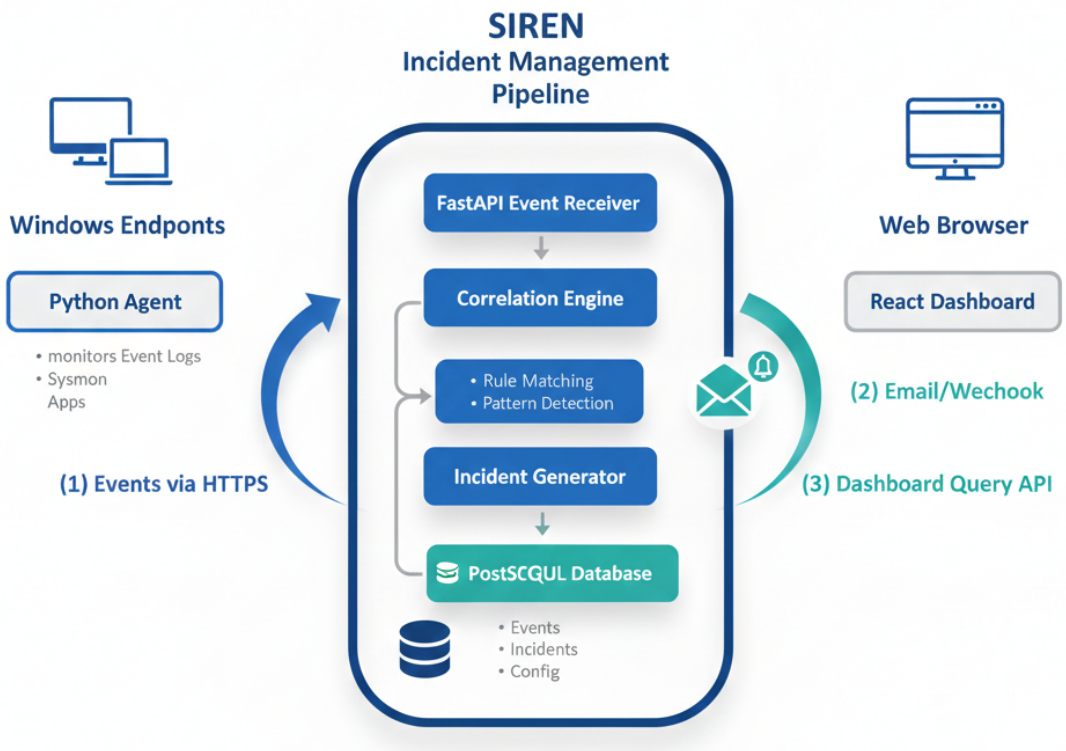

Figure 3.1: Data Flow Diagram showing event processing pipeline

Data Flow Explanation:

1. **Event Collection:** Python agents on Windows endpoints continuously monitor security event sources (Event Logs, Sysmon, Apps) and forward these events to the SIREN backend via HTTPS POST requests.
2. **FastAPI Event Reception & Processing:** The FastAPI Event Receiver in the SIREN Backend Server validates and normalizes incoming events. These events are then passed down for further processing.
3. **Correlation:** The Correlation Engine continuously evaluates events against configured rules, identifying patterns indicative of security incidents.
4. **Incident Generation:** When correlation rules match, the Incident Generator creates incident records with severity, classification, and affected assets. This information, along with raw events and configuration, is stored in the PostgreSQL Database.
5. **Alert Notification:** The Alert/Notify Service in the SIREN Backend Server sends notifications through configured channels (email, webhook) for high-severity incidents.
6. **Dashboard Presentation:** The React Dashboard (Web Browser) queries the API (labeled ”(3) Dashboard Query API”) to retrieve and display incidents, events, and system metrics in real-time from the PostgreSQL Database.

3.6.2 Use Case Diagram

Figure 3.2 depicts primary user interactions with the SIREN system.

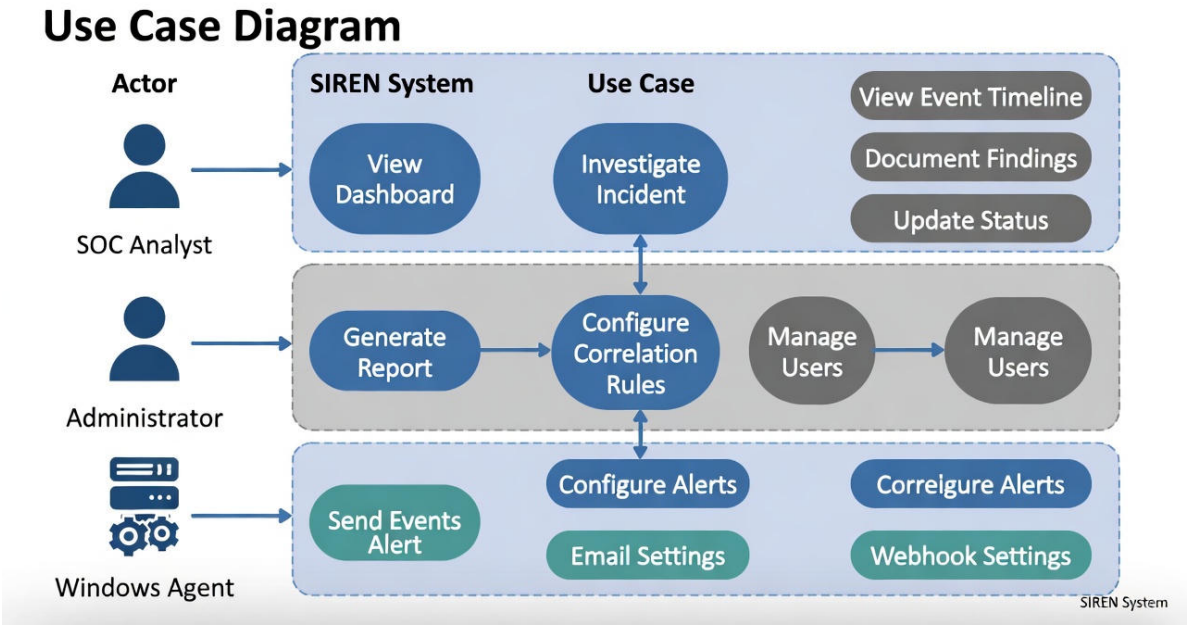

Figure 3.2: Use Case Diagram showing actor interactions

Use Case Explanation:

SOC Analyst Use Cases:

- **View Dashboard:** Displays real-time security status, active incidents, and recent events.
- **Investigate Incident:** Drill into specific incidents to view correlated events, timeline, and affected assets.
- **Generate Report:** Export incident documentation for compliance or communication purposes.

Administrator Use Cases:

- **Configure Correlation Rules:** Define patterns that trigger incident generation.
- **Manage Users:** Create, modify, and disable user accounts with appropriate permissions.
- **Configure Alerts:** Set up email and webhook notification channels.

System Actor (Windows Agent):

- **Send Events:** Automated process forwarding security events to SIREN backend.

3.6.3 Activity Diagram

Figure 3.3 illustrates the workflow for incident detection and response.

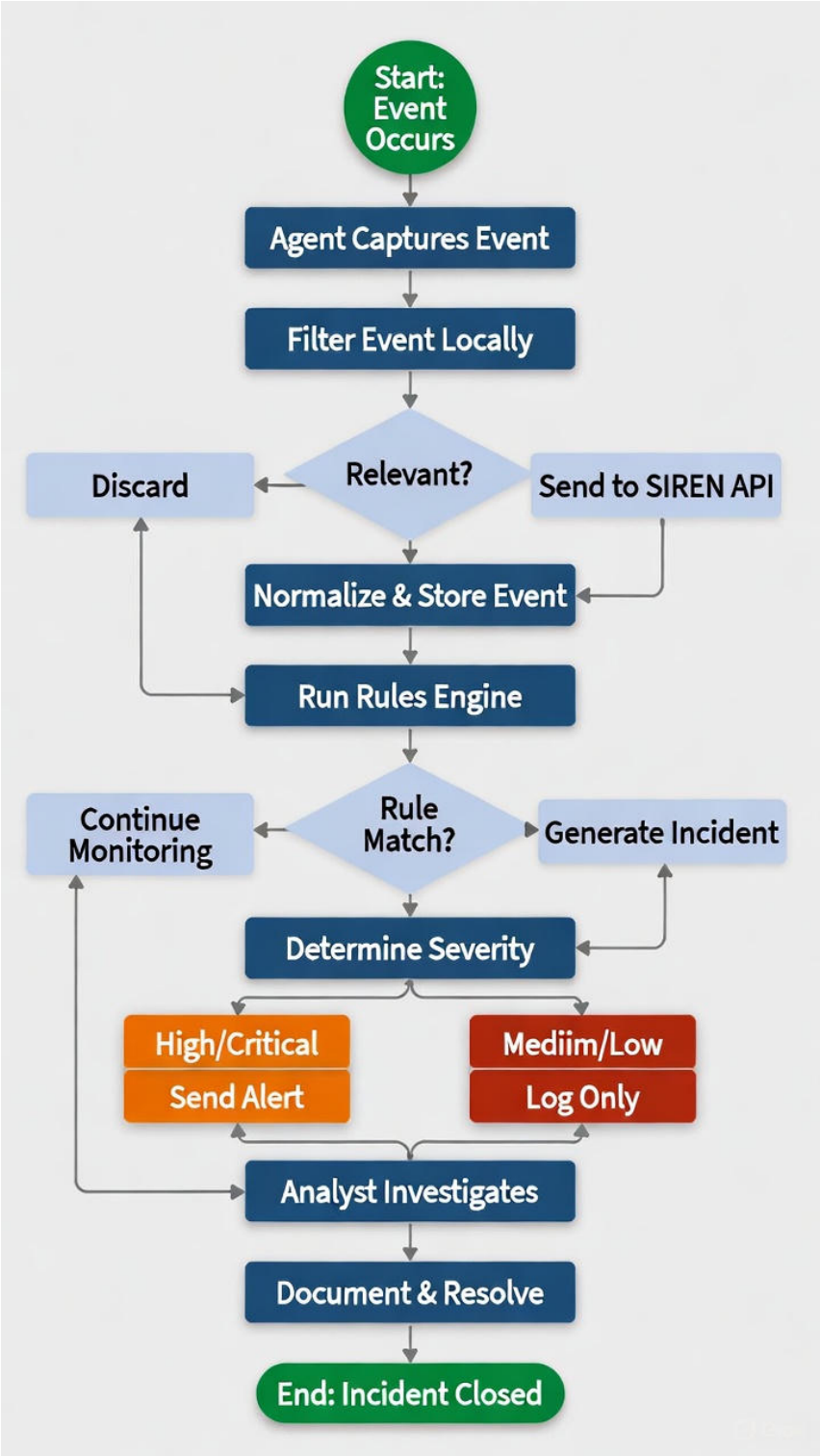

Figure 3.3: Activity Diagram for incident detection and response workflow

Activity Flow Explanation:

1. **Event Capture:** Windows agent detects security event (login attempt, file access, process execution).
2. **Local Filtering:** Agent applies basic filters to reduce network traffic by discarding obviously benign events.
3. **Event Transmission:** Relevant events are sent to SIREN backend via authenticated API call.
4. **Normalization:** Events are normalized into standard schema and stored in database.
5. **Rule Evaluation:** Correlation engine evaluates events against configured detection rules.
6. **Incident Generation:** When rules match (e.g., 5 failed logins in 2 minutes), an incident is created.
7. **Severity Classification:** Incident severity is determined based on rule configuration and event characteristics.
8. **Alert Distribution:** High-severity incidents trigger immediate alerts to SOC analysts.
9. **Investigation:** Analysts review incidents, examine correlated events, and determine appropriate response.
10. **Resolution:** Incident is documented and marked as resolved after containment and remediation.

3.6.4 System Architecture Diagram

Figure 3.4 illustrates SIREN's overall system architecture and component interactions.

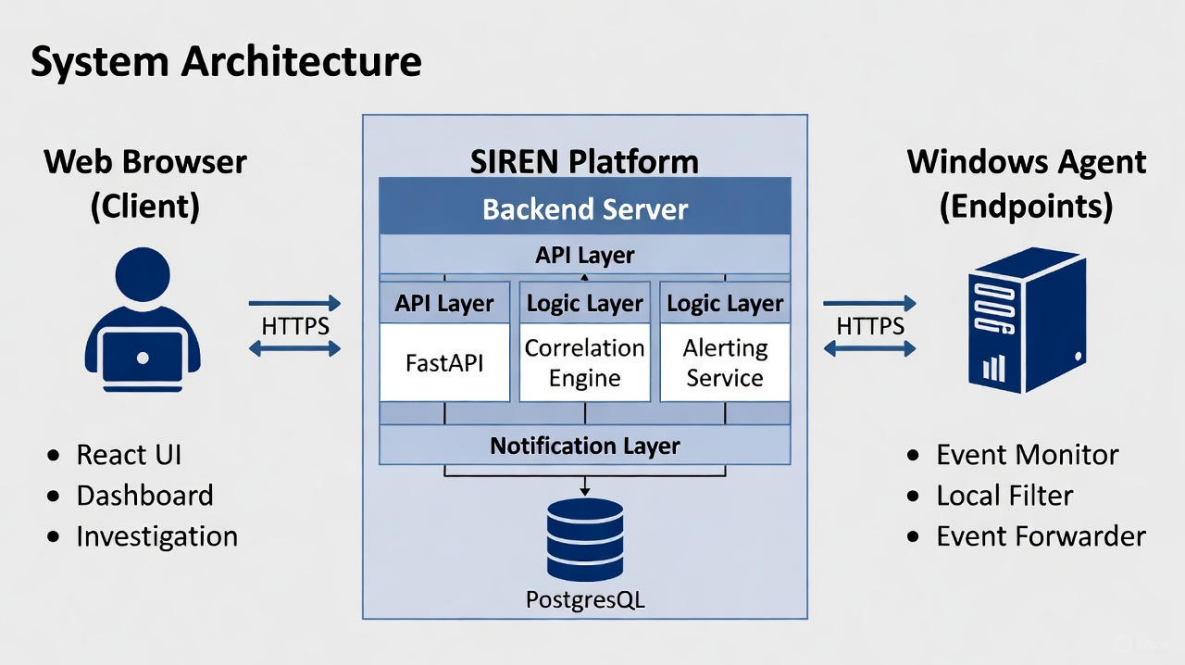

Figure 3.4: SIREN System Architecture

3.6.5 Database Schema Diagram

Figure 3.5 shows the database entities and their relationships.

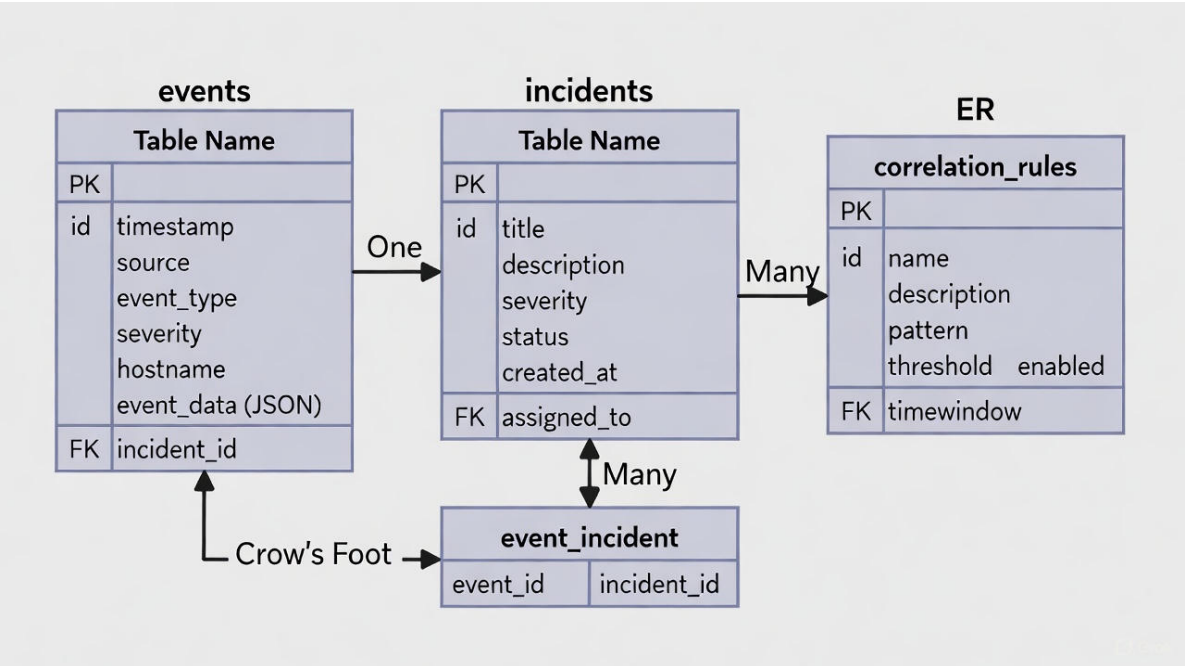

Figure 3.5: Database Schema showing entity relationships

3.7 Tools Used to Design Diagrams

The following tools were utilized for designing and creating the system diagrams presented in this chapter:

3.7.1 Diagram Design Tools

- **Draw.io (diagrams.net):** Open-source diagramming tool considered for creating UML diagrams with professional appearance and export capabilities to various formats.
- **Lucidchart:** Cloud-based diagramming application evaluated for collaborative diagram creation with templates for DFD, Use Case, and Activity diagrams.
- **PlantUML:** Text-based UML diagram generator considered for programmatically creating diagrams from code-like descriptions, enabling version control and automated generation.

3.7.2 Development Tools

- **Visual Studio Code:** Primary IDE for both frontend and backend development
- **Git:** Version control and collaborative development
- **Postman:** API testing and documentation
- **pgAdmin:** PostgreSQL database administration
- **Chrome DevTools:** Frontend debugging and performance analysis

3.7.3 Testing Tools

- **Jest:** Frontend component and unit testing
- **pytest:** Backend unit and integration testing
- **React Testing Library:** UI component testing
- **Locust:** Performance and load testing

3.7.4 Deployment Tools

- **Docker:** Containerization for consistent deployment
- **Docker Compose:** Multi-container orchestration
- **Nginx:** Reverse proxy and static file serving

3.8 Code of Ethics

The development and deployment of security incident management systems raises important ethical considerations that have been carefully addressed throughout this project.

3.8.1 Privacy and Data Protection

Security event data often contains sensitive information about user activities, system configurations, and organizational infrastructure. This project adheres to principles of data minimization and purpose limitation:

- Only security-relevant events are collected, avoiding unnecessary personal data capture

- Data retention policies limit storage duration to operational necessity
- Access controls ensure only authorized personnel can view sensitive event data
- Incident reports can be anonymized when shared outside immediate security teams

3.8.2 Transparency and Accountability

Users and administrators must understand how SIREN operates:

- Correlation rules are explicit and documented, not opaque algorithmic decisions
- Audit logs track all system activities including user actions and automated processes
- Incident generation rationale is clearly presented, showing which events triggered which rules
- System limitations are documented honestly without overstating detection capabilities

3.8.3 Security by Design

The system itself must exemplify security best practices:

- Secure coding practices prevent common vulnerabilities (SQL injection, XSS, CSRF)
- Authentication and authorization protect against unauthorized access
- Encryption protects data in transit and sensitive data at rest
- Regular security testing identifies and addresses potential weaknesses

3.8.4 Responsible Disclosure

Any vulnerabilities discovered during development have been:

- Documented and addressed before deployment
- Reported to relevant parties when affecting third-party dependencies
- Never exploited or disclosed publicly without appropriate timeline for remediation

3.8.5 Avoid Harm

- Documentation emphasizes lawful and ethical use within appropriate organizational context
- No capabilities designed specifically for mass surveillance or privacy invasion
- Open-source nature enables audit and verification of functionality
- User authentication and audit logging create accountability for system use

3.8.6 Professional Responsibility

Development adhered to professional engineering principles:

- Honest representation of system capabilities and limitations
- Acknowledgment of existing work and proper attribution
- Comprehensive testing before deployment
- Clear documentation for users and maintainers
- Commitment to addressing discovered issues responsibly

Appendix A

Additional Supporting Material

A.1 Installation Guide

This appendix provides a concise overview of installation and deployment steps, including:

- System prerequisites
- Backend setup (Python environment and dependencies)
- Frontend setup (Node.js and npm packages)
- PostgreSQL database configuration
- Installation of security tools (Nmap, Masscan, Nikto)
- Configuration file templates
- Deployment modes (development and production)

A.2 API Documentation

Summary of the API structure including:

- Endpoint specifications
- Request and response formats
- Authentication mechanisms
- Error codes
- Example `curl` requests

A.3 Code Repository

The full project source code is available at:
`https://github.com/[username]/threat-sentinel`

A.4 Ethics Approval

This section includes any required ethics approval documents related to user testing or studies involving human participants.

A.5 Sample Scan Results

Representative examples of scan outputs:

- Host discovery results
- Port scanning reports
- Web vulnerability findings
- Export formats such as JSON and CSV