Sentinel

Security Incident Detection & Response System

Enterprise-Grade SOC Automation Platform

Project Documentation

Version 3.0

December 2025

Developed for Security Operations and Incident Response

Table of Contents

1. Introduction & Objective

Sentinel is a comprehensive, real-time Security Incident Detection and Response System designed to automate Security Operations Center (SOC) workflows. The system provides enterprise-grade capabilities for threat detection, incident response, and security automation.

1.1 Project Objective

The primary objective is to develop a fully automated security monitoring platform that:

- Continuously monitors system logs for security threats
- Applies intelligent detection rules to identify incidents
- Enriches incidents with threat intelligence data
- Executes automated containment actions
- Provides real-time visualization and reporting

1.2 Key Features

- 5 Detection Rules: Brute force, rapid attempts, sudo failures, off-hours login, user enumeration
- Automated Response: IP blocking, process termination, email alerts
- Threat Intelligence: GeoIP location, IP reputation scoring (AbuseIPDB)
- Real-time Dashboard: Streamlit-based visualization with filtering and export
- Configurable Policies: Per-severity automation controls

2. Theoretical Background

2.1 Security Operations Center (SOC)

A Security Operations Center is a centralized unit that monitors, detects, analyzes, and responds to cybersecurity incidents. Traditional SOCs require significant manual effort, leading to alert fatigue and delayed response times.

2.2 Incident Response Lifecycle

The system implements the standard IR lifecycle:

- Detection: Identify security events through log analysis
- Analysis: Apply detection rules to classify incidents
- Containment: Execute automated response actions
- Eradication: Block malicious IPs and terminate processes
- Recovery: Document and report incidents
- Lessons Learned: Track metrics and improve detection

2.3 Threat Intelligence

Threat intelligence enriches incidents with contextual data including geographic location, ISP information, and reputation scores. This enables risk-based prioritization and informed decision-making.

2.4 Automation in Security

Security automation reduces mean time to respond (MTTR) by executing predefined actions without human intervention. The system implements configurable automation policies that balance security with operational requirements.

3. System Design & Architecture

3.1 Architecture Overview

The system follows a three-tier architecture with clear separation of concerns:

- Detection Layer: Monitors logs, parses events, applies detection rules
- API Layer: Enriches incidents, manages database, executes automation
- Presentation Layer: Provides dashboard, reporting, and configuration

3.2 Component Interaction Flow

1. Detection Agent monitors system logs (journalctl)
2. Log Parser extracts authentication events
3. Detection Rules identify security incidents
4. Containment Engine applies automated response
5. API Backend enriches with threat intelligence
6. Database stores incident (42 fields)
7. Alert Manager sends notifications
8. Dashboard displays real-time visualization
9. User manages incidents and policies

3.3 Technology Stack

- Backend: Python 3.8+, Flask (REST API)
- Database: SQLite with SQLAlchemy ORM
- Dashboard: Streamlit (real-time visualization)
- Detection: Custom rule engine with time-window analysis
- Threat Intel: ip-api.com (GeoIP), AbuseIPDB (reputation)
- Logging: systemd journald with fallback to file-based

4. Core Components

4.1 Detection Engine

The detection engine is the core of the system, responsible for monitoring logs and identifying security incidents.

Key Modules:

- detection_agent.py: Main monitoring agent (430 lines)
- log_parser.py: Parses journalctl and auth.log (281 lines)
- detection_rules.py: Implements 5 detection rules
- containment.py: Executes automated response actions
- system_info.py: Collects system context
- network_monitor.py: Monitors network metrics

4.2 API Backend

Flask-based REST API that handles incident submission, threat intelligence enrichment, and database operations.

Endpoints:

```
POST /api/alert          - Submit incident
GET  /api/incidents      - List all incidents
GET  /api/incident/<id>  - Get incident details
POST /api/incident/<id>/resolve - Update status
```

4.3 Dashboard

Streamlit-based dashboard providing real-time visualization, filtering, export, and configuration management.

- Real-time updates (5-second auto-refresh)
- Advanced filtering (severity, IP, target, date, status)
- Export functionality (CSV, JSON)
- PDF report generation
- Automation policy configuration

5. Detection Rules & Automation

5.1 Detection Rules

The system implements 5 detection rules:

Rule	Threshold	Window	Severity
Brute Force	3 failed logins	60 sec	High
Rapid Attempts	10 failed logins	30 sec	Critical
Sudo Failures	3 sudo fails	5 min	High
Off-Hours Login	Login 10PM-6AM	N/A	Medium
User Enumeration	5 invalid users	2 min	Medium

5.2 Automation Policies

Automation policies control what actions are executed when incidents are detected. Policies are configurable per severity level.

Severity	Block IP	Email	Desktop Alert
Critical	Yes	Yes	Yes
High	Yes	Yes	Yes
Medium	No	No	Yes
Low	No	No	Yes

6. Threat Intelligence Integration

6.1 Data Sources

The system integrates two threat intelligence sources:

1. ip-api.com (Free, no API key required):

- Geographic location (country, city, coordinates)
- ISP and organization information
- Proxy/VPN detection

2. AbuseIPDB (Optional, free API key):

- IP reputation score (0-100)
- Abuse confidence level
- Total abuse reports

6.2 Risk Scoring Algorithm

Incidents are automatically scored based on abuse confidence, proxy/VPN usage, and hosting provider identification.

Risk Levels:

Low (0-24)	Medium (25-49)	High (50-74)	Critical (75-100)

7. Implementation & Deployment

7.1 System Requirements

- Operating System: Linux (Debian/Ubuntu/Kali recommended)
- Python: 3.8 or higher
- Permissions: Root access for containment actions
- Memory: ~200 MB (all components)
- CPU: <5% idle, <20% under load

7.2 Installation Steps

```
# 1. Install dependencies
pip install -r requirements.txt

# 2. Configure environment
cp .env.example .env

# 3. Initialize database
python3 server_backend/models.py

# 4. Start the system
sudo ./start_real_mode.sh
```

7.3 Critical Fix: sshd-session Support

Modern SSH servers (OpenSSH 9.0+) use "sshd-session" as the syslog identifier. The log parser monitors both identifiers for compatibility.

8. Sample Output & Screenshots

8.1 Detection Agent Output

When an incident is detected, the agent outputs:

```
[ALERT] INCIDENT DETECTED
Type: Brute Force
Severity: High
IP: 192.168.1.17
Target: root
Rule: 3+ failed login attempts
Timestamp: 2025-12-20 19:26:12

[STATS] Collecting system context...
  OS: Linux 6.5.0
  CPU Usage: 15.2%

[SEND] Sending alert to API backend...
[OK] Alert sent successfully
```

8.2 Database Schema

Each incident is stored with 42 fields including core data, GeoIP, threat intelligence, and SOC metadata.

8.3 API Response Example

```
{
  "id": 654,
  "type": "Brute Force",
  "severity": "High",
  "ip": "192.168.1.17",
  "geo_country": "Unknown",
  "threat_risk_level": "Low",
  "status": "Active"
}
```

9. Testing & Validation

9.1 Test Suite

The system includes comprehensive testing:

- test_detection.py - Detection rule validation
- test_api_direct.py - API endpoint testing
- test_automation_policies.py - Policy configuration tests
- test_network_features.py - Network monitoring tests
- test_backend.py - Backend functionality tests

9.2 Functional Testing

Testing methodology includes detection testing (simulate attacks), API testing (verify endpoints), and dashboard testing (real-time updates, filtering, export).

9.3 Performance Metrics

- Detection Latency: <1 second
- Database Capacity: 100,000+ incidents tested
- Memory Usage: ~200 MB (all components)
- CPU Usage: <5% idle, <20% under load
- Dashboard Refresh: 5 seconds (configurable)

10. Limitations & Future Improvements

10.1 Current Limitations

Detection Scope:

- Limited to authentication-based attacks (SSH, sudo)
- Does not monitor application-level attacks
- No file integrity monitoring

Threat Intelligence:

- Private IPs cannot be looked up in external databases
- Free tier API limits (ip-api: 45 req/min, AbuseIPDB: 1000/day)

Scalability:

- SQLite database (single-file, not distributed)
- Single-server deployment
- No load balancing or high availability

10.2 Future Improvements

Short-term (1-3 months):

- Add web application firewall (WAF) log parsing
- Implement machine learning for anomaly detection
- Add Slack/Discord notification integration

Medium-term (3-6 months):

- Migrate to PostgreSQL for better scalability
- Implement distributed detection agents
- Add SIEM integration (Splunk, ELK)

Long-term (6-12 months):

- Cloud deployment (AWS, Azure, GCP)
- Multi-tenant architecture
- Advanced threat hunting capabilities
- Compliance reporting (PCI-DSS, HIPAA, SOC 2)

10.3 Conclusion

Sentinel represents a comprehensive solution for automated security incident detection and response. The system successfully demonstrates enterprise-grade capabilities including real-time monitoring, intelligent detection, threat intelligence enrichment, and automated containment.

The project achieves its core objectives of reducing manual SOC workload, improving incident response times, and providing actionable security intelligence. With continued development, Sentinel has the potential to evolve into a production-ready security platform suitable for enterprise deployment.

Appendix A: Project Statistics

- Total Lines of Code: ~3,500
- Detection Rules: 5
- Threat Intelligence Sources: 2
- Database Fields: 42
- Test Suites: 9
- API Endpoints: 5
- Supported Log Sources: journalctl, auth.log

Appendix B: Quick Start Commands

```
# Installation
cd /home/kali/IR-Project/IR-System
pip install -r requirements.txt

# Start System
sudo ./start_real_mode.sh

# Access Dashboard
http://localhost:8501

# Run Tests
python3 tests/run_all_tests.py
```