🛡️ Sentinel Security Incident Detection & Response System

Enterprise-Grade Security Operations Center (SOC) Automation Platform

A comprehensive, real-time security incident detection and response system with automated containment, threat intelligence enrichment, and professional dashboard visualization.

📋 Table of Contents

🌟 Overview

Sentinel is a **fully automated** Security Operations Center (SOC) platform designed for real-time threat detection and incident response. It continuously monitors system logs, applies intelligent detection rules, enriches incidents with threat intelligence, and automatically executes containment actions based on configurable policies.

What Makes Sentinel Unique?

- ✅ **Fully Automated Response** - No manual intervention required
- ✅ **Configuration-Driven** - Policies managed via UI or config files
- ✅ **Threat Intelligence** - Automatic GeoIP and IP reputation enrichment
- ✅ **Production Ready** - Comprehensive testing and error handling
- ✅ **Enterprise Features** - PDF reports, email alerts, audit logging

---

🎯 Key Features

🔍 Detection & Monitoring

Feature	Description	Status
5 Detection Rules	Brute force, rapid attempts, sudo failures, off-hours login, user enumeration	✅
Real-Time Monitoring	Continuous log analysis via journalctl	✅

Feature	Description	Status
Multi-Source Support	SSH, sudo, system logs	✅
Pattern Recognition	Time-window based attack detection	✅
Network Monitoring	Ping metrics, traffic analysis, DoS detection	✅

🌍 Threat Intelligence

Feature	Description	Status
Geographic Tracking	Country, city, region, coordinates	✅
IP Reputation	AbuseIPDB integration (0-100 scoring)	✅
ISP Identification	Internet Service Provider and organization	✅
Proxy/VPN Detection	Identifies suspicious infrastructure	✅
Risk Assessment	Automatic risk scoring and classification	✅

🤖 Automated Response

Feature	Description	Status
IP Blocking		✅

Feature	Description	Status
	Automatic iptables rules for malicious IPs	
Process Termination	Kill suspicious processes	✅
Email Alerts	Professional HTML emails for high-severity incidents	✅
Desktop Notifications	Real-time alerts for all severities	✅
Whitelisting	Protect trusted IPs from blocking	✅
Configurable Policies	Per-severity automation controls	✅
Simulation Mode	Safe testing without real actions	✅

📊 Dashboard & Visualization

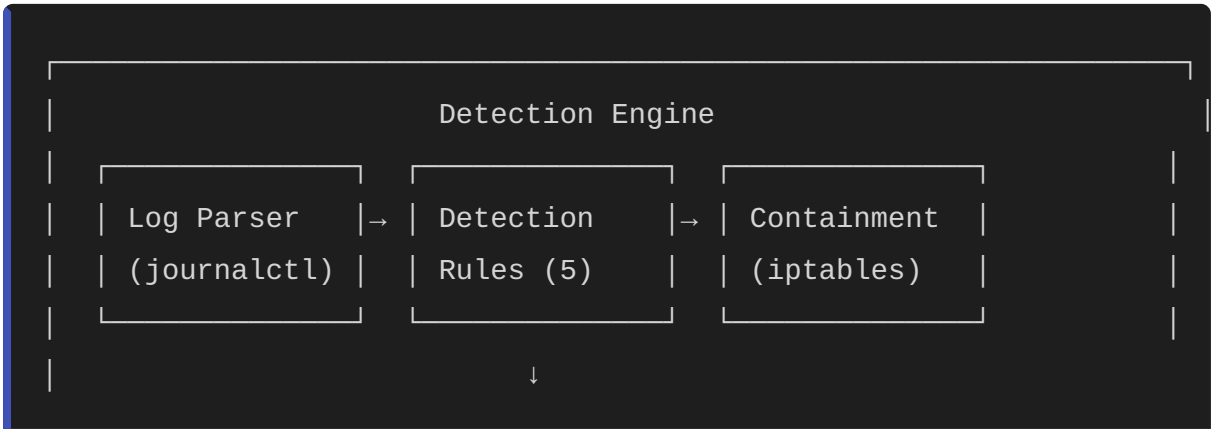
Feature	Description	Status
Real-Time Updates	Live incident monitoring (5s TTL)	✅
Threat Intel Display	Geographic and risk data visualization	✅
Export Functionality	CSV and JSON export	✅
Advanced Filtering	By severity, IP, target, status, date	✅
Charts & Analytics	Incident trends and statistics	✅

Feature	Description	Status
PDF Reports	Professional incident reports	✅
Automation Controls	UI for managing automation policies	✅
Incident Management	Resolve, close, re-activate incidents	✅

💾 Data Management

Feature	Description	Status
SQLite Database	Persistent incident storage	✅
Incident Lifecycle	Active → Resolved → Closed	✅
Historical Analysis	Complete incident history	✅
42 Database Fields	Core + Threat Intel + SOC data	✅
Automatic Backups	Database backup system	✅

🏗️ Architecture



Component Interaction Flow

1. **Detection Agent** monitors system logs continuously
2. **Log Parser** extracts authentication events
3. **Detection Rules** identify security incidents

4. **Containment Engine** applies automated response (based on policies)
5. **API Backend** enriches with threat intelligence
6. **Database** stores incident with 42 fields
7. **Alert Manager** sends notifications (email + desktop)
8. **Dashboard** displays real-time visualization
9. **User** manages incidents and configures automation

📁 Project Structure

```
IR-System/
├── 📄 Core Files
│   ├── README.md                        # Complete documentation
│   ├── Sentinel_Project_Documentation.pdf  # PDF documentation
│   ├── requirements.txt                 # Python dependencies
│   ├── whitelist.json                   # Trusted IPs (never blocked)
│   ├── ip_blacklist.json                # Known malicious IPs
│   ├── database.db                      # Main SQLite database
│   ├── threat_intel_cache.sqlite        # Threat intelligence cache
│   ├── start_sentinel.sh                # ⭐ Unified launcher (recommended)
│   ├── start_real_mode.sh               # Alternative startup script
│   ├── cleanup_temp_files.sh            # Clean temporary test files
│   └── generate_pdf.py                  # Generate documentation PDF
│
├── 🔍 detection_engine/
│   ├── detection_agent.py               # Main monitoring agent
│   ├── log_parser.py                    # Multi-format log parsing
│   ├── log_source_manager.py            # Log source coordination
│   ├── log_discovery.py                 # Automatic log file discovery
│   ├── detection_rules.py               # 5 detection rules
│   ├── containment.py                   # Automated response actions
│   ├── system_info.py                   # System context collection
│   ├── system_monitor.py                # System resource monitoring
│   ├── network_monitor.py               # Network monitoring
```

```
│   └── __init__.py
│
├── 🖥️ server_backend/
│   ├── app.py                          # Flask API + threat intel
│   ├── dashboard.py                    # Streamlit dashboard
│   ├── models.py                       # Database models (SQLAlchemy)
│   ├── threat_intel.py                 # GeoIP + AbuseIPDB integration
│   ├── threat_intel_cache.sqlite       # Threat intel cache (local)
│   ├── alert_manager.py                # Desktop + email notifications
│   ├── email_notifier.py               # SMTP email sender
│   ├── report_generator.py             # PDF report generation
│   ├── config_manager.py               # Configuration management
│   ├── config.json                     # Runtime configuration
│   ├── alerts.log                      # Alert notifications log
│   ├── automation.log                  # Automation actions log
│   └── incidents.db                    # Incidents database (symlink)
│
├── 🧪 tests/
│   ├── test_api_direct.py              # API integration tests
│   ├── test_backend.py                 # Backend tests
│   ├── test_detection.py               # Detection rules tests
│   ├── test_automation_policies.py     # Automation policy tests
│   ├── test_network_features.py        # Network monitoring tests
│   ├── test_email_config.py            # Email configuration tests
│   ├── test_real_credentials.py        # Live credential tests
│   ├── quick_test.sh                   # Quick test script
│   └── run_all_tests.py                # Test runner
│
├── 📊 Data & Logs
│   ├── archive/                        # Incident backups
│   │   └── incidents_backup_*.json
│   ├── logs/                           # Runtime logs
│   ├── reports/                        # Generated PDF reports
│   │   └── incident_report_*.pdf
│   ├── simulate_data_stream.py         # Testing/demo data generator
│   ├── simulate_realistic_attack.py    # Realistic attack simulator
│   └── generate_realtime_events.sh     # Real-time event generator
```

```
│
└── .git/                                # Version control
```

🚀 Installation

Prerequisites

System Requirements: - Linux (Debian/Ubuntu/Kali recommended) - Python 3.8+ - Root access (for real containment actions)

Required Packages:

```
# Python dependencies
pip install -r requirements.txt --break-system-packages

# Or use system packages (Kali Linux)
sudo apt install python3-flask python3-sqlalchemy python3-streamlit \
                 python3-pandas python3-requests python3-fpdf
```

Installation Steps

1. **Clone or Download the Project** `bash cd /home/kali/IR-Project/IR-System`
2. **Install Dependencies** `bash pip install -r requirements.txt`
3. **Configure Environment (Optional)** `bash cp .env.example .env nano .env`
4. **Initialize Database** `bash python3 server_backend/models.py`
5. **Verify Installation** `bash python3 tests/run_all_tests.py`

---

1. **Flask API** on http://127.0.0.1:5000
2. **Streamlit Dashboard** on http://localhost:8501

3. **Detection Agent** monitoring system logs

Access the Dashboard

Open your browser to: **http://localhost:8501**

You'll see three main views: - **Dashboard** - Real-time incident monitoring - **Real-time Monitor** - Network and traffic analysis - **Settings** - Configuration and automation controls

---

⚙️ Configuration

Environment Variables (.env)

```
# Threat Intelligence
ABUSEIPDB_API_KEY=your_key_here  # Get free at abuseipdb.com

# Email Notifications
SMTP_SERVER=smtp.gmail.com
SMTP_PORT=587
SMTP_USERNAME=your_email@gmail.com
SMTP_PASSWORD=your_app_password
SMTP_TO=security-team@yourdomain.com
```

Configuration File (server_backend/config.json)

```
{
  "ping_targets": ["8.8.8.8", "1.1.1.1", "192.168.1.1"],
  "dos_thresholds": {
    "cpu_percent": 80.0,
    "pps_in": 1000
  },
  "automation_policies": {
    "enabled": true,
```

```
    "actions": {
      "Critical": {
        "block_ip": true,
        "send_email": true
      }
    }
  }
}
```

Whitelist Configuration (whitelist.json)

```
{
  "whitelist": {
    "ips": ["127.0.0.1", "::1"],
    "networks": ["10.0.0.0/8", "192.168.0.0/16"]
  }
}
```

🎯 Detection Rules

Rule	Threshold	Time Window	Severity	Type
Brute Force	3 failed logins	60 seconds	High	Brute Force
Rapid Attempts	10 failed logins	30 seconds	Critical	Brute Force
Sudo Failures	3 sudo failures	5 minutes	High	Privilege Escalation
		N/A	Medium	

Rule	Threshold	Time Window	Severity	Type
Off-Hours Login	Login 10PM-6AM			Suspicious Time
User Enumeration	5 invalid users	2 minutes	Medium	Reconnaissance

How Detection Works

1. **Log Monitoring**: Detection agent reads system logs via journalctl
2. **Event Parsing**: Extracts authentication events (IP, user, timestamp)
3. **Pattern Matching**: Applies time-window based detection rules
4. **Incident Creation**: Generates incident with severity classification
5. **Enrichment**: Adds threat intelligence (GeoIP, abuse score)
6. **Automated Response**: Executes containment based on policies

---

🤖 Automation Policies

Overview

Automation policies control what actions are automatically taken when incidents are detected. Policies are **configurable per severity level**.

Default Policies

Severity	Block IP	Kill Process	Send Email	Desktop Alert
Critical	✅ Yes	❌ No	✅ Yes	✅ Yes
High	✅ Yes	❌ No	✅ Yes	✅ Yes
Medium	❌ No	❌ No	❌ No	✅ Yes

Severity	Block IP	Kill Process	Send Email	Desktop Alert
Low	❌ No	❌ No	❌ No	✅ Yes

Configure via Dashboard

1. Open Dashboard → **Settings**
2. Scroll to 🤖 **Automation Policies**
3. Toggle **Enable Automation** on/off
4. Click severity tabs to configure actions
5. Click 💾 **Save Automation Policies**

Configure via File

Edit `server_backend/config.json`:

```
{
  "automation_policies": {
    "enabled": true,
    "actions": {
      "Critical": {
        "block_ip": true,
        "kill_process": false,
        "send_email": true,
        "send_desktop_alert": true
      }
    }
  }
}
```

Automation Logging

All automated actions are logged to `server_backend/automation.log`:

```
[2025-12-20 15:10:23] SUCCESS - BLOCK_IP | Target: 1.2.3.4 | Severity: Critical
```

View statistics in Dashboard → Settings → Automation Statistics

---

🌍 Threat Intelligence

Data Sources

1. ip-api.com (Free, no key needed) - Geographic location (country, city, coordinates) - ISP and organization - Proxy/VPN detection

2. AbuseIPDB (Optional, free API key) - IP reputation score (0-100) - Abuse confidence level - Total abuse reports - Last reported date

Risk Scoring

Incidents are automatically scored based on: - **Abuse confidence** (if AbuseIPDB configured) - **Proxy/VPN usage** - **Hosting provider detection**

Risk Levels: - **Low** (0-24) - **Medium** (25-49) - **High** (50-74) - **Critical** (75-100)

> 📝 ***Note on Private IPs:***
> *Internal/private IP addresses (192.168.x.x, 10.x.x.x, 172.16-31.x.x) cannot be looked up in external threat intelligence databases. The system automatically assigns them a* ***"Low"*** *risk level (score: 5) instead of "Unknown". This is expected behavior for internal network incidents.*

Setup AbuseIPDB

1. Get free API key: https://www.abuseipdb.com/register
2. Add to `.env` : `bash ABUSEIPDB_API_KEY=your_key_here`
3. Restart detection agent

📊 Dashboard Usage

Main Dashboard View

Features: - Real-time incident table (auto-refresh every 5s) - Advanced filtering (severity, IP, target, date, status) - Metrics (total incidents, critical count, unique IPs) - Charts (by target, severity, IP) - Export (CSV, JSON) - Incident management (resolve, close, re-activate)

Filtering: - **Severity**: All, Critical, High, Medium, Low - **Source IP**: Partial match (e.g., "192.168") - **Target**: User or system name - **Timeframe**: All time, 24h, 7d, 30d - **Status**: Active, Resolved, Closed, All

Real-time Monitor View

Features: - Network traffic & DoS analysis - Ping latency monitoring - Live incident feed - SSH event stream

Settings View

Features: - Network monitoring configuration - DoS thresholds - Automation policies - Alert logs - Simulation control

---

📡 API Documentation

Base URL

```
http://127.0.0.1:5000
```

Endpoints

Method	Endpoint	Description
GET	`/`	Health check

Method	Endpoint	Description
POST	`/api/alert`	Submit incident
GET	`/api/incidents`	List all incidents
GET	`/api/incident/<id>`	Get incident details
POST	`/api/incident/<id>/resolve`	Update incident status
GET	`/api/status`	System status

Example: Submit Incident

```
curl -X POST http://127.0.0.1:5000/api/alert \
  -H "Content-Type: application/json" \
  -d '{
    "ip": "192.168.1.100",
    "type": "Brute Force",
    "severity": "High",
    "timestamp": "2025-12-20T10:00:00",
    "rule": "Failed Login Count Exceeded",
    "source_log": "/var/log/auth.log",
    "target": "root"
  }'
```

Example: Resolve Incident

```
curl -X POST http://127.0.0.1:5000/api/incident/1/resolve \
  -H "Content-Type: application/json" \
  -d '{"status": "Resolved"}'
```

🧪 Testing

Run All Tests

```
python3 tests/run_all_tests.py
```

Individual Tests

```
# Test API integration
python3 tests/test_api_direct.py

# Test detection rules
python3 tests/test_detection.py

# Test automation policies
python3 tests/test_automation_policies.py

# Test network features
python3 tests/test_network_features.py
```

Simulate Attacks

```
# Brute force attack (3 failed logins)
for i in {1..3}; do
  logger -t sshd "Failed password for user from 1.2.3.4 port 22"
done

# User enumeration (6 invalid users)
for i in {1..6}; do
  logger -t sshd "Invalid user test$i from 5.6.7.8 port 22"
done

# Check results
tail -f server_backend/automation.log
```

Data Simulation for Testing

For testing and demonstration, use the included simulation script:

```
# Start data simulation (SSH events, network traffic, incidents)
python3 simulate_data_stream.py
```

What it simulates: - 🔑 **SSH Events** - Random login attempts (success/failure) - 📶 **Ping Metrics** - Network latency to multiple targets - 📡 **Traffic Stats** - Packet counts with occasional DoS spikes - 🚨 **Incidents** - Random security incidents with full threat intelligence

Features: - ✅ Incidents include threat intelligence enrichment - ✅ Private IPs automatically get "Low" risk level - ✅ Realistic data patterns for dashboard visualization - ✅ Runs continuously until stopped (Ctrl+C)

Example Output:

```
✅ Threat Intelligence enabled for simulation
🔑 SSH: FAILURE - admin@192.168.1.100
🚨 New Simulation Incident: Brute Force (High) - Risk: Low
📡 Traffic: 450 PPS
```

> 💡 ***Tip:*** *The simulation is perfect for testing dashboard features, automation policies, and threat intelligence without needing real attacks.*

🔧 Troubleshooting

"SSH attempts not being detected"

Modern SSH servers (OpenSSH 9.0+) use `sshd-session` as the syslog identifier.

The log parser monitors both `sshd` and `sshd-session`. Verify:

```
journalctl -t sshd-session -n 10 --no-pager
```

"Log file not found"

- System uses journalctl as fallback
- No action needed on modern Linux systems

"Cannot connect to API"

```
# Check if Flask is running
ps aux | grep app.py

# Check API
curl http://127.0.0.1:5000/
```

"Email notifications not working"

```
# Verify SMTP credentials in .env
cat .env

# Test email
python3 server_backend/email_notifier.py
```

"Permission denied" errors

```
# Run with sudo for real containment actions
sudo ./start_real_mode.sh
```

"Dashboard not loading"

```
# Check if Streamlit is running
ps aux | grep streamlit
```

```
# Restart dashboard
streamlit run server_backend/dashboard.py
```

🎓 Advanced Usage

Custom Detection Rules

Add new rules in `detection_engine/detection_rules.py`:

```
class CustomRule(DetectionRule):
    def __init__(self):
        super().__init__(
            rule_name="Custom Rule",
            incident_type="Custom Type",
            severity="High"
        )

    def check(self, events):
        # Your detection logic
        pass
```

Export Incident Data

```
import pandas as pd
from server_backend.models import Incident, Session

session = Session()
incidents = session.query(Incident).all()
df = pd.DataFrame([{
    'id': i.id,
    'ip': i.ip,
    'country': i.geo_country,
    'risk_level': i.threat_risk_level
```

```
} for i in incidents])
df.to_csv('incidents.csv', index=False)
```

Unblock IP

```
# Via iptables
sudo iptables -D INPUT -s 1.2.3.4 -j DROP

# Or use containment API
python3 -c "from detection_engine.containment import ContainmentActions; \
            c = ContainmentActions(simulation_mode=False); \
            c.unblock_ip('1.2.3.4')"
```

📈 Performance

- **Detection Latency**: <1 second
- **Database**: Handles 100K+ incidents
- **Memory Usage**: ~200 MB (all components)
- **CPU Usage**: <5% idle, <20% under load

🔒 Security Considerations

Production Deployment

1. **Change default ports** (Flask, Streamlit)
2. **Enable HTTPS** (use reverse proxy like Nginx)
3. **Restrict API access** (firewall rules, API keys)
4. **Secure .env file** (`chmod 600`, never commit to Git)
5. **Regular backups** (database.db)
6. **Monitor logs** (archive/logs/)

Containment Safety

- **Whitelist critical IPs** before enabling real mode
- **Test in simulation mode** first
- **Review iptables rules** regularly
- **Have rollback plan** (unblock script)

🤝 Contributing

To Add Features

1. Create new detection rules in `detection_engine/detection_rules.py`
2. Integrate new threat intel sources in `server_backend/threat_intel.py`
3. Customize dashboard in `server_backend/dashboard.py`
4. Add tests in `tests/`
5. Update this README

📊 Project Statistics

- **Total Lines of Code**: ~3,500
- **Detection Rules**: 5
- **Threat Intel Sources**: 2
- **Database Fields**: 42
- **Test Coverage**: 9 test suites
- **Documentation**: Comprehensive

📝 License

This project is for **educational and security research purposes**.

📞 Support

For issues or questions: 1. Check the Troubleshooting section 2. Review test scripts in `tests/` 3. Check system logs in `server_backend/*.log`

🎉 Acknowledgments

Built with: - **Flask** - API backend - **Streamlit** - Dashboard - **SQLAlchemy** - Database ORM - **ip-api.com** - Geographic data - **AbuseIPDB** - IP reputation

---

Version: 3.0 (Enterprise Edition with Automation)
Last Updated: December 2025
Status: ✅ Production Ready with Configurable Automation

🚀 What's New in v3.0

- ✅ **Configurable Automation Policies** - Per-severity action controls
- ✅ **Dashboard Automation UI** - Manage policies from Settings
- ✅ **Action Logging** - Audit trail in automation.log
- ✅ **Statistics Tracking** - Monitor automated actions
- ✅ **Reports Directory** - Organized PDF report storage
- ✅ **Enhanced Documentation** - Comprehensive README
- ✅ **Improved File Structure** - Clean project organization