AI SOC Triage Assistant

> **AI-assisted Security Operations Center workflow for Log Summarization and Incident Triage**

🎯 Overview

The AI SOC Triage Assistant is a defensive security tool that automates the analysis of security logs to help Security Operations Center (SOC) analysts identify, triage, and respond to security incidents more efficiently.

Key Features

- 📥 **Log Ingestion**: Process raw security logs from multiple sources (auth, web, firewall, endpoint)
- 🔄 **Normalization**: Convert diverse log formats into a common schema
- 🔗 **Correlation**: Group related events into coherent incidents using rule-based detection
- 📊 **Triage Scoring**: Automated severity assessment with confidence levels
- 🗺️ **MITRE ATT&CK Mapping**: Map detected signals to MITRE techniques
- 📝 **Summarization**: Generate human-readable incident summaries (deterministic or LLM-powered)
- 📁 **Case Export**: Produce case files in JSON and Markdown formats
- 🌐 **API & UI**: FastAPI REST API and Streamlit web interface

> ⚠ **This is a defensive project**. It focuses on analysis, triage, and reporting—not exploitation.

🏷 Table of Contents

PROF

🚀 Installation

Prerequisites

- Python 3.9 or higher
- pip package manager

Setup

```
# Clone the repository
git clone https://github.com/ShadiBahaa/ai-soc-triage-assistant.git
cd ai-soc-triage-assistant

# Create virtual environment
python -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate

# Install dependencies
pip install -U pip
pip install -r requirements.txt

# Copy environment file
cp .env.example .env
```

⚡ Quick Start

Run the complete pipeline with a single command:

```
# Generate sample logs, normalize, correlate, and triage
python -m src.api.pipeline --generate

# Or run step by step:
python src/data/generate_sample_logs.py
python src/parsing/normalize.py
python src/correlation/correlate.py
python -m src.api.pipeline
```

View the generated case files in `docs/cases/`.

🏗️ Architecture

```
ai-soc-triage-assistant/
├── data/
│   ├── raw/                    # Raw input logs
│   └── processed/              # Normalized events and incidents
├── docs/
│   ├── cases/                  # Generated case files (JSON + MD)
│   └── schemas/                # Data schemas
```

```
├── src/
│   ├── data/                    # Log generation
│   ├── parsing/                 # Normalization
│   ├── correlation/             # Event correlation
│   ├── triage/                  # Scoring and summarization
│   ├── llm/                     # LLM integration
│   ├── api/                     # FastAPI application
│   ├── ui/                      # Streamlit interface
│   └── utils/                   # Case file writing
├── tests/                       # Unit tests
├── requirements.txt
└── README.md
```

Data Flow

```
Raw Logs → Normalize → Correlate → Triage → Summarize → Export
   ↓           ↓           ↓          ↓          ↓          ↓
.jsonl     Common      Incidents   Severity   Summary   JSON/MD
           Schema                  + MITRE             Case Files
```

📊 Log Sources

The system supports four log types:

Source	Description	Key Fields
auth	Authentication logs	user, src_ip, event_type, status
web	Web access logs	path, method, status_code, user_agent
fw	Firewall logs	src_ip, dst_port, action, protocol
endpoint	Process execution	process_name, command_line, parent_process

PROF

Normalized Event Schema

```
{
  "timestamp": "ISO-8601 string",
  "source": "auth|web|fw|endpoint|other",
  "host": "string",
  "user": "string|null",
  "src_ip": "string|null",
  "dst_ip": "string|null",
  "event_type": "string",
  "action": "string|null",
  "status": "success|failure|info|null",
  "severity_hint": "low|medium|high|critical|null",
```

```
  "raw": "object"
}
```

🔗 Correlation Strategy

Events are grouped into incidents using:

1. **Correlation Key**: `(host, user, src_ip)` tuple
2. **Time Window**: Events within configurable window (default: 60 minutes)
3. **Pattern Detection**: Rule-based signal identification

Detected Signals

Signal	Description	Threshold
`possible_bruteforce`	Multiple failed logins	≥10 failures
`bruteforce_then_success`	Success after failures	Failures + success
`suspicious_web_paths`	Access to sensitive paths	≥1 request to .env, etc.
`possible_port_scan`	Multiple blocked connections	≥5 denies
`suspicious_process_execution`	Known attack tools	mimikatz, psexec, etc.
`possible_lateral_movement`	Auth + suspicious process	Combined signals

📈 Triage Scoring

Severity Calculation

Severity is calculated based on signal weights:

Score	Severity
≥10	Critical
≥6	High
≥3	Medium
❤️	Low

PROF

MITRE ATT&CK Mapping

Signal	MITRE Technique	Tactic
Brute Force	T1110	Credential Access
Suspicious Web	T1190	Initial Access
Port Scan	T1046	Discovery

Signal	MITRE Technique	Tactic
Suspicious Process	T1059	Execution
Lateral Movement	T1021	Lateral Movement

Confidence Score

Confidence is calculated as: `0.5 + min(0.45, 0.08 × severity_score)`

🔧 Running the Pipeline

Command Line

```
# Full pipeline with log generation
python -m src.api.pipeline --generate

# Just triage (assumes logs already processed)
python -m src.api.pipeline

# Skip case file writing
python -m src.api.pipeline --no-cases

# Use LLM for summarization (requires configuration)
python -m src.api.pipeline --llm
```

Individual Steps

```
# Step 1: Generate sample logs
python src/data/generate_sample_logs.py

# Step 2: Normalize logs
python src/parsing/normalize.py

# Step 3: Correlate events
python src/correlation/correlate.py

# Step 4: Triage and export
python -m src.api.pipeline
```

PROF

🌐 API Reference

Start the API server:

```
uvicorn src.api.app:app --reload --port 8000
```

Endpoints

Method	Endpoint	Description
GET	`/`	API information
GET	`/health`	Health check
GET	`/run`	Run triage pipeline
POST	`/run/full`	Run complete pipeline
GET	`/incidents`	List all incidents
GET	`/incidents/{id}`	Get specific incident
GET	`/stats`	Pipeline statistics

Example Request

```
# Run the triage pipeline
curl "http://localhost:8000/run"

# Run full pipeline with log generation
curl -X POST "http://localhost:8000/run/full?generate_logs=true"

# Get incidents filtered by severity
curl "http://localhost:8000/incidents?severity=high"
```

Swagger Documentation

Visit `http://localhost:8000/docs` for interactive API documentation.

🖥️ Web Interface

PROF

Start the Streamlit UI:

```
streamlit run src/ui/app.py
```

Features:

- **Dashboard**: Overview with metrics and charts
- **Pipeline**: Run analysis with options
- **Incidents**: Browse and filter incidents
- **Events**: View normalized events
- **Settings**: Configuration information

🤖 LLM Integration

The system supports optional LLM-powered summarization:

Configuration

Edit `.env`:

```
LLM_PROVIDER=openai    # Options: openai, anthropic, local, none
LLM_API_KEY=your-key
LLM_MODEL=gpt-4o-mini  # Optional: specific model
```

Supported Providers

Provider	Models	Requirements
OpenAI	gpt-4o, gpt-4o-mini, etc.	`pip install openai`
Anthropic	claude-3-*, etc.	`pip install anthropic`
Local	Ollama models	Ollama server running

Without LLM

When `LLM_PROVIDER=none`, the system uses deterministic summarization that generates comprehensive summaries without external API calls.

📄 Example Output

Case File (Markdown)

```
# INC-1234567

> **Type:** Credential Compromise
> **Generated:** 2024-01-15 10:30:00 UTC

## Status: 🔵 HIGH

- **Confidence:** 82%
- **Severity Score:** 7.5
- **⚠ REQUIRES IMMEDIATE ATTENTION**

## Summary

An attacker from 198.51.100.10 attempted to gain access to the account
'alice' on srv-web-01 through brute force, generating 30 failed login
attempts. The brute force attack was successful...

## MITRE ATT&CK Mapping

| Technique | Name | Tactic |
```

```
|-----------|------|--------|
| [T1110](https://attack.mitre.org/techniques/T1110/) | Brute Force |
Credential Access |

## Recommended Actions

1. 🔐 Check account lockout and MFA status for impacted user
2. 🔍 Review authentication logs for the same source IP
3. 🚫 Consider blocking the suspicious source IP
```

Case File (JSON)


```
{
  "meta": {
    "generated_at": "2024-01-15T10:30:00Z",
    "version": "1.0",
    "tool": "AI SOC Triage Assistant"
  },
  "incident": {
    "incident_id": "INC-1234567",
    "incident_type": "Credential Compromise",
    "key": {
      "host": "srv-web-01",
      "user": "alice",
      "src_ip": "198.51.100.10"
    },
    "signals": [...],
    "event_count": 36
  },
  "triage": {
    "severity": "high",
    "confidence": 0.82,
    "mitre": [...]
  },
  "summary": "..."
}
```


PROF

Ethics & Privacy

Synthetic Data

All sample logs in this project are **synthetically generated** and do not contain real security events or personal information. The generation script creates realistic-but-fake data suitable for:

- Testing and development
- Portfolio demonstration
- Educational purposes

Production Use

When using with real logs:

1. **Redact PII** before processing
2. **Follow compliance** requirements (GDPR, HIPAA, etc.)
3. **Secure storage** for case files
4. **Access controls** on the API/UI
5. **Audit logging** for accountability

Responsible Disclosure

This tool is for **defensive purposes only**. Do not use it to:

- Attack systems without authorization
- Process data you don't have rights to
- Bypass security controls

🧪 Running Tests

```
# Run all tests
python -m pytest tests/

# Run with coverage
python -m pytest tests/ --cov=src
```

🤝 Contributing

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/amazing-feature`
3. Commit changes: `git commit -m 'Add amazing feature'`
4. Push to branch: `git push origin feature/amazing-feature`
5. Open a Pull Request

PROF

📜 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- MITRE ATT&CK® for the threat framework
- The security community for research and best practices

Built for portfolio demonstration and educational purposes.