CyberDucky Mini SIEM 🦆🔒

A SOC Analyst-Focused Security Information and Event Management System

CyberDucky Mini SIEM is a full-stack web application designed specifically for Security Operations Center (SOC) analysts to analyze Zscaler NSS Web Logs with advanced threat detection, anomaly analysis, and AI-powered insights.

📋 Table of Contents

🎯 Overview

CyberDucky Mini SIEM provides SOC analysts with a powerful platform to:

- **Upload and parse** Zscaler NSS Web Logs (CSV format)
- **Detect threats** using multiple detection methods (rule-based, statistical, AI-powered)
- **Analyze anomalies** with 4 core statistical detection algorithms
- **Visualize data** with 7+ interactive chart types
- **Investigate incidents** with unified analysis across all log files
- **Track metrics** with real-time dashboards and risk scoring

Target Users

- **SOC Analysts** - Primary users who need to investigate security incidents
- **Security Engineers** - Configure detection rules and thresholds
- **Incident Responders** - Investigate and respond to threats

---

✨ Key Features

🔍 Multi-Method Threat Detection

1. **Rule-Based Detection**

- Malware detection (malware, virus, trojan, ransomware)
- Phishing detection (phishing, credential harvesting)
- C2 beaconing detection (command-and-control patterns)
- Data exfiltration detection (large uploads, suspicious file transfers)

2. **Statistical Anomaly Detection** (4 Core Methods)

    - **Z-Score Analysis** - Detects outliers using 3-sigma rule (rate anomalies, unusual behavior)
    - **Percentile-Based Detection** - Identifies top 1% anomalies (data exfiltration, large uploads)
    - **EWMA (Exponentially Weighted Moving Average)** - Detects trend deviations (persistent threats)
    - **Burst Detection** - Identifies sudden spikes using rolling statistics (DDoS, brute force attacks)

3. **AI-Powered Analysis**

    - Local LLM integration (Ollama with phi3:mini)
    - Context-aware threat assessment
    - Natural language threat descriptions
    - Confidence scoring

📊 Advanced Visualizations

- **Anomaly Time Series** - Track anomalies over time
- **Risk Score Trendline** - Monitor risk trends with EWMA
- **Event Timeline** - Chronological event visualization
- **Requests Per Minute** - Traffic pattern analysis with burst detection
- **Category Distribution** - URL category breakdown
- **Top Threats** - Most frequent threats
- **User Activity Heatmap** - Activity patterns by user and time

🎛️ SOC Analyst Dashboard

- **Overview Dashboard** - Aggregated metrics across all log files

    - Total log files, entries, anomalies, threats
    - Top risky users, IPs, and threats
    - Anomaly trends over time
    - Advanced analytics section

- **Unified Analysis** - Drill-down investigation

    - Filter by username, IP, threat name, category, risk score
    - View all matching entries across all files
    - File breakdown showing data sources
    - Anomaly and log entry tables

- **File Analysis** - Individual file deep-dive

    - File-specific metrics and statistics
    - All visualizations for single file
    - Export capabilities

🔐 Security Features

- JWT-based authentication
- User isolation (users only see their own data)
- Secure file upload with validation
- SQL injection prevention (parameterized queries)
- XSS protection (React auto-escaping)

🏗 Architecture

High-Level Architecture

```
┌──────────────────────────────────────────────────────────────┐
│                          Frontend                             │
│  React + TypeScript + Vite + Tailwind CSS                    │
│  - Overview Dashboard (aggregated metrics)                    │
│  - Unified Analysis (cross-file investigation)               │
│  - File Analysis (single file deep-dive)                     │
│  - Upload Logs                                               │
└──────────────────────────────────────────────────────────────┘
                      │ HTTP/REST API
                      │ (JWT Authentication)
┌─────────────────────▼────────────────────────────────────────┐
│                          Backend                              │
│  Flask + SQLAlchemy + PostgreSQL                             │
│  ┌─────────────────────────────────────────────────────────┐ │
│  │ Controllers (API Endpoints)                             │ │
│  │  - Auth, Upload, Dashboard, Analysis, Visualization │ │
│  └─────────────────────────────────────────────────────────┘ │
│  ┌──────────────────────▼──────────────────────────────────┐ │
│  │ Services (Business Logic)                               │ │
│  │  - Log Parser Service                                   │ │
│  │  - Anomaly Detection Service (Rule + Statistical)       │ │
│  │  - Statistical Analysis Service                         │ │
│  │  - Visualization Data Service                           │ │
│  │  - LLM Service (Ollama Integration)                     │ │
│  └─────────────────────────────────────────────────────────┘ │
│  ┌──────────────────────▼──────────────────────────────────┐ │
│  │ Repositories (Data Access)                              │ │
│  │  - User, LogFile, LogEntry, Anomaly                     │ │
│  └─────────────────────────────────────────────────────────┘ │
│  ┌──────────────────────▼──────────────────────────────────┐ │
│  │ Models (Database Schema)                                │ │
│  │  - User, LogFile, LogEntry, Anomaly                     │ │
│  └─────────────────────────────────────────────────────────┘ │
└──────────────────────────────────────────────────────────────┘
                      │
                      │
         ┌────────────┴─────────────┬──────────────┐
         │                          │              │
┌────────▼────────┐  ┌──────────────▼─────┐  ┌─────▼──────┐
│   PostgreSQL    │  │       Ollama       │  │  Uploads   │
```

```
│    Database      │    │   (LLM Service)    │    │   Volume     │
└──────────────────┘    └────────────────────┘    └──────────────┘
```

Design Patterns

- **MVC (Model-View-Controller)** - Separation of concerns
- **Repository Pattern** - Data access abstraction
- **Service Layer Pattern** - Business logic encapsulation
- **Strategy Pattern** - Extensible log parsers
- **Factory Pattern** - Parser selection and instantiation

��� Technology Stack

Backend

Technology	Version	Purpose
Python	3.11+	Programming language
Flask	3.0+	Web framework
PostgreSQL	15+	Relational database
SQLAlchemy	2.0+	ORM
Flask-JWT-Extended	4.5+	Authentication
NumPy	1.24+	Statistical computing
SciPy	1.11+	Scientific computing
Ollama	Latest	Local LLM inference

Frontend

Technology	Version	Purpose
React	18+	UI framework
TypeScript	5+	Type safety
Vite	5+	Build tool
TailwindCSS	3+	Styling
Recharts	2+	Data visualization
Axios	1+	HTTP client
React Router	6+	Routing
Lucide React	Latest	Icons

Infrastructure

Technology	Purpose
Docker	Containerization
Docker Compose	Multi-container orchestration
Nginx	Reverse proxy (production)

��� Quick Start

Prerequisites

- **Docker** and **Docker Compose** installed
- **Ollama** installed (for AI features)
- **8GB RAM** minimum
- **10GB disk space** for logs and database

Installation

1. **Clone the repository**

   ```
   git clone <repository-url>
   cd CyberDuckyMiniSIEM
   ```

2. **Start Ollama and pull the model**

   ```
   ollama serve
   ollama pull phi3:mini
   ```

3. **Start the application**

   Windows:

   ```
   .\start-docker.ps1
   ```

   Linux/Mac:

   ```
   docker-compose up -d
   ```

4. **Access the application**

   - Frontend: http://localhost:5173
   - Backend API: http://localhost:5000
   - Ollama: http://localhost:11434

5. **Create an account**

    - Navigate to http://localhost:5173
    - Click "Register" and create an account
    - Login with your credentials

6. **Upload logs**

    - Click "Upload Logs" in the navigation
    - Select "Zscaler NSS Web Logs"
    - Upload a CSV file from `backend/sample_data/`
    - Wait for processing to complete

7. **View analysis**

    - Navigate to "Overview Dashboard"
    - Explore metrics, visualizations, and anomalies
    - Click on users/IPs/threats for unified analysis

Sample Data

Sample Zscaler logs are provided in `backend/sample_data/` directory:

- `comprehensive_zscaler_sample.csv` - Comprehensive sample with all detection methods triggered

See Sample Data Guide for details.

��� How It Works

1. Log Upload and Parsing

User uploads a Zscaler CSV file:

```
User → Frontend → Backend API → Parser Factory → Zscaler Parser
```

Zscaler Parser Process:

1. **Validate CSV format** - Check headers match Zscaler NSS format
2. **Parse each row** - Extract 35 fields per log entry
3. **Normalize data** - Convert timestamps, IP addresses, byte counts
4. **Calculate risk score** - Based on threat category, action, URL category
5. **Save to database** - Store in `log_entries` table

Field Mapping Example:

Zscaler Field	Database Field	Type	Description
`datetime`	`timestamp`	DateTime	Event timestamp
`login`	`username`	String	User login

Zscaler Field	Database Field	Type	Description
`csip`	`source_ip`	String	Client source IP
`cip`	`destination_ip`	String	Destination IP
`urlcategory`	`url_category`	String	URL category
`threatname`	`threat_name`	String	Threat signature
`action`	`action`	String	Allowed/Blocked
`requestsize`	`source_bytes`	Integer	Upload size
`responsesize`	`destination_bytes`	Integer	Download size

See Parser Guide for complete field mapping.

---

2. Anomaly Detection Pipeline

Three-Stage Detection Process:

```
Log Entries → Stage 1: Rule-Based → Stage 2: Statistical → Stage 3: LLM →
Anomalies
```

Stage 1: Rule-Based Detection

Fast, high-confidence detection of known threats:

```
# Malware Detection
if 'malware' in threat_name.lower() or 'virus' in threat_name.lower():
    create_anomaly(
        type='malware_detected',
        severity='critical',
        confidence=0.95
    )

# Phishing Detection
if 'phishing' in url_category.lower():
    create_anomaly(
        type='phishing_attempt',
        severity='critical',
        confidence=0.90
    )

# Data Exfiltration
if source_bytes > 100_000_000:  # 100 MB upload
    create_anomaly(
        type='data_exfiltration',
        severity='high',
```

```
        confidence=0.85
    )
```

Stage 2: Statistical Analysis

Applies 4 core statistical methods to detect anomalies:

1. Z-Score Analysis (Outlier Detection)

Detects values that deviate significantly from the mean using the 3-sigma rule.

```
z_score = (value - mean) / std_dev
if abs(z_score) > 3.0:
    anomaly_detected()
```

Use Cases:

- Unusual request rates (user making 100 requests/min when average is 10)
- Abnormal risk scores
- Outlier detection in any metric

Example:

```
# Detect unusual request rate
user_requests_per_minute = [10, 12, 11, 9, 150, 10, 11]
mean = 30.4
std_dev = 52.3
z_score_for_150 = (150 - 30.4) / 52.3 = 2.29  # Not anomaly (< 3.0)

# But if we have more normal data:
user_requests_per_minute = [10, 12, 11, 9, 10, 11, 150]
mean = 30.4
std_dev = 49.8
z_score_for_150 = (150 - 30.4) / 49.8 = 2.40  # Still not anomaly

# With proper baseline:
normal_rpm = [10, 12, 11, 9, 10, 11, 12, 10, 11, 150]
mean = 20.6
std_dev = 41.8
z_score_for_150 = (150 - 20.6) / 41.8 = 3.09  # ANOMALY!
```

2. Percentile-Based Detection (Top 1% Threshold)

Identifies extreme values in the top or bottom percentiles.

```
threshold_99th = np.percentile(values, 99)
if value > threshold_99th:
    anomaly_detected()
```

Use Cases:

- Data exfiltration (top 1% of upload sizes)
- Large downloads
- Extreme risk scores

Example:

```
# Upload sizes in bytes
upload_sizes = [1000, 2000, 1500, 3000, 2500, 50000000]
threshold_99th = 49,505,000  # 99th percentile
# 50MB upload exceeds threshold → ANOMALY!
```

3. EWMA - Exponentially Weighted Moving Average (Trend Detection)

Detects deviations from expected trends by giving more weight to recent values.

```
ewma = alpha * current_value + (1 - alpha) * previous_ewma
deviation = abs(current_value - ewma)
if deviation > threshold * std_dev:
    anomaly_detected()
```

Use Cases:

- Persistent high risk (user's risk score trending upward)
- Gradual behavior changes
- Slow attacks that build over time

Example:

```
# User risk scores over time
risk_scores = [10, 12, 15, 18, 22, 70]  # Gradual increase then spike
alpha = 0.3  # Weight for current value

ewma_values = []
ewma = risk_scores[0]  # Start with first value

for score in risk_scores[1:]:
    ewma = 0.3 * score + 0.7 * ewma
    ewma_values.append(ewma)

# ewma_values = [10.6, 11.92, 13.54, 16.08, 32.26]
# At score=70: deviation = |70 - 32.26| = 37.74 → ANOMALY!
```

4. Burst Detection (Rolling Statistics)

Detects sudden spikes in activity using sliding window analysis.

```
window_values = values[i-window_size:i]
window_mean = np.mean(window_values)
window_std = np.std(window_values)
z_score = (current_value - window_mean) / window_std

if z_score > threshold_sigma:
    burst_detected()
```

Use Cases:

- DDoS attacks (sudden spike in requests)
- Brute force attempts (rapid failed logins)
- Port scanning (burst of connections)
- Blocked request spikes

Example:

```
# Blocked requests per minute
blocked_per_minute = [2, 3, 2, 1, 3, 2, 45, 3, 2]
window_size = 5
threshold_sigma = 2.0

# At index 6 (value=45):
window = [2, 3, 2, 1, 3]  # Previous 5 values
window_mean = 2.2
window_std = 0.75
z_score = (45 - 2.2) / 0.75 = 57.07  # BURST DETECTED!
```

Stage 3: LLM Analysis

For high-severity anomalies, the system uses a local LLM (Ollama + phi3:mini) to provide context-aware analysis:

```
# LLM Prompt
prompt = f"""
Analyze this security anomaly:
- User: {username}
- Anomaly Type: {anomaly_type}
- Risk Score: {risk_score}
- Context: {log_entry_details}

Provide:
1. Threat assessment
2. Recommended actions
3. Urgency level
"""
```

```
llm_response = ollama.generate(model='phi3:mini', prompt=prompt)
```

LLM Output Example:

```
Threat Assessment: User john.doe accessed a known phishing site and
attempted to download malware. This is a critical security incident.

Recommended Actions:
1. Immediately isolate the user's device
2. Force password reset
3. Scan device for malware
4. Review user's recent activity

Urgency: CRITICAL - Respond within 15 minutes
```

---

3. Visualization Generation

The system generates 7 types of visualizations:

1. **Anomaly Time Series** - Anomaly count over time
2. **Risk Score Trendline** - Risk trends with EWMA
3. **Event Timeline** - Events grouped by time buckets
4. **Requests Per Minute** - Traffic patterns with burst detection
5. **Category Distribution** - URL categories (pie/bar chart)
6. **Top Threats** - Most frequent threats
7. **User Activity Heatmap** - Activity by user and time

See SOC Analyst Guide for visualization details.

---

4. Unified Analysis

When a SOC analyst clicks on a user, IP, or threat in the Overview Dashboard:

```
Click on "john.doe" → Navigate to /unified-analysis?username=john.doe
                    → Backend filters ALL log entries across ALL files
                    → Returns: entries, anomalies, statistics, file breakdown
                    → Frontend displays unified view
```

Benefits:

- See all activity for a user across multiple log files
- Identify patterns across time
- Correlate anomalies
- Complete investigation context

��� Design Decisions

Why These 4 Statistical Methods?

1. Z-Score Analysis ★★★★★

- **Chosen because:** Industry standard, easy to understand (3-sigma rule), works well for outlier detection
- **SOC Value:** Analysts understand "3 standard deviations from normal"
- **Use Case:** Rate anomalies, unusual behavior

2. Percentile-Based ★★★★★

- **Chosen because:** Simple threshold (top 1%), directly identifies extreme values
- **SOC Value:** Clear cutoff for "large" uploads/downloads
- **Use Case:** Data exfiltration detection

3. EWMA ★★★★

- **Chosen because:** Detects trends and gradual changes, complements Z-score
- **SOC Value:** Catches slow attacks that build over time
- **Use Case:** Persistent threats, behavior drift

4. Burst Detection ★★★★★

- **Chosen because:** Critical for attack detection (DDoS, brute force)
- **SOC Value:** Immediate visibility into sudden spikes
- **Use Case:** Attack patterns, scanning activity

Methods NOT Included:

- **IQR (Interquartile Range)** - Redundant with Z-score, only used for boxplot visualization
- **Pearson Correlation** - Too complex, limited SOC value, not used in detection pipeline
- **KDE (Kernel Density Estimation)** - Academic, not actionable for SOC analysts

Why Local LLM (Ollama)?

- **Privacy:** Sensitive security data never leaves your infrastructure
- **Cost:** No API fees, unlimited usage
- **Speed:** Local inference is fast enough for batch processing
- **Offline:** Works without internet connection

Why PostgreSQL?

- **ACID Compliance:** Critical for security data integrity
- **JSON Support:** Flexible for storing anomaly metadata
- **Performance:** Handles millions of log entries efficiently
- **UUID Primary Keys:** Distributed-friendly, no collision risk

Why React + TypeScript?

- **Type Safety:** Catch errors at compile time

- **Developer Experience:** Excellent tooling and IDE support
- **Performance:** Virtual DOM for efficient updates
- **Ecosystem:** Rich library ecosystem (Recharts, React Router)

��� Parser Architecture

Overview

The parser architecture is designed for **extensibility** - easily add new log sources (CrowdStrike, Okta, AWS, etc.) without modifying existing code.

Components

1. BaseParser (Abstract Class)

```
class BaseParser(ABC):
    """Abstract base class for all log parsers"""

    @abstractmethod
    def parse_line(self, line: str, line_number: int) -> Optional[Dict[str, Any]]:
        """Parse a single log line into a dictionary"""
        pass

    @abstractmethod
    def detect_format(self, file_path: str) -> bool:
        """Detect if this parser can handle the file"""
        pass
```

2. ZscalerParser (Concrete Implementation)

Parses Zscaler NSS Web Logs (CSV format with 35 fields).

3. ParserFactory

```
class ParserFactory:
    """Factory for creating appropriate parser instances"""

    _parsers = {
        'zscaler': ZscalerParser,
        # Future: 'crowdstrike': CrowdStrikeParser,
        # Future: 'okta': OktaParser,
    }

    @classmethod
    def get_parser(cls, log_type: str = None, file_path: str = None) ->
BaseParser:
        """Get parser by type or auto-detect from file"""
        if log_type:
            return cls._parsers[log_type]()
```

```
        # Auto-detection
        for parser_class in cls._parsers.values():
            parser = parser_class()
            if parser.detect_format(file_path):
                return parser

        raise ValueError("No suitable parser found")
```

Zscaler Field Mapping

Zscaler Field	Normalized Field	Description
`time`	`timestamp`	Event timestamp
`login`	`username`	User login name
`sip`	`source_ip`	Source IP address
`dip`	`destination_ip`	Destination IP address
`url`	`url`	Requested URL
`urlcat`	`url_category`	URL category
`threatname`	`threat_name`	Detected threat
`risk`	`risk_score`	Risk score (0-100)
`action`	`action`	Action taken (allowed/blocked)
`reqsize`	`bytes_sent`	Request size in bytes
`respsize`	`bytes_received`	Response size in bytes
`devicehostname`	`device_name`	Device hostname
`location`	`location`	User location
`dept`	`department`	User department

See Parser Guide for complete field mapping and how to add new parsers.

��� Anomaly Detection

Detection Methods

1. Rule-Based Detection

Fast, high-confidence detection of known threats:

```
# Malware Detection
if 'malware' in threat_name.lower():
```

```python
    create_anomaly(
        type='malware_detected',
        severity='critical',
        confidence=0.95
    )

# Phishing Detection
if 'phishing' in url_category.lower():
    create_anomaly(
        type='phishing_attempt',
        severity='critical',
        confidence=0.90
    )

# C2 Beaconing
if 'command-and-control' in threat_category:
    create_anomaly(
        type='c2_communication',
        severity='critical',
        confidence=0.92
    )

# Data Exfiltration
if source_bytes > 100_000_000:  # 100 MB
    create_anomaly(
        type='data_exfiltration',
        severity='high',
        confidence=0.85
    )
```

2. Statistical Detection

Z-Score Analysis:

```python
def detect_zscore_anomalies(data, threshold=3.0):
    mean = np.mean(data)
    std_dev = np.std(data)
    anomalies = []

    for i, value in enumerate(data):
        z_score = (value - mean) / std_dev
        if abs(z_score) > threshold:
            anomalies.append({
                'index': i,
                'value': value,
                'z_score': z_score,
                'mean': mean,
                'std_dev': std_dev
            })

    return anomalies
```

EWMA (Exponentially Weighted Moving Average):

```
def detect_ewma_anomalies(data, alpha=0.3, threshold=2.0):
    ewma = data[0]
    anomalies = []

    for i, value in enumerate(data[1:], 1):
        deviation = abs(value - ewma)
        std = np.std(data[:i])

        if deviation > threshold * std:
            anomalies.append({
                'index': i,
                'value': value,
                'expected': ewma,
                'deviation': deviation
            })

        ewma = alpha * value + (1 - alpha) * ewma

    return anomalies
```

Percentile-Based Detection:

```
def detect_percentile_anomalies(data, percentile=99):
    threshold = np.percentile(data, percentile)
    anomalies = []

    for i, value in enumerate(data):
        if value > threshold:
            anomalies.append({
                'index': i,
                'value': value,
                'threshold': threshold,
                'percentile': percentile
            })

    return anomalies
```

Burst Detection:

```
def detect_burst(data, window=10, threshold_sigma=2.0):
    anomalies = []

    for i in range(window, len(data)):
        window_data = data[i-window:i]
        window_mean = np.mean(window_data)
```

```
        window_std = np.std(window_data)

        z_score = (data[i] - window_mean) / window_std

        if z_score > threshold_sigma:
            anomalies.append({
                'index': i,
                'value': data[i],
                'window_mean': window_mean,
                'z_score': z_score
            })

    return anomalies
```

3. LLM-Based Detection

Context-aware analysis using local LLM:

```
def analyze_with_llm(anomaly, log_entry):
    prompt = f"""
    Analyze this security anomaly:

    User: {log_entry.username}
    IP: {log_entry.source_ip}
    Anomaly Type: {anomaly.anomaly_type}
    Risk Score: {log_entry.risk_score}
    URL: {log_entry.url}
    Threat: {log_entry.threat_name}
    Action: {log_entry.action}

    Provide:
    1. Threat assessment (1-2 sentences)
    2. Recommended actions (3-5 bullet points)
    3. Urgency level (LOW/MEDIUM/HIGH/CRITICAL)
    """

    response = ollama.generate(
        model='phi3:mini',
        prompt=prompt,
        temperature=0.3
    )

    return response['response']
```

See Anomaly Detection Guide for detailed explanations.

��� API Reference

Authentication

Register

```
POST /api/auth/register
Content-Type: application/json

{
  "username": "analyst1",
  "email": "analyst1@company.com",
  "password": "SecurePass123!"
}

Response: 201 Created
{
  "message": "User created successfully",
  "user": {
    "id": "uuid",
    "username": "analyst1",
    "email": "analyst1@company.com"
  }
}
```

Login

```
POST /api/auth/login
Content-Type: application/json

{
  "username": "analyst1",
  "password": "SecurePass123!"
}

Response: 200 OK
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGc...",
  "user": {
    "id": "uuid",
    "username": "analyst1",
    "email": "analyst1@company.com"
  }
}
```

Log Upload

Upload Log File

```
POST /api/upload
Authorization: Bearer <token>
Content-Type: multipart/form-data
```

```
file: <binary>
log_type: zscaler

Response: 200 OK
{
  "message": "File uploaded and processed successfully",
  "log_file": {
    "id": "uuid",
    "original_filename": "zscaler_logs.csv",
    "status": "completed",
    "total_entries": 1500,
    "anomaly_count": 45,
    "threat_count": 12
  }
}
```

Dashboard

Get Overview

```
GET /api/dashboard/overview
Authorization: Bearer <token>

Response: 200 OK
{
  "total_files": 5,
  "total_entries": 7500,
  "total_anomalies": 230,
  "critical_anomalies": 45,
  "avg_risk_score": 42.5,
  "high_risk_entries": 450,
  "unique_users": 125,
  "unique_ips": 89,
  "threat_count": 67
}
```

Get Unified Analysis

```
GET /api/dashboard/unified-analysis?username=john.doe&min_risk=70
Authorization: Bearer <token>

Response: 200 OK
{
  "log_entries": [...],
  "anomalies": [...],
  "statistics": {
    "total_count": 1500,
    "avg_risk_score": 75.5,
```

```
    "high_risk_count": 450,
    "anomaly_count": 45
  }
}
```

Visualizations

Get All Visualizations

```
GET /api/visualization/all-visualizations/<file_id>
Authorization: Bearer <token>

Response: 200 OK
{
  "anomaly_time_series": [...],
  "risk_trendline": [...],
  "event_timeline": [...],
  "requests_per_minute": [...]
}
```

��� Development

Project Structure

```
CyberDuckyMiniSIEM/
├── backend/
│   ├── app/
│   │   ├── __init__.py             # Flask app factory
│   │   ├── config.py               # Configuration
│   │   ├── models/                 # SQLAlchemy models
│   │   │   ├── user.py
│   │   │   ├── log_file.py
│   │   │   ├── log_entry.py
│   │   │   └── anomaly.py
│   │   ├── repositories/           # Data access layer
│   │   │   ├── user_repository.py
│   │   │   ├── log_file_repository.py
│   │   │   ├── log_entry_repository.py
│   │   │   └── anomaly_repository.py
│   │   ├── services/               # Business logic
│   │   │   ├── log_processing_service.py
│   │   │   ├── anomaly_detection_service.py
│   │   │   ├── statistical_analysis_service.py
│   │   │   ├── visualization_data_service.py
│   │   │   └── llm_service.py
│   │   ├── parsers/                # Log parsers
│   │   │   ├── base_parser.py
│   │   │   ├── zscaler_parser.py
```

```
│   │   │   └── parser_factory.py
│   │   └── controllers/             # API endpoints
│   │       ├── auth_controller.py
│   │       ├── dashboard_controller.py
│   │       ├── analysis_controller.py
│   │       ├── upload_controller.py
│   │       └── visualization_controller.py
│   ├── sample_data/                 # Sample logs
│   ├── requirements.txt             # Python dependencies
│   ├── Dockerfile
│   └── run.py                       # Application entry point
├── frontend/
│   ├── src/
│   │   ├── components/              # React components
│   │   │   ├── MetricsCard.tsx
│   │   │   ├── MetricsOverview.tsx
│   │   │   ├── DataTableModal.tsx
│   │   │   ├── LogEntryDetails.tsx
│   │   │   ├── NavigationBar.tsx
│   │   │   └── VisualizationWidgets.tsx
│   │   ├── pages/                   # Page components
│   │   │   ├── Login.tsx
│   │   │   ├── Register.tsx
│   │   │   ├── OverviewDashboard.tsx
│   │   │   ├── UnifiedAnalysis.tsx
│   │   │   ├── Analysis.tsx
│   │   │   └── UploadLogs.tsx
│   │   ├── services/                # API services
│   │   │   └── api.ts
│   │   ├── types/                   # TypeScript types
│   │   │   └── index.ts
│   │   ├── App.tsx                  # Main app component
│   │   └── main.tsx                 # Entry point
│   ├── package.json                 # Node dependencies
│   ├── Dockerfile
│   └── vite.config.ts               # Vite configuration
├── documentation/                   # Documentation
│   ├── README.md                    # Documentation index
│   ├── architecture/
│   │   └── SYSTEM_ARCHITECTURE.md
│   ├── guides/
│   │   ├── SOC_ANALYST_GUIDE.md
│   │   ├── PARSER_GUIDE.md
│   │   ├── ANOMALY_DETECTION.md
│   │   └── SAMPLE_DATA.md
│   └── deployment/
│       └── DOCKER_DEPLOYMENT.md
├── docker-compose.yml               # Docker orchestration
├── start-docker.ps1                 # Windows startup script
├── DATABASE_SCHEMA.sql              # Database schema
└── README.md                        # This file
```

Local Development

Backend:

```
cd backend
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt
flask run
```

Frontend:

```
cd frontend
npm install
npm run dev
```

Database:

```
docker run -d \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=postgres \
  -e POSTGRES_DB=cyberducky_siem \
  -p 5432:5432 \
  postgres:15-alpine
```

Running Tests

```
# Backend tests
cd backend
pytest

# Frontend tests
cd frontend
npm test
```

Environment Variables

Backend (.env):

```
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/cyberducky_siem
JWT_SECRET_KEY=your-secret-key-change-in-production
OLLAMA_URL=http://localhost:11434
OLLAMA_DEFAULT_MODEL=phi3:mini
```

```
LLM_ENABLED=true
FLASK_ENV=development
```

Frontend (.env):

```
VITE_API_URL=http://localhost:5000/api
```

��� Documentation

Available Guides

- **System Architecture** - System design and patterns
- **SOC Analyst Guide** - Quick reference for analysts
- **Parser Guide** - How to add new log sources
- **Anomaly Detection** - Detection methods explained
- **Sample Data** - Sample data guide
- **Docker Deployment** - Deployment guide

Quick Links

For SOC Analysts:

1. Getting Started
2. Investigation Workflows
3. Anomaly Types

For Developers:

1. System Architecture
2. Adding New Parsers
3. Development Setup

For System Administrators:

1. Docker Deployment
2. Environment Variables
3. Troubleshooting

��� Security Considerations

Authentication & Authorization

- **JWT Tokens** - Secure, stateless authentication
- **Password Hashing** - Werkzeug secure password storage
- **User Isolation** - Users only see their own data
- **Token Expiration** - Configurable token lifetime

Data Protection

- **SQL Injection Prevention** - Parameterized queries via SQLAlchemy
- **XSS Protection** - React auto-escaping
- **CORS Configuration** - Restricted cross-origin access
- **File Upload Validation** - Type and size checks

Privacy

- **Local LLM** - Sensitive data never leaves your infrastructure
- **No External APIs** - All processing done locally
- **User Data Isolation** - Multi-tenant security

��� License

This project is for educational and internal use. See LICENSE file for details.

��� Acknowledgments

- **Zscaler** - For NSS Web Log format documentation
- **Ollama** - For local LLM inference
- **Flask** - For the excellent web framework
- **React** - For the powerful UI library

��� Support

For questions or issues:

1. Check the documentation
2. Review the SOC Analyst Guide
3. Check the troubleshooting guide

Built with 💜 for SOC Analysts