SOaC Framework - Phase 2A Analysis Summary

React UI Development for Device Integration and Configuration

Date: November 13, 2025
Version: 1.0
Purpose: Comprehensive analysis to guide Phase 2A React UI development

---

Table of Contents

---

1. Executive Summary

Current State

The SOaC Framework Phase 1 has successfully implemented a **CLI-based security operations framework** with the following core components:

- ✅ **Correlation Engine**: Multi-source event correlation
- ✅ **Use Case Manager**: 10 security use cases with MAGMA framework
- ✅ **SOAR Playbook Manager**: 6 automated response playbooks
- ✅ **Threat Intelligence Module**: Pre-loaded threat actor profiles
- ✅ **Configuration Processor**: Excel-based rule loading for EntraID and PaloAlto

Phase 2A Objectives

Build a **React-based web UI** that enables:

1. **Device Integration Management**: Configure and manage PaloAlto NGFW, EntraID, and SIEM integrations
2. **Rule Configuration**: Visual interface for creating and managing detection rules
3. **Dashboard & Monitoring**: Real-time visibility of threats, incidents, and playbook executions

4. **Use Case Management**: UI for managing the lifecycle of security use cases

Key Findings

🔴 **Critical Gap**: No REST API layer exists - all components are Python modules called directly
🟡 **Authentication Gap**: No user authentication or authorization system implemented
🟢 **Strong Foundation**: Excellent data structures and operational models ready for UI integration

---

2. Current Backend Analysis

2.1 Project Structure

```
soac-framework/
├── app.py                          # Main CLI entry point
├── src/
│   ├── correlation_engine.py      # Event correlation logic
│   ├── config_processor.py        # Excel/JSON config processing
│   ├── use_case_manager.py        # Use case lifecycle management
│   ├── soar_playbooks.py          # Automated response playbooks
│   └── threat_intelligence.py     # Threat actor profiles & IOCs
├── config/
│   ├── config_template.json       # Main configuration
│   ├── entraid_rules.json         # Generated from Excel
│   └── paloalto_rules.json        # Generated from Excel
├── data/
│   ├── EntraID_Authentication_Rules.xlsx
│   └── PaloAlto_NGFW_Rules.xlsx
├── docs/
│   ├── FRAMEWORK_OVERVIEW.md
│   ├── USE_CASES.md
│   └── THREAT_LANDSCAPE.md
└── tests/
    ├── test_correlation_engine.py
    └── test_use_case_manager.py
```

2.2 Core Components Analysis

app.py - Main Application

- **Purpose**: CLI orchestrator for all framework components
- **Functionality**:
- Loads configuration from `config_template.json`
- Initializes all modules (correlation, use cases, SOAR, threat intel)
- Processes configuration files (EntraID, PaloAlto)
- Runs demo scenarios
- No web server or API endpoints

Key Methods:

```
- __init__(config_file)        # Initialize framework
- process_configurations()     # Load device rules
- generate_reports()           # Create coverage reports
- run_demo()                   # Demonstrate capabilities
```

src/correlation_engine.py - Event Correlation

- **Purpose**: Multi-source event correlation based on tactical patterns
- **Patterns Supported**: R1 (Ransomware), D1 (Data Exfil), C1 (Credential), IN1 (Intrusion)
- **Core Logic**:
- Groups events by entity (UserName, ComputerName, IP)
- Matches events against pattern phases (MITRE ATT&CK tactics)
- Creates incidents when ≥3 phases match
- Calculates confidence levels (Low/Medium/High/Critical)

Key Methods:

```
- correlate_events(events, pattern_id, time_window) → incidents
- _group_by_entity(events) → entity_groups
- _match_pattern_phases(events, pattern_id) → phases
- _create_incident(...) → incident_dict
```

Data Flow:

```
Events → Group by Entity → Match Phases → Calculate Confidence → Create Incident
```

src/config_processor.py - Configuration Management

- **Purpose**: Load and convert security rules from Excel to operational formats
- **Supported Sources**: EntraID, PaloAlto NGFW
- **Export Formats**: JSON, YAML (SIGMA-compatible)

Key Methods:

```
- load_entraid_rules(file_path) → rules_list
- load_paloalto_rules(file_path) → rules_list
- export_to_json(output_dir)
- export_to_yaml(output_dir)
- generate_detection_summary() → summary_dict
```

Rule Schema (Common Structure):

```
{
    'id': 'ENTRAID-001',
    'use_case': 'Intrusion',
    'detection_rule': 'Description...',
    'incident_rule': 'Correlation logic...',
    'severity': 'High',
    'mitre_tactic': 'Credential Access',
    'mitre_technique': 'T1110',
    'category': 'Account Abuse',
    'query': 'CQL query template...',
    'source': 'EntraID' | 'PaloAlto'
}
```

src/use_case_manager.py - Use Case Lifecycle

- **Purpose**: Manage 10 security use cases through their lifecycle
- **Framework**: MAGMA (Mission → Activity → Goals → Mitigation → Actions)

- **Statuses**: DRAFT → TESTING → ACTIVE → TUNING → RETIRED

10 Pre-loaded Use Cases:

1. UC-001-RANSOMWARE
2. UC-002-DATA-THEFT
3. UC-003-DOS
4. UC-004-SUPPLY-CHAIN
5. UC-005-INTRUSION
6. UC-006-MALWARE
7. UC-007-MISCONFIGURATION
8. UC-008-SOCIAL-ENGINEERING
9. UC-009-INFO-MANIPULATION
10. UC-010-FINANCIAL-FRAUD

Key Methods:

```
- get_use_case(use_case_id) → use_case_dict
- list_use_cases(status=None) → use_cases_list
- update_use_case_status(use_case_id, new_status)
- get_coverage_report() → coverage_dict
- export_use_cases(file_path)
```

src/soar_playbooks.py - Automated Response

- **Purpose**: Execute automated response workflows for threats
- **6 Pre-built Playbooks**:
- PB-R1-RANSOMWARE (MTTI: 3 min)
- PB-D1-EXFILTRATION (MTTI: 5 min)
- PB-IN1-INTRUSION (MTTI: 10 min)
- PB-FF1-FRAUD (MTTI: 5 min)
- PB-M2-MALWARE (MTTI: 3 min)
- PB-DOS1-DOS (MTTI: 5 min)

Playbook Structure:

```
{
    'id': 'PB-R1-RANSOMWARE',
    'name': 'Ransomware Containment and Recovery',
    'pattern_id': 'R1',
    'steps': [
        {
            'step': 1,
            'action': 'isolate_endpoint',
            'integration': 'Falcon API',
            'description': 'Quarantine infected host'
        },
        # ... more steps
    ],
    'mtti_target': '3 minutes',
    'automation_level': 'full' | 'semi-automated'
}
```

Key Methods:

```
- execute_playbook(playbook_id, incident) → execution_result
- get_playbook(playbook_id) → playbook_dict
- list_playbooks() → playbooks_list
- get_execution_history(limit) → executions_list
```

src/threat_intelligence.py - Threat Actors

- **Purpose**: Manage threat actor profiles and IOCs
- **9 Pre-loaded Threat Actors**:
- Cybercriminals: LockBit, Clop, FIN12
- Nation-States: APT29, APT41, Lazarus Group
- Hacktivists: KillNet/Anonymous
- Insiders: Malicious Insider
- Supply Chain: UNC Groups

Threat Actor Schema:

```
{
    'name': 'LockBit',
    'type': 'Cybercriminal',
    'geography': 'Global',
    'motivation': 'Financial - RaaS operator',
    'techniques': ['Double extortion', 'Lateral movement'],
    'impact': 'Encryption of critical systems',
    'mitre_tactics': ['Initial Access', 'Execution'],
    'mitre_techniques': ['T1486', 'T1021.001'],
    'severity': 'Critical',
    'likelihood': '★★★★★'
}
```

Key Methods:

```
- get_threat_actor(actor_name) → actor_dict
- list_threat_actors(actor_type) → actors_list
- add_ioc(ioc_dict)
- search_iocs(ioc_type, threat_actor) → iocs_list
- get_threat_landscape_summary() → summary_dict
```

2.3 Configuration Schema

config/config_template.json Structure:

```
{
  "framework": {
    "name": "SOaC Framework",
    "version": "1.0.0",
    "organization": "SOaC Framework Team",
    "environment": "production"
  },
  "correlation": {
    "confidence_threshold": 3,
    "time_windows": {
      "real_time": 15,
      "short_term": 90,
      "long_term": 1440
    },
    "entity_pivots": ["UserName", "ComputerName", "aip", "SHA256HashData"]
  },
  "integrations": {
    "falcon": { "api_url": "", "client_id": "", "enabled": true },
    "entraid": { "tenant_id": "", "client_id": "", "enabled": true },
    "paloalto": { "api_url": "", "api_key": "", "enabled": true },
    "servicenow": { "instance_url": "", "username": "", "enabled": true }
  },
  "soar": {
    "auto_execute_playbooks": false,
    "require_approval_for": ["PB-FF1-FRAUD"],
    "notification_channels": {
      "email": { "enabled": true },
      "teams": { "enabled": true }
    }
  }
}
```


---

3. Data Structures and Schemas

3.1 EntraID Authentication Rules

Excel Schema (16 rules loaded):

Column	Type	Description	Example
#	Integer	Rule number	1
Use Case	String	Use case category	“Intrusion”
Detection Rule	String	Plain language description	“Many auth failures per account”
Incident Rule	String	Correlation logic	“Burst from same IP + success”
Severity	String	High/Medium/Low	“High”
MITRE Tactic	String	ATT&CK tactic	“Credential Access”
MITRE Technique	String	ATT&CK technique	“T1110 (Brute Force)”
Category	String	Rule category	“Account Abuse”
CQL Detection Query Template	String	Query logic	“#event.module=entraid…”

Sample Rule:


```
{
  "#": 1,
  "Use Case": "Intrusion",
  "Detection Rule": "Many authentication failures per account (brute-force)",
  "Incident Rule": "Burst from same IP and subsequent success",
  "Severity": "High",
  "MITRE Tactic": "Credential Access",
  "MITRE Technique": "T1110 (Brute Force)",
  "Category": "Account Abuse",
  "CQL Detection Query Template": "#event.module=entraid | event.dataset = entraid.signin..."
}
```


3.2 PaloAlto NGFW Rules

Excel Schema (12 rules loaded):

Same column structure as EntraID, with PaloAlto-specific queries.

Sample Rule:

```
{
  "#": 1,
  "Use Case": "Intrusion",
  "Detection Rule": "Hosts repeatedly connecting to same external IP (beaconing/C2)",
  "Incident Rule": "Pattern sustained across multiple hosts",
  "Severity": "High",
  "MITRE Tactic": "Command and Control",
  "MITRE Technique": "T1071.001 (Web Protocols)",
  "Category": "Beaconing",
  "CQL Detection Query Template": "#repo = paloalto | event.panw.panos.action = al-
low..."
}
```


3.3 Device Integration Schema (Proposed)

For UI configuration, we need to define:

```typescript
interface DeviceIntegration {
  id: string;                    // UUID
  name: string;                  // User-friendly name
  type: 'paloalto' | 'entraid' | 'siem' | 'falcon' | 'servicenow';
  enabled: boolean;
  config: {
    // Type-specific configuration
    api_url?: string;
    api_key?: string;
    client_id?: string;
    client_secret?: string;
    tenant_id?: string;
    // ... other fields
  };
  connection_status: 'connected' | 'disconnected' | 'error';
  last_tested: Date;
  rules_count: number;
  created_at: Date;
  updated_at: Date;
}
```

3.4 Detection Rule Schema (Unified)

```typescript
interface DetectionRule {
  id: string;                    // e.g., "ENTRAID-001"
  device_id: string;             // Reference to device integration
  use_case_id: string;           // Reference to use case
  name: string;                  // Detection rule name
  description: string;           // Plain language description
  incident_rule: string;         // Correlation logic
  severity: 'Critical' | 'High' | 'Medium' | 'Low';
  mitre_tactic: string;          // e.g., "Credential Access"
  mitre_technique: string;       // e.g., "T1110"
  category: string;              // e.g., "Account Abuse"
  query: string;                 // CQL or query template
  enabled: boolean;
  status: 'draft' | 'testing' | 'active' | 'disabled';
  false_positive_rate?: number;
  detection_count?: number;
  created_at: Date;
  updated_at: Date;
}
```

3.5 Incident Schema

```typescript
interface Incident {
  incident_id: string;           // e.g., "INC-R1-20251113093000"
  pattern_id: string;            // e.g., "R1" (Ransomware)
  entity_key: string;            // e.g., "user:jdoe|computer:DESKTOP-001"
  phases_matched: string[];      // MITRE tactics
  confidence_level: 'Critical' | 'High' | 'Medium' | 'Low';
  event_count: number;
  events: Event[];
  severity: 'Critical' | 'High' | 'Medium' | 'Low';
  status: 'open' | 'investigating' | 'contained' | 'resolved' | 'false_positive';
  assigned_to?: string;
  playbook_executions: PlaybookExecution[];
  timestamp: Date;
}
```

---

4. Operational Models Analysis

The uploaded operational model documents provide detailed workflows for:

4.1 Data Theft / Exfiltration (Pattern D1)

Correlation Pattern:

```
Staging → Transfer → Cloud Write
```

Phases:
1. **Local Collection** (Falcon Endpoint): Data staged in ZIP/RAR files
2. **Network Transfer** (Falcon Cloud/PAN-OS): Outbound connections
3. **SaaS/DNS Activity** (Cisco Umbrella): Cloud storage domains
4. **Cloud Upload** (CloudTrail/Azure): S3 PutObject, Blob uploads

Time Window: 0-60 minutes
Pivot Entities: UserName, ComputerName, TargetFileName, aip

6 SOAR Playbooks:
1. Endpoint Isolation
2. Identity Containment
3. Network Containment
4. Cloud Containment
5. Proofpoint & Email Correlation
6. Notification, Forensics, Case Management

4.2 Financial Fraud / Theft (Pattern FF1)

Correlation Pattern:

```
Access Compromise → Fraudulent Transaction → Anomalous Behavior → Mitigation
```

Phases:
1. **Account Compromise** (EntraID): MFA bypass, new geography login
2. **Transaction Manipulation** (ERP/Finance): Payment creation, vendor updates
3. **Cloud/API Misuse** (CloudTrail): Unusual financial dataset access
4. **Data Exfiltration** (Falcon/Umbrella): Invoice/payment file transfers
5. **Anomalous Behavior** (Falcon/AD): Off-hours access, PowerShell usage

Time Window: 6 hours
MTTF Target: < 5 minutes (Mean Time to Freeze Transaction)

6 SOAR Playbooks:
1. Transaction Hold and Reversal
2. Identity and Account Lockdown
3. Network and SaaS Data Block
4. Cloud & API Lockdown
5. Forensic Investigation and Audit
6. Executive and Legal Communication

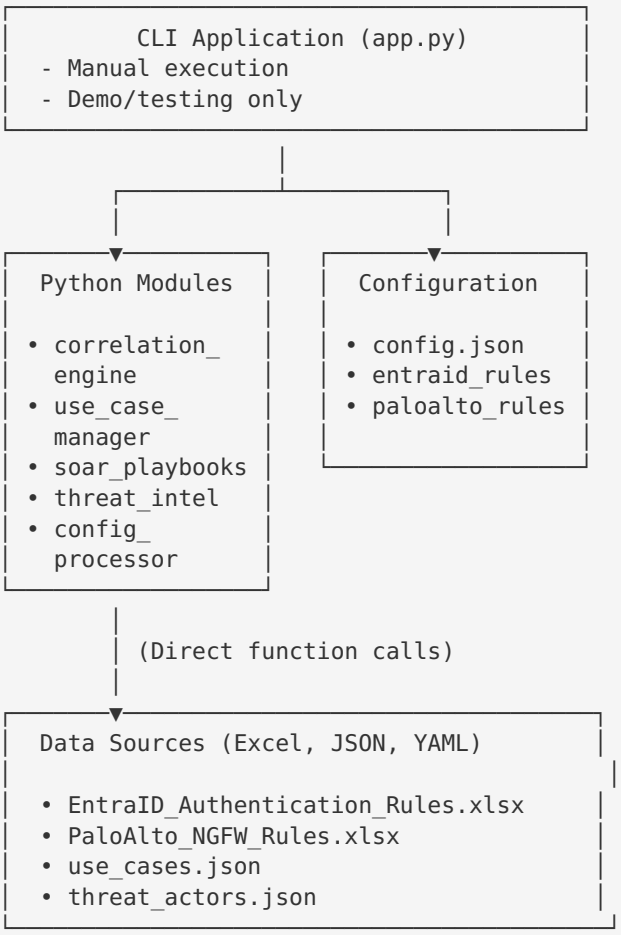
4.3 Key Insights for UI Design

1. **Multi-Phase Detection**: UI must show correlation across 3-5 phases
2. **Entity Pivots**: Need to display relationships (user → device → network → cloud)
3. **Time Windows**: Real-time (5-15 min), Short-term (30-90 min), Long-term (6-24 hrs)
4. **SOAR Integration**: UI should show playbook execution status and step completion
5. **KPI Metrics**: MTTI, MTTDA, Containment Success Rate, False Positive Rate

---

5. Architecture Overview

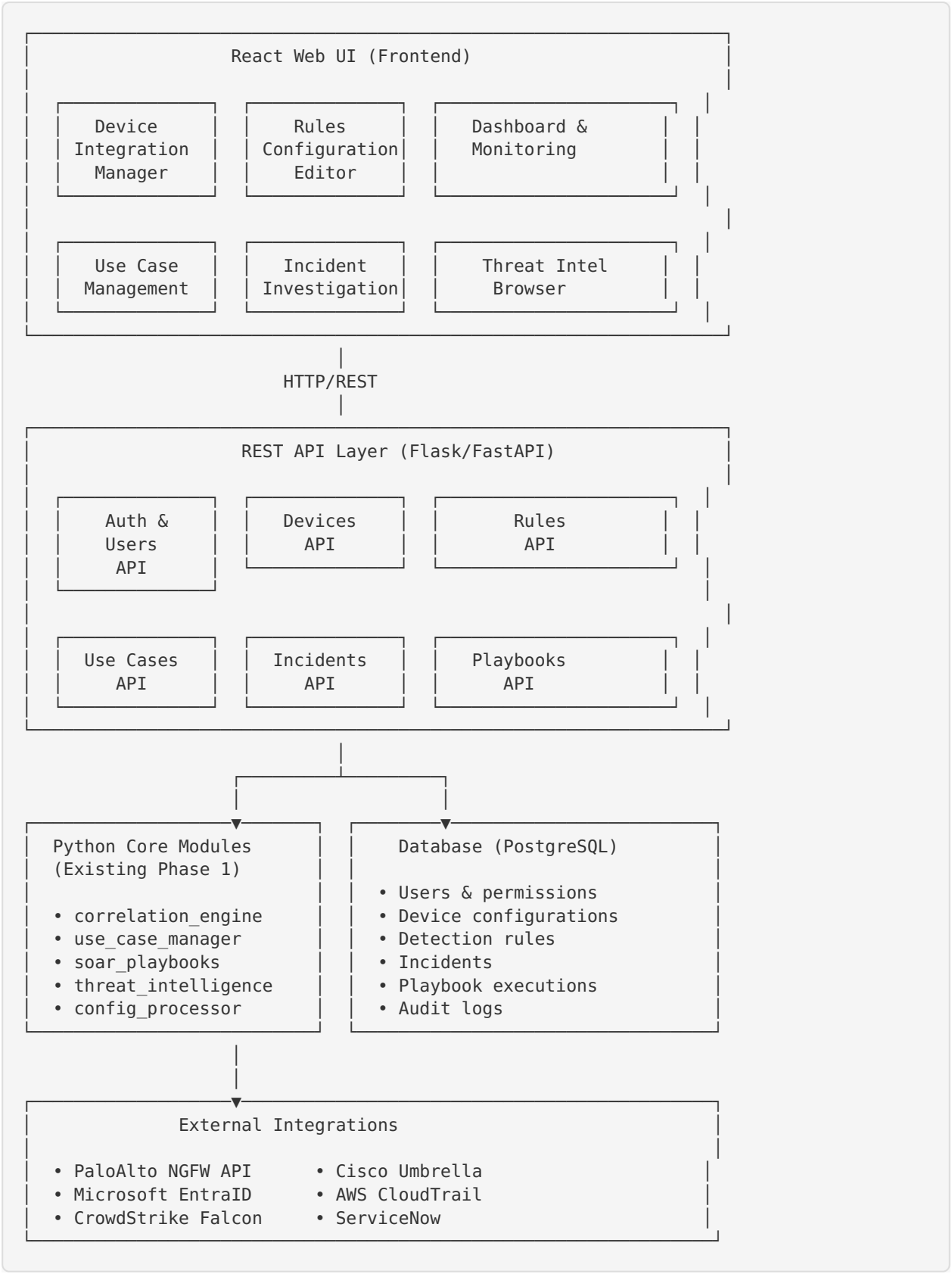
5.1 Current Architecture (Phase 1)



Limitations:

- ❌ No REST API layer
- ❌ No web interface
- ❌ No authentication/authorization
- ❌ No database (file-based storage only)
- ❌ No real-time monitoring
- ❌ No user management

5.2 Proposed Architecture (Phase 2A)



6. React UI Integration Requirements

6.1 Technical Stack Recommendations

Frontend:
- **Framework**: React 18+ with TypeScript
- **State Management**: Redux Toolkit or Zustand
- **UI Library**: Material-UI (MUI) or Ant Design (recommended for enterprise dashboards)
- **Data Visualization**: Recharts or Apache ECharts
- **Forms**: React Hook Form + Zod validation
- **API Client**: Axios with React Query for caching
- **Routing**: React Router v6
- **Authentication**: JWT tokens with refresh mechanism

Backend API:
- **Framework**: FastAPI (recommended) or Flask-RESTful
- **Authentication**: JWT (PyJWT) + bcrypt for passwords
- **Database ORM**: SQLAlchemy
- **Database**: PostgreSQL 14+
- **API Documentation**: OpenAPI/Swagger (auto-generated by FastAPI)
- **CORS**: Configured for React dev server and production

DevOps:
- **Docker**: Containerize both React and Python services
- **nginx**: Reverse proxy for production
- **Docker Compose**: Local development environment

6.2 Integration Points

1. Device Management

UI Components Needed:
- Device list view with status indicators
- Add/Edit device form with type-specific fields
- Connection test button with real-time feedback
- Device metrics dashboard (rules count, last sync, health status)

API Endpoints Required:

```
GET    /api/v1/devices                # List all devices
POST   /api/v1/devices                # Add new device
GET    /api/v1/devices/{id}           # Get device details
PUT    /api/v1/devices/{id}           # Update device config
DELETE /api/v1/devices/{id}           # Remove device
POST   /api/v1/devices/{id}/test      # Test connection
GET    /api/v1/devices/{id}/rules     # Get device rules
POST   /api/v1/devices/{id}/sync      # Sync rules from device
```

Data Flow:

```
React Form → POST /api/v1/devices → Validate → Save to DB →
Test Connection → Update Status → Return Response
```

2. Rule Configuration

UI Components Needed:

- Rule list with filtering (device, severity, status, use case)
- Rule editor with:
- Metadata fields (name, description, severity)
- MITRE ATT&CK selector (tactics/techniques)
- Query editor with syntax highlighting
- Test/validation panel
- Rule import from Excel
- Bulk enable/disable actions

API Endpoints Required:

```
GET    /api/v1/rules                    # List all rules
POST   /api/v1/rules                    # Create rule
GET    /api/v1/rules/{id}               # Get rule details
PUT    /api/v1/rules/{id}               # Update rule
DELETE /api/v1/rules/{id}               # Delete rule
POST   /api/v1/rules/import             # Import from Excel
POST   /api/v1/rules/{id}/test          # Validate rule query
PATCH  /api/v1/rules/{id}/status        # Enable/disable rule
GET    /api/v1/rules/mitre-techniques   # Get MITRE taxonomy
```

3. Dashboard & Monitoring

UI Components Needed:

- Real-time metrics cards:
- Active Incidents
- Open Investigations
- Playbook Executions (24h)
- Average MTTI/MTTDA
- Incident timeline chart
- Top 10 threat actors detected
- Use case coverage heatmap (MITRE ATT&CK matrix)
- Device health status panel
- Recent alerts feed

API Endpoints Required:

```
GET    /api/v1/dashboard/metrics        # Real-time KPIs
GET    /api/v1/dashboard/incidents      # Recent incidents
GET    /api/v1/dashboard/timeline       # Incident timeline data
GET    /api/v1/dashboard/coverage       # MITRE coverage map
GET    /api/v1/dashboard/device-health # Device status summary
```

WebSocket for Real-Time Updates:

```
WS     /ws/incidents                    # Live incident feed
WS     /ws/playbook-executions          # Live playbook status
```

4. Use Case Management

UI Components Needed:
- Use case list with status badges
- Use case details view (MAGMA framework structure)
- Status transition workflow (Draft → Testing → Active → Tuning → Retired)
- Coverage report visualization
- Rule assignment to use cases

API Endpoints Required:

```
GET    /api/v1/use-cases              # List all use cases
POST   /api/v1/use-cases              # Create use case
GET    /api/v1/use-cases/{id}         # Get use case details
PUT    /api/v1/use-cases/{id}         # Update use case
PATCH  /api/v1/use-cases/{id}/status  # Change status
GET    /api/v1/use-cases/{id}/rules   # Get assigned rules
GET    /api/v1/use-cases/coverage     # Coverage report
```

5. Incident Investigation

UI Components Needed:
- Incident list with advanced filtering
- Incident detail view with:
- Entity relationship graph (user → device → network → cloud)
- Event timeline
- Matched phases visualization
- Confidence score breakdown
- Related threat actor profiles
- Playbook execution viewer
- Status management (open → investigating → contained → resolved)
- Comments/notes section

API Endpoints Required:

```
GET    /api/v1/incidents              # List incidents
GET    /api/v1/incidents/{id}         # Get incident details
PATCH  /api/v1/incidents/{id}/status  # Update status
POST   /api/v1/incidents/{id}/comments # Add comment
GET    /api/v1/incidents/{id}/playbooks # Get executed playbooks
POST   /api/v1/incidents/{id}/execute-playbook # Trigger playbook
```

6. Threat Intelligence Browser

UI Components Needed:
- Threat actor directory with search/filter
- Threat actor profile page (motivation, techniques, impact)
- IOC search interface
- MITRE ATT&CK technique mapping
- Threat landscape summary

API Endpoints Required:

```
GET    /api/v1/threat-intel/actors     # List threat actors
GET    /api/v1/threat-intel/actors/{id} # Get actor profile
GET    /api/v1/threat-intel/iocs       # Search IOCs
POST   /api/v1/threat-intel/iocs       # Add IOC
GET    /api/v1/threat-intel/summary    # Threat landscape summary
```

---

7. Recommended UI Architecture

7.1 Component Hierarchy

```
App
├── AuthProvider (JWT token management)
├── Router
│   ├── Login
│   ├── Dashboard (/)
│   │   ├── MetricsCards
│   │   ├── IncidentTimeline
│   │   ├── ThreatActorsPanel
│   │   ├── DeviceHealthPanel
│   │   └── RecentAlertsPanel
│   │
│   ├── Devices (/devices)
│   │   ├── DeviceList
│   │   ├── DeviceForm (Add/Edit)
│   │   ├── DeviceDetails
│   │   └── ConnectionTestModal
│   │
│   ├── Rules (/rules)
│   │   ├── RuleList
│   │   ├── RuleEditor
│   │   ├── RuleImport
│   │   └── MitreSelector
│   │
│   ├── UseCases (/use-cases)
│   │   ├── UseCaseList
│   │   ├── UseCaseDetails
│   │   ├── MagmaFrameworkView
│   │   └── CoverageReport
│   │
│   ├── Incidents (/incidents)
│   │   ├── IncidentList
│   │   ├── IncidentDetails
│   │   ├── EntityGraph
│   │   ├── EventTimeline
│   │   └── PlaybookViewer
│   │
│   ├── ThreatIntel (/threat-intel)
│   │   ├── ActorDirectory
│   │   ├── ActorProfile
│   │   ├── IOCSearch
│   │   └── MitreMatrix
│   │
│   └── Settings (/settings)
│       ├── CorrelationConfig
│       ├── SoarConfig
│       ├── UserManagement
│       └── AuditLog
```

7.2 State Management Strategy

Global State (Redux/Zustand):
- Authentication state (user, token, permissions)
- Active devices list
- Use cases list
- Threat actors list
- Dashboard metrics cache

Server State (React Query):
- Incidents (with pagination, filtering, real-time updates)
- Rules (with search, filtering)
- Device details
- Playbook executions

Local State (useState/useReducer):
- Form inputs
- UI toggles (modals, drawers)
- Filter selections

7.3 Key Features to Implement

Phase 2A - Core Features (MVP)

1. ✅ **Device Management**
   - Add/Edit/Delete PaloAlto, EntraID, SIEM devices
   - Connection testing
   - Status monitoring

2. ✅ **Rule Configuration**
   - CRUD operations for detection rules
   - Import from Excel (EntraID, PaloAlto)
   - Rule validation and testing

3. ✅ **Dashboard**
   - Real-time metrics
   - Incident feed
   - Device health overview

4. ✅ **Authentication & Authorization**
   - Login/logout
   - JWT token management
   - Role-based access (Admin, Analyst, Viewer)

Phase 2B - Enhanced Features

1. 🔄 **Use Case Management**
   - Lifecycle management
   - Coverage reporting
   - Rule assignment

2. 🔄 **Incident Investigation**
   - Advanced filtering and search
   - Entity relationship visualization
   - Playbook execution triggering

3. 🔄 **Threat Intelligence Browser**
   - Actor profiles
   - IOC management
   - MITRE ATT&CK integration

7.4 Design System Guidelines

Color Scheme (Security Operations Theme):
- Primary: Blue (#1976D2) - Trust, professionalism
- Secondary: Orange (#FF6B35) - Alerts, attention
- Success: Green (#4CAF50) - Connected, healthy
- Warning: Yellow (#FFC107) - Medium severity
- Error: Red (#F44336) - Critical, high severity
- Neutral: Gray (#757575) - Disabled, neutral info

Severity Color Mapping:

```
.severity-critical { background: #D32F2F; }
.severity-high     { background: #FF6B35; }
.severity-medium   { background: #FFC107; }
.severity-low      { background: #4CAF50; }
```

Typography:
- Headings: Inter or Roboto (sans-serif)
- Body: System fonts for performance
- Code/Queries: Fira Code or Monaco (monospace)

---

8. API Specifications Needed

8.1 REST API Structure

Base URL: `http://localhost:5000/api/v1` (development)

Authentication: JWT Bearer Token

```
Authorization: Bearer <token>
```

Common Response Format:

```
{
  "success": true,
  "data": { ... },
  "message": "Operation successful",
  "timestamp": "2025-11-13T09:30:00Z"
}
```

Error Response Format:

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid device configuration",
    "details": {
      "api_url": ["This field is required"]
    }
  },
  "timestamp": "2025-11-13T09:30:00Z"
}
```

8.2 Core API Endpoints

Authentication API

```
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh
GET    /api/v1/auth/me
```

Example Login Request:

```
POST /api/v1/auth/login
{
  "username": "admin",
  "password": "SecurePass123!"
}
```

Response:

```
{
  "success": true,
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "dGhpcyBpcyBhIHJlZnJlc2ggdG9rZW4...",
    "expires_in": 3600,
    "user": {
      "id": "user-001",
      "username": "admin",
      "email": "admin@soac.local",
      "role": "admin"
    }
  }
}
```

Devices API

```
# Device Model
{
  "id": "dev-001",
  "name": "PaloAlto-Main-NGFW",
  "type": "paloalto",
  "enabled": true,
  "config": {
    "api_url": "https://firewall.company.com/api",
    "api_key": "***********",
    "verify_ssl": true
  },
  "connection_status": "connected",
  "last_tested": "2025-11-13T09:00:00Z",
  "rules_count": 12,
  "created_at": "2025-11-01T10:00:00Z",
  "updated_at": "2025-11-13T08:00:00Z"
}
```

Endpoints:

```
GET    /api/v1/devices
POST   /api/v1/devices
GET    /api/v1/devices/{id}
PUT    /api/v1/devices/{id}
DELETE /api/v1/devices/{id}
POST   /api/v1/devices/{id}/test      # Test connection
POST   /api/v1/devices/{id}/sync      # Sync rules
```

Rules API

```
# Rule Model
{
  "id": "rule-entraid-001",
  "device_id": "dev-001",
  "use_case_id": "UC-005-INTRUSION",
  "name": "Brute Force Detection",
  "description": "Many authentication failures per account",
  "incident_rule": "Burst from same IP and subsequent success",
  "severity": "High",
  "mitre_tactic": "Credential Access",
  "mitre_technique": "T1110",
  "category": "Account Abuse",
  "query": "#event.module=entraid | event.outcome = failure...",
  "enabled": true,
  "status": "active",
  "false_positive_rate": 0.05,
  "detection_count": 127,
  "created_at": "2025-11-01T10:00:00Z",
  "updated_at": "2025-11-13T08:00:00Z"
}
```

Endpoints:

```
GET    /api/v1/rules?device_id=&severity=&status=
POST   /api/v1/rules
GET    /api/v1/rules/{id}
PUT    /api/v1/rules/{id}
DELETE /api/v1/rules/{id}
POST   /api/v1/rules/import           # Import from Excel
POST   /api/v1/rules/{id}/test        # Validate query
PATCH  /api/v1/rules/{id}/status      # Enable/disable
```

Dashboard API

```
GET    /api/v1/dashboard/metrics
```

Response:

```
{
  "success": true,
  "data": {
    "active_incidents": 3,
    "open_investigations": 7,
    "playbook_executions_24h": 15,
    "mtti_average_minutes": 4.2,
    "mttda_average_minutes": 6.8,
    "device_health": {
      "connected": 5,
      "disconnected": 1,
      "error": 0
    },
    "incidents_by_severity": {
      "critical": 1,
      "high": 2,
      "medium": 4,
      "low": 0
    }
  }
}
```

Use Cases API

```
# Use Case Model
{
  "id": "UC-001-RANSOMWARE",
  "title": "Ransomware Detection and Response",
  "mission": "Ensure continuity of services and data access",
  "activity": "Encryption of files, backup deletion, extortion",
  "goals": ["Detect ransomware encryption tools", "Detect backup modification"],
  "mitigation": ["EDR", "Immutable backups", "Network segmentation"],
  "mitre_techniques": ["T1486", "T1059.001", "T1027"],
  "status": "active",
  "severity": "Critical",
  "owner": "SOaC Framework Team",
  "rules_count": 8,
  "incidents_count": 2,
  "created_date": "2025-11-01T10:00:00Z",
  "last_updated": "2025-11-13T08:00:00Z"
}
```

Incidents API

```
# Incident Model
{
  "incident_id": "INC-R1-20251113093000",
  "pattern_id": "R1",
  "pattern_name": "Ransomware Chain",
  "entity_key": "user:jdoe|computer:DESKTOP-001",
  "phases_matched": ["Initial Access", "Execution", "Impact"],
  "confidence_level": "High",
  "event_count": 5,
  "severity": "Critical",
  "status": "investigating",
  "assigned_to": "analyst@soac.local",
  "entities": {
    "user": "jdoe",
    "computer": "DESKTOP-001",
    "ip": "192.168.1.100"
  },
  "playbook_executions": [
    {
      "execution_id": "EXEC-PB-R1-001",
      "playbook_id": "PB-R1-RANSOMWARE",
      "status": "completed",
      "steps_completed": 5,
      "steps_total": 5
    }
  ],
  "created_at": "2025-11-13T09:30:00Z",
  "updated_at": "2025-11-13T09:45:00Z"
}
```

8.3 Database Schema

Key Tables:

```sql
-- Users and authentication
CREATE TABLE users (
    id UUID PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL,  -- 'admin', 'analyst', 'viewer'
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Device integrations
CREATE TABLE devices (
    id UUID PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    type VARCHAR(50) NOT NULL,  -- 'paloalto', 'entraid', 'siem'
    enabled BOOLEAN DEFAULT TRUE,
    config JSONB NOT NULL,
    connection_status VARCHAR(20),  -- 'connected', 'disconnected', 'error'
    last_tested TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Detection rules
CREATE TABLE rules (
    id VARCHAR(50) PRIMARY KEY,
    device_id UUID REFERENCES devices(id) ON DELETE CASCADE,
    use_case_id VARCHAR(50),
    name VARCHAR(200) NOT NULL,
    description TEXT,
    incident_rule TEXT,
    severity VARCHAR(20) NOT NULL,
    mitre_tactic VARCHAR(100),
    mitre_technique VARCHAR(50),
    category VARCHAR(100),
    query TEXT NOT NULL,
    enabled BOOLEAN DEFAULT TRUE,
    status VARCHAR(20) DEFAULT 'draft',
    false_positive_rate FLOAT,
    detection_count INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Incidents
CREATE TABLE incidents (
    incident_id VARCHAR(50) PRIMARY KEY,
    pattern_id VARCHAR(10) NOT NULL,
    pattern_name VARCHAR(100),
    entity_key VARCHAR(500),
    phases_matched JSONB,
    confidence_level VARCHAR(20),
    event_count INTEGER,
    events JSONB,
    severity VARCHAR(20),
    status VARCHAR(50) DEFAULT 'open',
    assigned_to VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

```sql
-- Playbook executions
CREATE TABLE playbook_executions (
    execution_id VARCHAR(50) PRIMARY KEY,
    incident_id VARCHAR(50) REFERENCES incidents(incident_id),
    playbook_id VARCHAR(50) NOT NULL,
    playbook_name VARCHAR(200),
    status VARCHAR(50),  -- 'pending', 'running', 'completed', 'failed'
    steps_completed INTEGER,
    steps_total INTEGER,
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Audit logs
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    action VARCHAR(100) NOT NULL,
    resource_type VARCHAR(50),
    resource_id VARCHAR(50),
    details JSONB,
    ip_address VARCHAR(45),
    created_at TIMESTAMP DEFAULT NOW()
);
```

---

9. Gaps and Recommendations

9.1 Critical Gaps (Must Address for Phase 2A)

1. No REST API Layer 🔴

Current State: All functionality is accessed via direct Python function calls
Impact: Cannot integrate with React UI
Recommendation:
- Build REST API using **FastAPI** (preferred) or Flask-RESTful
- Wrap existing Python modules with API endpoints
- Implement proper error handling and validation

Effort: 40 hours

2. No Authentication System 🔴

Current State: No user management, login, or permissions
Impact: Security risk, no multi-user support
Recommendation:
- Implement JWT-based authentication
- Add user model with roles (Admin, Analyst, Viewer)
- Integrate with EntraID for SSO (future enhancement)

Effort: 20 hours

3. No Database 🔴

Current State: File-based storage (JSON, Excel)
Impact: No transactional integrity, poor scalability, no audit trail
Recommendation:

- Implement PostgreSQL database
- Use SQLAlchemy ORM
- Migrate configuration data from JSON to DB
- Keep Excel import/export as a feature

Effort: 30 hours

4. No Real-Time Updates 🔴

Current State: CLI-based, one-time execution
Impact: Dashboard won't reflect live changes
Recommendation:
- Implement WebSocket server (Socket.IO or native WebSockets)
- Push incident updates and playbook executions to UI
- Add polling fallback for compatibility

Effort: 15 hours

9.2 Important Gaps (Should Address)

5. No Configuration Validation 🟡

Current State: Minimal validation when loading configs
Impact: Bad configurations can crash the system
Recommendation:
- Use Pydantic models for API request validation
- Implement device connection testing before saving
- Add query syntax validation for detection rules

Effort: 10 hours

6. No Audit Trail 🟡

Current State: No logging of user actions
Impact: Cannot track who changed what
Recommendation:
- Add audit_logs table
- Log all CRUD operations (devices, rules, use cases)
- Display audit history in UI

Effort: 8 hours

7. Limited Error Handling 🟡

Current State: Basic try-except blocks, limited error messages
Impact: Difficult to debug issues
Recommendation:
- Implement structured error responses
- Add detailed logging (ELK stack integration later)
- User-friendly error messages in UI

Effort: 5 hours

9.3 Nice-to-Have Enhancements

8. No RBAC (Role-Based Access Control) 🟢

Recommendation: Implement granular permissions (view, edit, delete for each resource type)
Effort: 12 hours

9. No API Rate Limiting 🟢

Recommendation: Add rate limiting to prevent abuse (Flask-Limiter or FastAPI RateLimiter)
Effort: 4 hours

10. No API Versioning 🟢

Recommendation: Already planned with `/api/v1/` prefix - good foundation
Effort: 2 hours

---

10. Implementation Roadmap

Phase 2A - Sprint 1: Foundation (Week 1-2)

Goal: Build API foundation and authentication

Week 1: Backend API Setup

- [ ] Set up FastAPI project structure
- [ ] Implement database schema (PostgreSQL + SQLAlchemy)
- [ ] Create User model and authentication endpoints
- [ ] Implement JWT token generation and validation
- [ ] Set up CORS for React integration
- [ ] Write API endpoint for device CRUD operations

Deliverables:
- FastAPI application with `/api/v1/auth` and `/api/v1/devices` endpoints
- PostgreSQL database with migrations
- Postman collection for API testing

Week 2: Core API Endpoints

- [ ] Implement Rules API endpoints
- [ ] Implement Dashboard API endpoints
- [ ] Add connection testing for devices
- [ ] Integrate existing Python modules (correlation_engine, etc.)
- [ ] Write unit tests for API endpoints
- [ ] Set up Docker development environment

Deliverables:
- Complete REST API for devices and rules
- Docker Compose setup for local development
- API documentation (Swagger UI)

Phase 2A - Sprint 2: React UI Core (Week 3-4)

Goal: Build React UI foundation and device management

Week 3: React Setup & Device Management

- [ ] Initialize React TypeScript project (Vite)
- [ ] Set up Material-UI or Ant Design
- [ ] Implement authentication flow (login, JWT storage, auto-logout)
- [ ] Build Device Management pages:
- Device list view
- Add/Edit device form
- Connection test functionality
- [ ] Set up React Query for API calls
- [ ] Implement global state management (Zustand/Redux)

Deliverables:
- React application with authentication
- Device management UI (CRUD operations)
- Responsive design for desktop and tablet

Week 4: Rule Configuration UI

- [ ] Build Rule List page with filtering
- [ ] Create Rule Editor component
- [ ] Implement Excel import functionality
- [ ] Add MITRE ATT&CK technique selector
- [ ] Build query editor with syntax highlighting
- [ ] Add rule validation and testing

Deliverables:
- Complete rule configuration UI
- Excel import/export functionality
- Rule testing interface

Phase 2A - Sprint 3: Dashboard & Monitoring (Week 5-6)

Goal: Real-time dashboard and incident feed

Week 5: Dashboard Development

- [ ] Build dashboard layout with metrics cards
- [ ] Implement real-time metrics API integration
- [ ] Create incident timeline chart (Recharts)
- [ ] Add device health status panel
- [ ] Build recent alerts feed
- [ ] Implement WebSocket connection for live updates

Deliverables:
- Real-time dashboard with KPIs
- Live incident feed
- Device health monitoring

Week 6: Polish & Testing

- [ ] End-to-end testing (Cypress or Playwright)

- [ ] API integration testing
- [ ] UI/UX refinements
- [ ] Performance optimization
- [ ] Documentation (user guide, deployment guide)
- [ ] Security audit (OWASP top 10 checks)

Deliverables:
- Production-ready application
- Test coverage reports
- Deployment documentation

Phase 2B - Future Enhancements (Week 7+)

Use Case Management:
- [ ] Use case lifecycle UI
- [ ] MAGMA framework visualization
- [ ] Coverage reporting

Incident Investigation:
- [ ] Advanced incident search and filtering
- [ ] Entity relationship graph (D3.js or Cytoscape.js)
- [ ] Playbook execution triggering and monitoring
- [ ] Investigation notes and collaboration

Threat Intelligence:
- [ ] Threat actor directory
- [ ] IOC management interface
- [ ] MITRE ATT&CK matrix visualization
- [ ] Threat landscape analytics

Advanced Features:
- [ ] SSO integration (EntraID)
- [ ] RBAC with granular permissions
- [ ] Notification system (email, Teams, Slack)
- [ ] Export reports (PDF, CSV, Excel)
- [ ] API key management for integrations

Conclusion

This comprehensive analysis provides a solid foundation for Phase 2A development. The key priorities are:

1. **Build REST API layer** to expose existing Python functionality
2. **Implement authentication** for multi-user access
3. **Add database layer** for persistent storage
4. **Create React UI** for device and rule management
5. **Build real-time dashboard** for monitoring

The existing Phase 1 codebase is well-structured and can be incrementally enhanced with the API layer. The operational models provide clear guidance for UI workflows, especially for multi-phase correlation and SOAR playbook execution.

Next Steps:
1. Review and approve this analysis
2. Set up development environment (PostgreSQL, FastAPI, React)
3. Begin Sprint 1: Backend API Foundation
4. Parallel track: Design UI mockups and wireframes

Document Version: 1.0
Last Updated: November 13, 2025
Prepared By: SOaC Framework Development Team