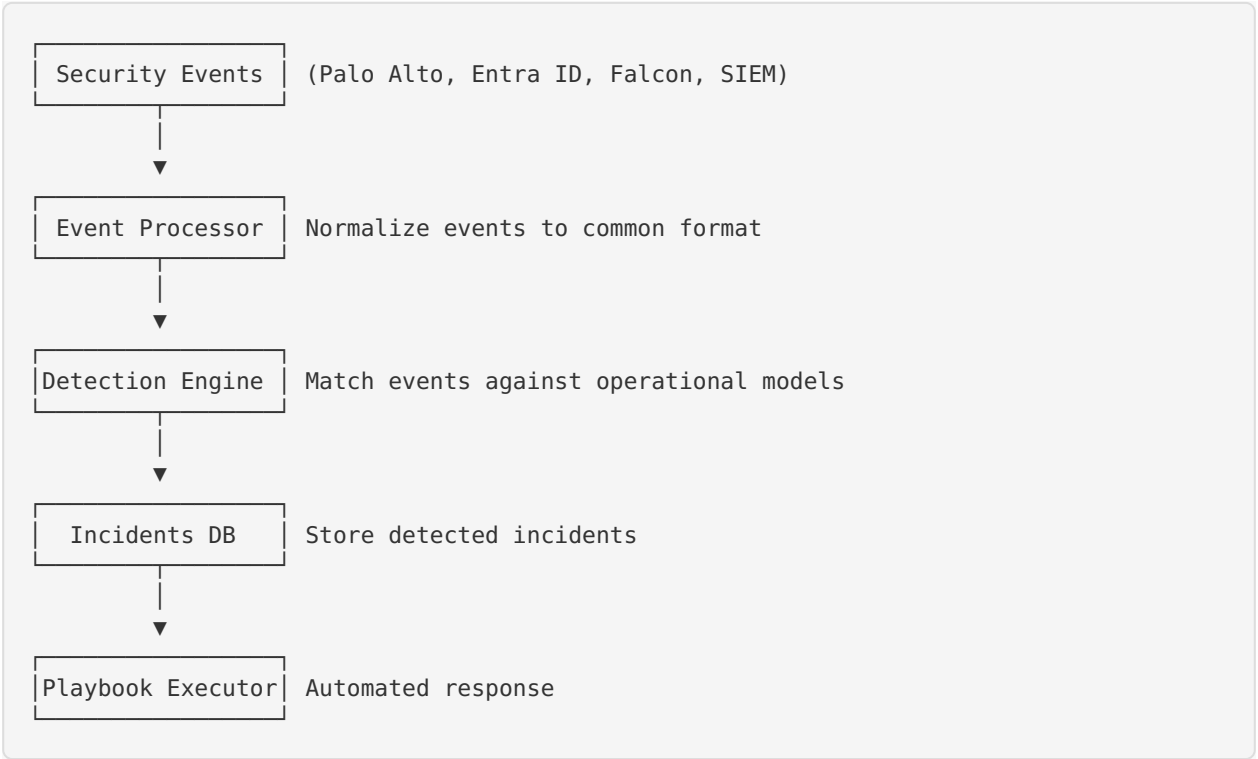
Operational Models & Detection Engine

Overview

The SOaC Framework Phase 3B introduces an intelligent detection engine that processes security events against operational models to detect multi-phase attacks. Operational models define attack patterns, detection logic, correlation rules, and automated response playbooks.

Architecture



Operational Model Structure

Each operational model is defined in a DOCX document with the following sections:

1. Objective

- **Goal**: What the model aims to detect
- **Business Outcome**: Impact on security posture

2. Correlation Pattern

- **Pattern ID**: Unique identifier (e.g., D1, FF1, M2)
- **Description**: Attack flow description
- **Phases**: Sequential attack stages
- **Correlation Window**: Time window for correlating events (e.g., 60 minutes)
- **Pivot Entities**: Fields used for correlation (UserName, ComputerName, IP, etc.)

3. Detection Queries

- **Query A-E**: Individual detection queries for each phase
- **Combined Rule**: Correlation logic joining all queries
- **Trigger Condition**: Threshold for incident creation (e.g., ≥3 phases matched)

4. Alert Policy

- **Severity**: Critical, High, Medium, Low
- **Trigger**: Condition for creating alert
- **Suppression Window**: De-duplication period
- **Escalation Path**: SOC → DFIR → Management
- **Runbook Reference**: Associated playbook ID

5. Response Playbooks

- **Step-by-step**: Automated response actions
- **Integrations**: APIs and tools involved
- **Logic**: Conditional execution paths

6. Decision Matrix

- **Conditions**: Mapping incident attributes to playbooks
- **Response Paths**: Which playbooks to execute
- **Playbooks Triggered**: Automated response actions

7. KPI Metrics

- **MTTD**: Mean Time to Detect
- **MTTC**: Mean Time to Contain
- **False Positive Rate**: Detection accuracy
- **Containment Success Rate**: Response effectiveness

Available Operational Models

1. Data Theft / Exfiltration

- **Pattern ID**: D1
- **Phases**: Staging → Transfer → Cloud Upload
- **Severity**: High
- **Sources**: Falcon Endpoint, PAN-OS, Umbrella, CloudTrail
- **Correlation Window**: 60 minutes

2. Financial Fraud

- **Pattern ID**: FF1
- **Phases**: Account Compromise → Transaction → Data Exfil
- **Severity**: Critical
- **Sources**: Entra ID, ERP Systems, CloudTrail
- **Correlation Window**: 6 hours

3. Malware Infection

- **Pattern ID**: M2
- **Phases**: Delivery → Execution → C2 → Persistence

- **Severity**: Critical
- **Sources**: Proofpoint, Falcon, Umbrella, PAN-OS
- **Correlation Window**: 60 minutes

4. Denial of Service (DoS)

- **Pattern ID**: DDoS1
- **Phases**: Flood → Service Degradation → Mitigation
- **Severity**: Critical
- **Sources**: PAN-OS, Umbrella, CloudTrail
- **Correlation Window**: 15 minutes

5. Intrusion Detection

- **Pattern ID**: I1
- **Phases**: Initial Foothold → Privilege Abuse → Lateral Movement
- **Severity**: Critical
- **Sources**: Falcon, AD, PAN-OS, Infoblox
- **Correlation Window**: 90 minutes

Detection Engine

Event Processing Flow

1. **Event Ingestion**
   - Events received from devices (Palo Alto, Entra ID, etc.)
   - Raw events in vendor-specific format

2. **Event Normalization**
   - Convert to common schema
   - Extract key fields: UserName, ComputerName, IP, etc.
   - Add event classification

3. **Pattern Matching**
   - Match event against operational model phases
   - Check indicators and sources
   - Identify which phase the event belongs to

4. **Entity State Tracking**
   - Group events by entity (user, computer, IP)
   - Track matched phases per entity
   - Clean old events outside correlation window

5. **Incident Creation**
   - Check if threshold met (e.g., 3+ phases)
   - Calculate confidence level (high/medium/low)
   - Create incident in database

6. **Playbook Execution**
   - Determine applicable playbooks from decision matrix
   - Execute automated response actions
   - Log all actions for audit

Correlation Logic

```
# Entity-based correlation
entity_key = f"user:{UserName}|host:{ComputerName}"

# Phase tracking
phases_matched = ["Staging", "Network Transfer", "Cloud Upload"]

# Threshold check
if len(phases_matched) >= 3:
    create_incident(
        confidence="high",
        severity="Critical"
    )
```

Confidence Levels

- **High (80-100%)**: Most phases matched, high certainty
- **Medium (50-79%)**: Some phases matched, requires investigation
- **Low (<50%)**: Few phases matched, potential false positive

API Endpoints

Incidents Management

List Incidents

```
GET /api/v1/incidents/
Query Parameters:
  - status: open|investigating|contained|resolved|false_positive
  - severity: Critical|High|Medium|Low
  - pattern_id: Model ID
  - days: Lookback period (default: 7)
```

Get Incident Details

```
GET /api/v1/incidents/{incident_id}
```

Update Incident

```
PATCH /api/v1/incidents/{incident_id}
Body: {
  "status": "investigating",
  "assigned_to": "analyst@company.com"
}
```

Assign Incident

```
POST /api/v1/incidents/{incident_id}/assign
Body: {
  "analyst": "analyst@company.com"
}
```

Execute Playbook

```
POST /api/v1/incidents/{incident_id}/execute-playbook
Body: {
  "playbook_id": "PB_1",  # Optional, executes all if omitted
  "mode": "manual|automated"
}
```

Operational Models

List Models

```
GET /api/v1/operational-models/
```

Get Model Details

```
GET /api/v1/operational-models/{model_id}
```

Reload Models

```
POST /api/v1/operational-models/reload
```

Detection Engine

Process Event

```
POST /api/v1/detection/process-event
Body: {
  "event": { ... },
  "source": "paloalto|entraid|falcon|siem"
}
```

Process Batch Events

```
POST /api/v1/detection/process-batch
Body: {
  "events": [ ... ],
  "source": "paloalto"
}
```

Get Detection Stats

```
GET /api/v1/detection/stats
```

Creating Custom Operational Models

Step 1: Create DOCX Document

Use the provided template structure with these sections:

1. Objective
2. Correlation Pattern

3. Detection Queries
4. Alert Policy
5. Dashboard Panels
6. SOAR Playbooks
7. Decision Matrix
8. KPI Metrics

Step 2: Define Attack Phases

Create a table with columns:
- **Phase**: Name of attack stage
- **Source**: Data source (Falcon, PAN-OS, etc.)
- **Indicators**: Observable behaviors
- **Correlation Fields**: Fields for correlation

Example:

```
Phase: Data Staging
Source: Falcon Endpoint
Indicators: ZIP, RAR, CSV files in temp directories
Correlation Fields: UserName, ComputerName, TargetFileName
```

Step 3: Write Detection Queries

Use LogScale/Humio syntax:

```
Query A – Data Staging
#event_simpleName=DataStaged or #event_simpleName=FileWriteInfo
| TargetFileName=/\.(zip|rar|7z|csv)$/i
| groupBy([ComputerName, UserName], function=count())
```

Step 4: Define Correlation Rule

```
join(
  queryA=DataStaging,
  queryB=NetworkTransfer,
  queryC=CloudUpload,
  on=[UserName, ComputerName],
  within=1h
)
| groupBy([UserName, ComputerName], function=count())
| _count >= 3
```

Step 5: Create Response Playbooks

Define step-by-step actions:

```
PLAYBOOK 1: Endpoint Isolation
Step 1: Validate correlation match
Step 2: Isolate endpoint via Falcon API
Step 3: Tag device for DFIR
Step 4: Notify SOC team
```

Step 6: Upload and Load

1. Save DOCX file to `/home/ubuntu/Uploads/` directory
2. Name it with “operational model” suffix (e.g., “Ransomware Operational Model.docx”)
3. Call reload API endpoint:
   `bash`
   `POST /api/v1/operational-models/reload`

Testing

Generate Test Events

```
POST /api/v1/detection/test-event
```

This creates sample events for Data Theft pattern and processes them through the detection engine.

Manual Event Processing

```
curl -X POST http://localhost:8000/api/v1/detection/process-event \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "event": {
      "UserName": "john.doe@company.com",
      "ComputerName": "LAPTOP01",
      "FileName": "sensitive.zip"
    },
    "source": "falcon"
  }'
```

Performance Considerations

Detection Engine

- Events processed in-memory for speed
- Entity state cleaned based on correlation window
- Database writes only on incident creation

Scaling

- Stateless detection engine (can run multiple instances)
- Use message queue (Redis, Kafka) for event ingestion
- Distribute entity tracking across instances

Optimization

- Index database fields: pattern_id, status, severity, created_at
- Cache operational models in memory
- Batch event processing for high volumes

Troubleshooting

Models Not Loading

- Check DOCX files in `/home/ubuntu/Uploads/`
- Verify file names contain “operational model”
- Check backend logs for parsing errors

Incidents Not Created

- Verify correlation window
- Check if threshold is met (e.g., 3+ phases)
- Review entity key generation
- Check event normalization

Playbook Execution Fails

- Verify playbook ID exists in operational model
- Check incident has required fields
- Review decision matrix conditions

Security Considerations

- Incidents contain sensitive data - apply RBAC
- Playbook execution requires admin privileges
- Log all detection and response actions
- Encrypt credentials for external integrations

Roadmap

Phase 4A: Advanced Analytics

- ML-based anomaly detection
- Threat intelligence enrichment
- Behavioral baselining

Phase 4B: Orchestration

- Integration with SOAR platforms
- Automated threat hunting
- Red team emulation

Phase 4C: Compliance

- Automated reporting
- Audit trail management
- Regulatory framework mapping

References

- **MITRE ATT&CK**: https://attack.mitre.org/
- **LogScale Query Language**: https://library.humio.com/
- **Falcon API**: https://falcon.crowdstrike.com/documentation/

- **Palo Alto API**: https://docs.paloaltonetworks.com/

Support

For issues or questions:
- GitHub Issues: https://github.com/ge0mant1s/soac-framework
- Documentation: docs/README.md
- API Docs: http://localhost:8000/api/docs