Prototype Development Workflow Plan
NLP Middleware for Wazuh / ELK SIEM

Prototype Phase Technical Document

February 5, 2026

Contents

1 Introduction

This document defines the structured prototype development workflow for building a Natural Language Interface middleware over ELK-based SIEM platforms (Wazuh / Elastic SIEM). The plan is divided into four execution phases spanning four weeks, focusing on rapid environment setup, schema intelligence, API bridging, and frontend visualization.

2 Phase 1: Environment Setup (Week 1)

2.1 Objective

Establish a working SIEM environment before AI integration.

2.2 Action 1: Install Wazuh / Elastic

- Use Wazuh All-in-One Docker deployment.
- Ensures fastest local deployment.
- Provides Elasticsearch + Wazuh Manager + Dashboard in single stack.

2.3 Action 2: Generate Sample Data

- Install Wazuh Agent on host machine.
- Generate controlled security events:
  - Intentional failed login attempts
  - Suspicious file access attempts
  - Network authentication failures
- Ensures dataset availability for testing queries.

2.4 Action 3: Python Virtual Environment

- Create isolated workspace.
- Install required libraries:

```
pip install langchain openai streamlit elasticsearch requests
```

3 Phase 2: Mapping Strategy (Week 2)

3.1 Objective

Enable AI to understand SIEM log structure and field mappings.

3.2 Action 4: Export Schema

- Use Elasticsearch _mapping API.
- Export JSON schema describing index structure.
- Example:

```
GET index_name/_mapping
```

Purpose:

- Map natural language entities to SIEM fields.
- Example Mapping:
  - user → winlog.event_data.TargetUserName

3.3 Action 5: Prompt Engineering

- Build System Prompt containing:
  - Extracted schema
  - 3–5 NL to DSL translation examples

Example Translation:

```
NL: Show me failed logins
DSL:
{
 "query": {
   "match": {
     "event.action": "logon-failure"
   }
 }
}
```

4 Phase 3: API Bridge (Week 3)

4.1 Objective

Connect AI-generated queries to SIEM data execution layer.

4.2 Action 6: Build Elasticsearch Connector

- Use elasticsearch-py client.
- Accept JSON DSL query from AI.
- Send query to Elasticsearch / Wazuh backend.

4.3 Core Responsibilities

- Query validation
- Error handling
- Response normalization
- Timeout and performance control

4.4 Action 7: Context Handling

- Implement LangChain Memory.
- Maintain multi-turn investigation context.

Example Context Flow

- Query 1: Show yesterday login attempts
- Query 2: Filter only VPN
- System retains original time filter.

5 Phase 4: Frontend and Visualization (Week 4)

5.1 Objective

Build demonstration-ready interface for prototype showcase.

5.2 Action 8: Streamlit User Interface

- Chat-style interface
- Display structured outputs

Visualization Methods

- st.table() for log datasets
- st.plotly_chart() for reports and analytics

5.3 Action 9: Agentic Reasoning Loop (Advanced)

- Optional advanced capability.
- Implement LangGraph agent flow.

Example Reasoning Chain

1. Detect user intent (malware investigation)
2. Query alerts index
3. Extract suspicious IPs
4. Perform secondary enrichment queries

6 Technology Stack

Component	Technology
SIEM Platform	Wazuh / Elastic SIEM
Backend Language	Python
AI Orchestration	LangChain / LangGraph
Frontend	Streamlit
Database / Search	Elasticsearch
API Communication	REST APIs

7 Prototype Success Metrics

- Natural Language to DSL accuracy rate.
- Multi-turn context retention performance.
- Query execution latency.
- Report generation correctness.
- System stability under repeated queries.

8 Risk Areas

- Schema mismatch across log sources.
- Ambiguous natural language interpretation.
- Query performance degradation under complex filters.
- Context drift across long conversations.

9 Future Extensions

- Reinforcement learning using analyst feedback.
- Automated anomaly reasoning layer.
- Cross-SIEM query federation.
- SOC workflow automation triggers.

10 Conclusion

This phased workflow enables rapid prototype development while ensuring strong integration between NLP intelligence and SIEM operational data. The approach prioritizes execution speed, modular scalability, and demonstration readiness.