Agentic Workflow for Automated Binary Analysis

Project Plan

Youness Anouar

January 25, 2026

Project Overview

Duration: 8 weeks
Repository: https://github.com/Uness10/Agentic-Workflow-for-Automated-Binary-Analysis

AI-powered system that automates security analysis of PE (.exe) and ELF binaries using LLM-orchestrated specialized tools built with the **Agno** framework and **FastMCP** servers.

Objectives:

1. Design and implement high-level analysis tools as MCP servers
2. Build intelligent agent orchestration for automated workflows
3. Detect malicious patterns, obfuscation, and suspicious behaviors
4. Generate comprehensive security reports with actionable insights
5. Package the entire system as reproducible Docker containers

Methodology

High-Level Approach

The system follows a **modular, agent-driven architecture** using the ReAct (Reasoning + Acting) paradigm:

1. **Input Processing:** Binary files (.exe/.elf) are submitted and validated (format detection, size checks)
2. **Reasoning:** The LLM-based agent analyzes the binary type and determines the optimal analysis strategy
3. **Tool Selection:** Agent dynamically selects which MCP tools to invoke based on context and intermediate findings
4. **Specialized Analysis:** Each MCP server performs focused, high-level analysis (semantic, not raw commands)
5. **Result Correlation:** Findings from multiple tools are aggregated, cross-referenced, and synthesized
6. **Report Generation:** A structured security report is produced with severity ratings and recommendations

The agent operates in an iterative loop—if initial analysis reveals suspicious patterns (e.g., packing detected), it triggers deeper investigation with additional tools.

System Architecture Diagram

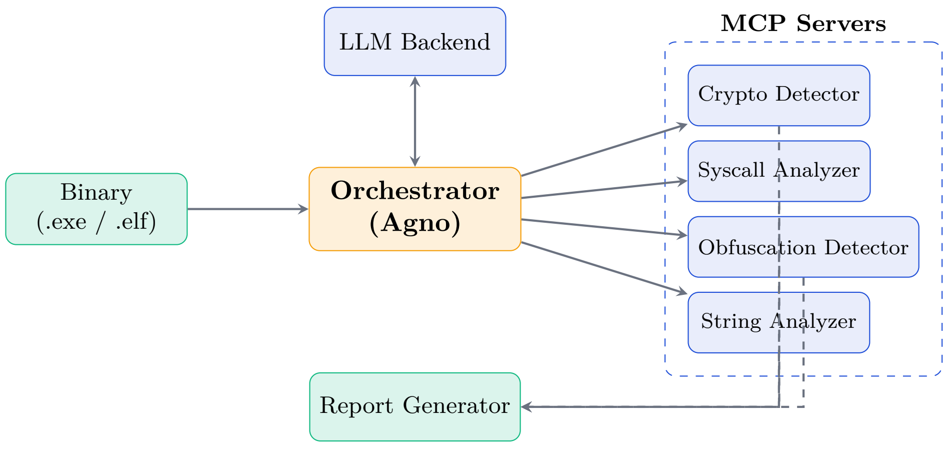

Figure 1: System Architecture — Agent orchestrates MCP tools via LLM reasoning

Key Components and Interactions

Component	Role & Interactions
Orchestrator Agent	Central coordinator (Agno framework). Receives binary input, queries LLM for reasoning, invokes MCP tools, aggregates results. Implements ReAct loop for iterative analysis.
LLM Backend	Provides reasoning capabilities. Agent sends context (binary metadata, partial findings) and receives tool selection decisions and analysis interpretations.
MCP Tool Servers	Five specialized analyzers exposed via FastMCP protocol. Each tool is stateless, accepts file path, returns structured JSON. Tools can be invoked in parallel or sequentially.
Report Generator	Consumes aggregated tool outputs. Applies severity scoring, generates Markdown/JSON reports with MITRE ATT&CK mappings and remediation suggestions.

Technology Stack Justification

Layer	Technology	Justification
Agent Framework	Agno	Modern Python framework with native tool orchestration, async support, and clean agent abstractions. Preferred over LangChain for simplicity.
Tool Protocol	FastMCP	Lightweight Model Context Protocol implementation. Enables clean tool exposure to LLM agents with structured I/O schemas.
Binary Analysis	Radare2, LIEF	Radare2: mature RE framework for deep analysis. LIEF: cross-platform binary parsing (PE/ELF/Mach-O) with Python bindings.
PE/ELF Parsing	pefile, pyelftools	Specialized libraries for header parsing, import tables, sections. More reliable than generic parsers for format-specific details.
LLM	OpenAI / Claude	State-of-the-art reasoning capabilities. Claude preferred for longer context; OpenAI for function calling reliability.
Containerization	Docker	Ensures reproducibility across environments. Critical for packaging complex dependencies (Radare2, native libs).

MCP Tools Specification

Tools are **high-level semantic analyzers** (not command wrappers). Each accepts a binary path and returns structured JSON.

1. Crypto Detector

- Identifies cryptographic constants (AES, RC4, RSA)
- Detects custom encryption patterns
- Returns: algorithm, confidence, location

2. Syscall Analyzer

- Maps imports to MITRE ATT&CK
- Detects suspicious API combinations
- Returns: behaviors, risk score, techniques

3. Obfuscation Detector

- Identifies packers (UPX, etc.)
- Analyzes entropy anomalies
- Returns: packer ID, evasion score

4. String Analyzer

- Extracts contextual strings
- Categorizes URLs, IPs, and commands
- Returns: categorized strings, risk flags

5. Metadata Extractor — Parses PE/ELF headers; extracts timestamps, compiler info, signatures, and flags anomalies.

Implementation Plan

Timeline with Milestones

Weeks	Milestone	Deliverables
1–2	Foundation	Repository structure, Dockerfile, FastMCP template, Agno skeleton, data models
3–4	Core Tools	Crypto, String, Syscall, Metadata MCP servers + unit tests
5–6	Orchestration	Obfuscation MCP, agent workflow, tool coordination, error handling
7–8	Delivery	Report generator, E2E testing, documentation, Docker Compose, demo

Task Breakdown and Assignments

Milestone 1 (Weeks 1–2): Foundation & Infrastructure

- Initialize Git repository with modular structure (`/agents`, `/tools`, `/core`)
- Create base Dockerfile with dependencies (Radare2, pefile, pyelftools, LIEF)
- Implement FastMCP server boilerplate with JSON schema definitions
- Build Agno agent skeleton with tool registration mechanism
- Design common data models: `BinaryInfo`, `AnalysisResult`, `Finding`
- Implement binary utilities: format detection (PE/ELF), file validation, hash computation

Milestone 2 (Weeks 3–4): Core MCP Tools

- Implement Crypto Detector: constant scanning, entropy-based detection, algorithm identification
- Implement String Analyzer: extraction, regex categorization, context retrieval
- Implement Syscall Analyzer: import table parsing, API-to-ATT&CK mapping, risk scoring
- Implement Metadata Extractor: header parsing, timestamp analysis, anomaly detection
- Write unit tests for each tool (pytest); validate with benign and malicious samples

Milestone 3 (Weeks 5–6): Advanced Tools & Orchestration

- Implement Obfuscation Detector: packer signatures, entropy analysis, anti-debug checks
- Integrate all MCP tools with Agno agent via FastMCP protocol
- Define agent system prompts and ReAct decision logic
- Implement tool selection strategy (parallel vs. sequential based on findings)
- Add result correlation: cross-reference findings, deduplicate, compute aggregate risk
- Handle errors: timeouts, malformed binaries, LLM failures with graceful fallbacks

Milestone 4 (Weeks 7–8): Integration & Delivery

- Design report schema: JSON for programmatic use, Markdown for human readability
- Implement severity scoring: weighted combination of tool findings
- Perform end-to-end integration testing with diverse binary corpus
- Finalize `docker-compose.yml` for multi-container orchestration

- Write `start.sh`: environment variable checks, dependency validation, service startup
- Complete documentation: README, API reference, usage tutorials, architecture diagrams
- Prepare demo: analyze sample malware, generate reports, record walkthrough

Dependencies and Risk Mitigation

Risk	**Impact**	**Mitigation Strategy**
LLM API rate limits	Workflow interruption	Implement response caching, exponential retry, local LLM fallback (Ollama)
Binary parsing failures	Tool crashes	Use multiple parsers (pefile + LIEF) with fallback chain; validate inputs
Large binary analysis	Timeouts, OOM	Chunked processing, configurable timeouts, memory limits in Docker
False positives/negatives	Inaccurate reports	Calibrate detection thresholds, ensemble scoring, confidence intervals
Docker build failures	Deployment blocked	Pin all dependency versions, use multi-stage builds, CI/CD validation

Deliverables & Success Criteria

Deliverables

- Python codebase and test suite
- Dockerfile and docker-compose.yml
- start.sh with environment validation
- README, API docs, tutorials
- Demo analysis reports

Success Criteria

- ☐ Five MCP tools fully functional
- ☐ Agent orchestrates multi-tool analysis
- ☐ Supports PE and ELF binaries
- ☐ Docker runs without intervention
- ☐ Reports contain actionable findings

Important Notes

The system architecture, technology stack, and project milestones are intended as a reference plan. They may be adjusted or refined as the project evolves to address new findings, integration challenges, or performance optimizations. All modifications will maintain the modular agent–MCP design and the principle of reproducible, containerized deployment.