SENG 696 – Agent-Based Software Engineering

Detailed Design Document

Multi-Agent Security-Risk Triage for C/C++ Cryptographic Code

Reyhaneh Farahmand
UCID: 30271200

November 2025

Contents

Chapter 1

Introduction and Detailed Design Overview

1.1 Purpose of this Document

This Detailed Design Document refines the GAIA-based preliminary design of the **Multi-Agent Security-Risk Triage System** into an implementable architecture. It focuses on:

- Agent architecture and internal responsibilities;
- Inter-agent communication mechanisms (protocols, messages, data formats);
- Behavioral models (use cases, sequences, activities);
- Data and knowledge sharing structures for security-risk scoring.

1.2 System Overview

The system is a **Multi-Agent System (MAS)** that triages security risk in C/C++ cryptographic code (e.g., AES, SHA, ECDSA) by combining:

1. AI-origin likelihood per file (p_{AI});
2. Static-analysis vulnerability findings;
3. A fusion policy that amplifies severity when AI-origin probability is high.

Input: Repository snapshot or pull request diff.
Output: Structured JSON artifacts and a human-readable ranked risk report.

1.3 Primary Agents

- **OrchestratorAgent:** Controls workflow and communication between agents.
- **AiOriginAgent:** Determines AI-origin likelihood per file.
- **VulnAgent:** Runs static analyzers and aggregates results.
- **SeverityAgent:** Computes composite risk scores and triage.
- **ReportAgent:** Generates final reports in multiple formats.

Chapter 2

Use Case Diagram of Agents

2.1 Overview

The following diagrams show how external actors (developers, CI/CD systems) interact with the MAS, and how agents cooperate internally.

2.2 OrchestratorAgent

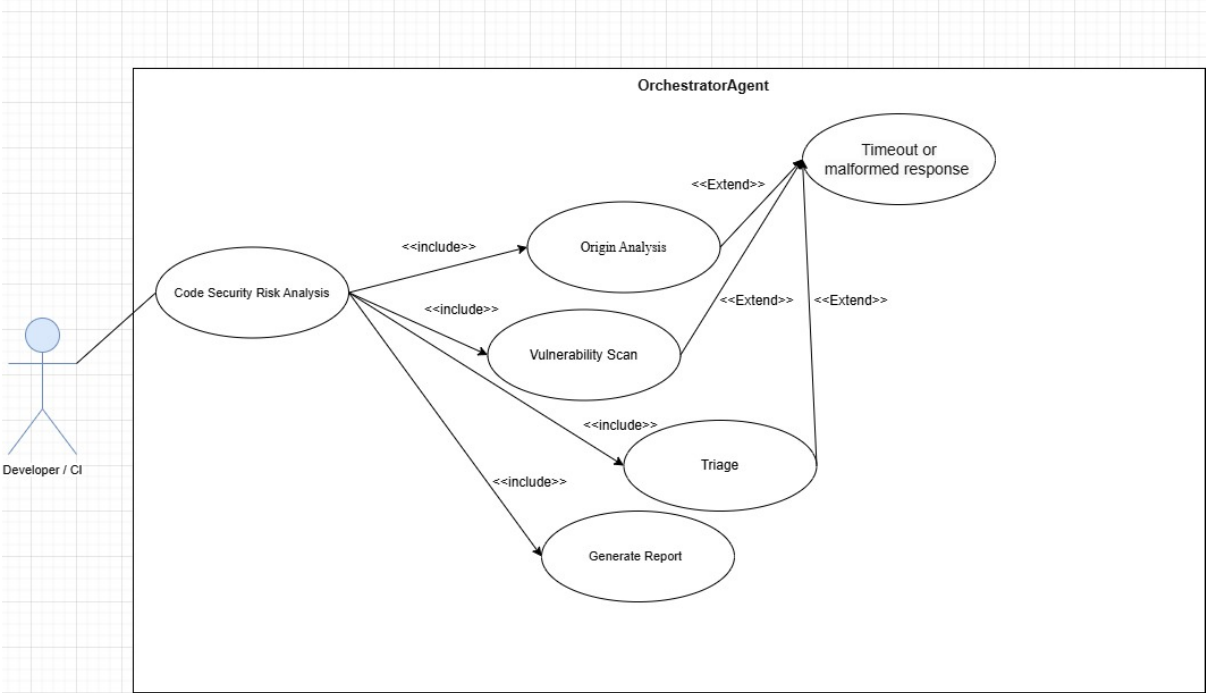


Figure 2.1: Use Case: OrchestratorAgent

2.3 AiOriginAgent

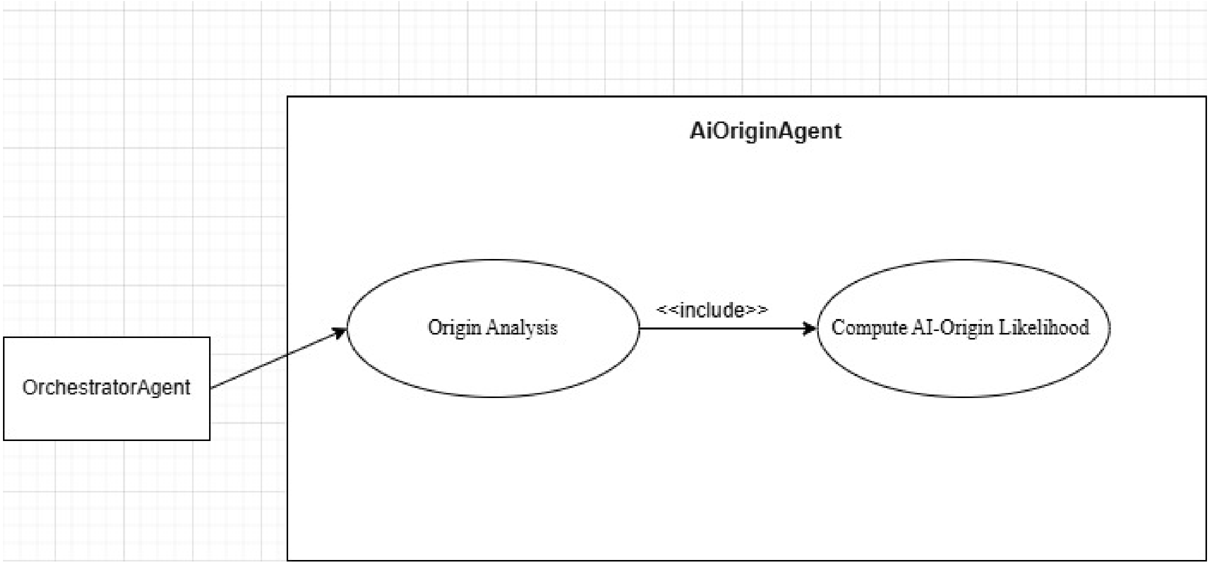


Figure 2.2: Use Case: AiOriginAgent

2.4 VulnAgent

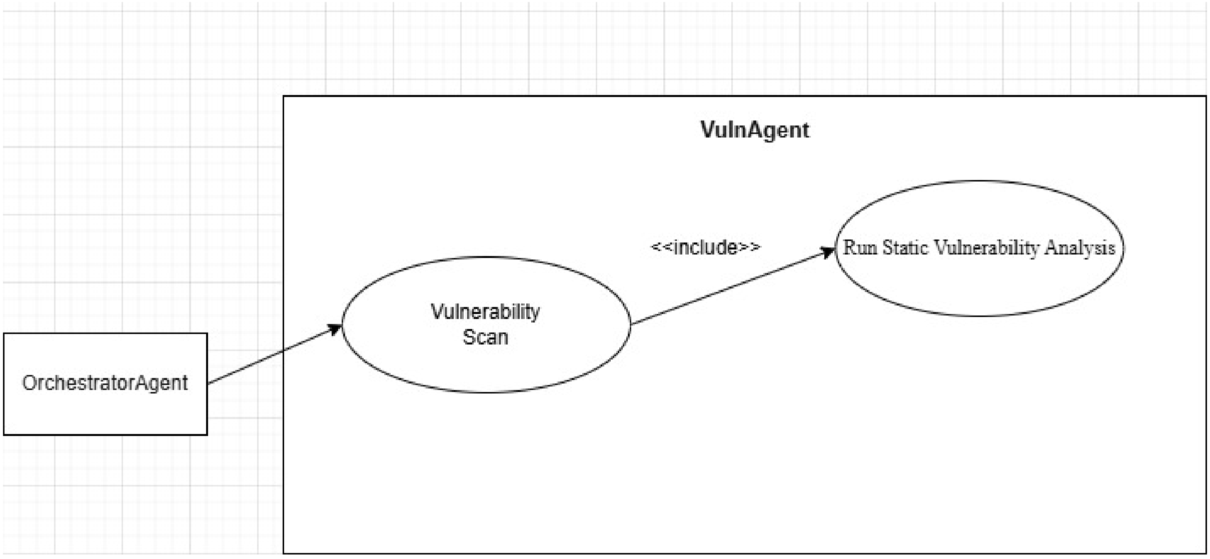


Figure 2.3: Use Case: VulnAgent

2.5 SeverityAgent

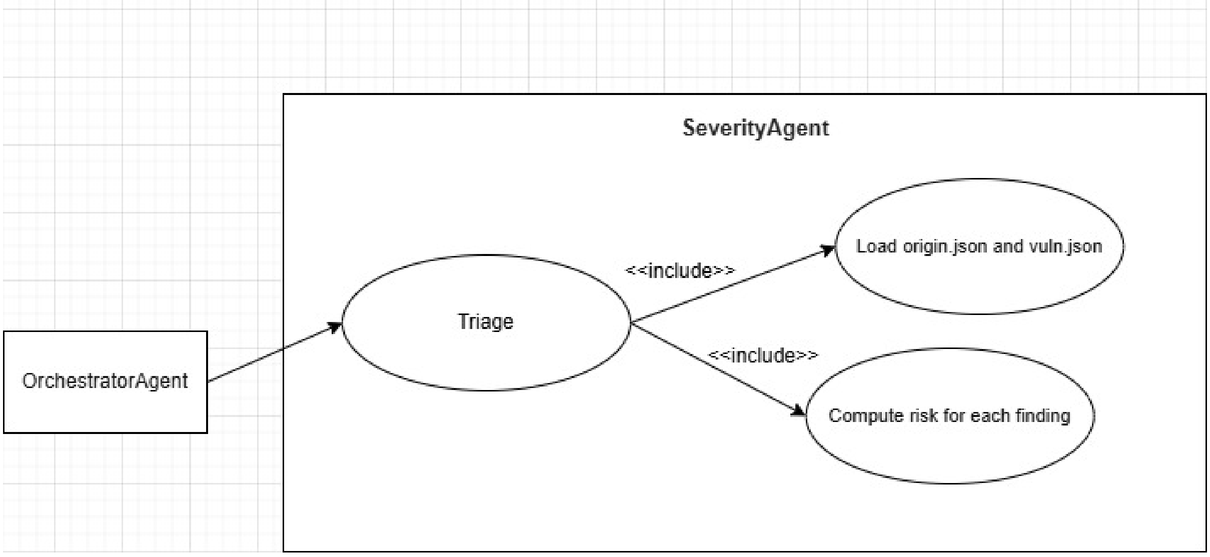


Figure 2.4: Use Case: SeverityAgent

2.6 ReportAgent

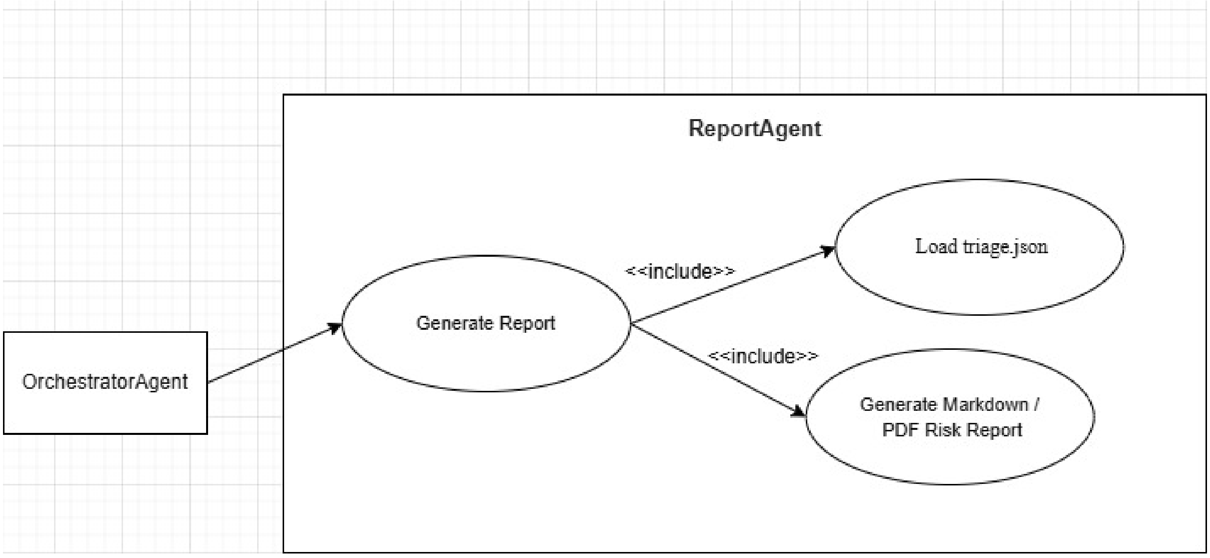


Figure 2.5: Use Case: ReportAgent

2.7 Whole System

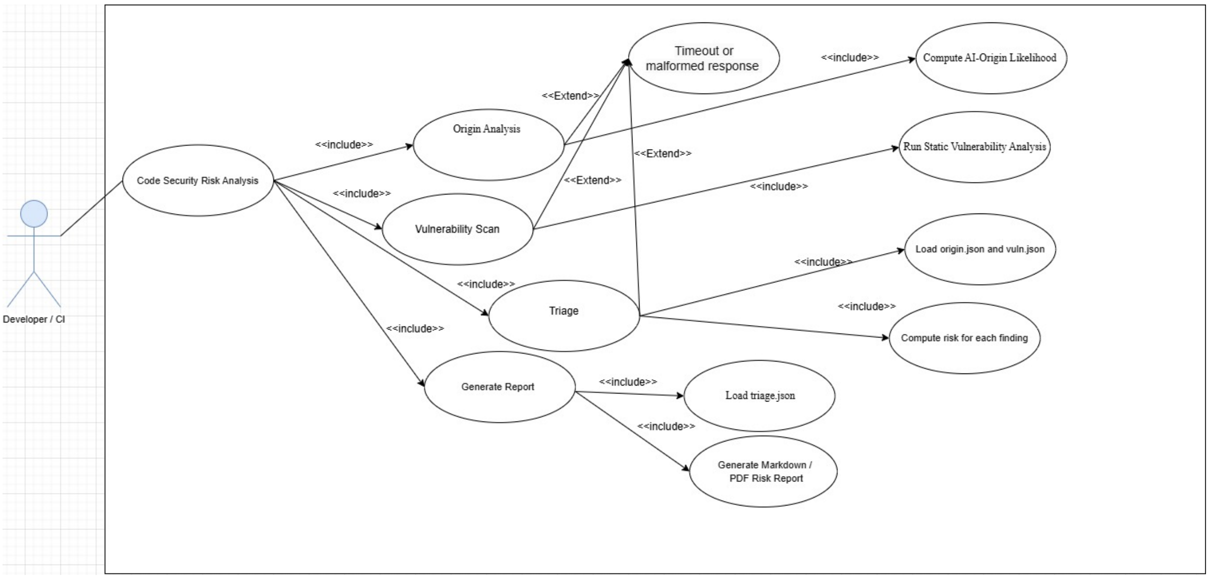


Figure 2.6: System-Level Use Case Diagram

Chapter 3

Class Diagram

Detailed Class Diagram

The following class diagram represents the static structure of the Multi-Agent Security-Risk Triage System. It includes all agents, shared components, and data entities.

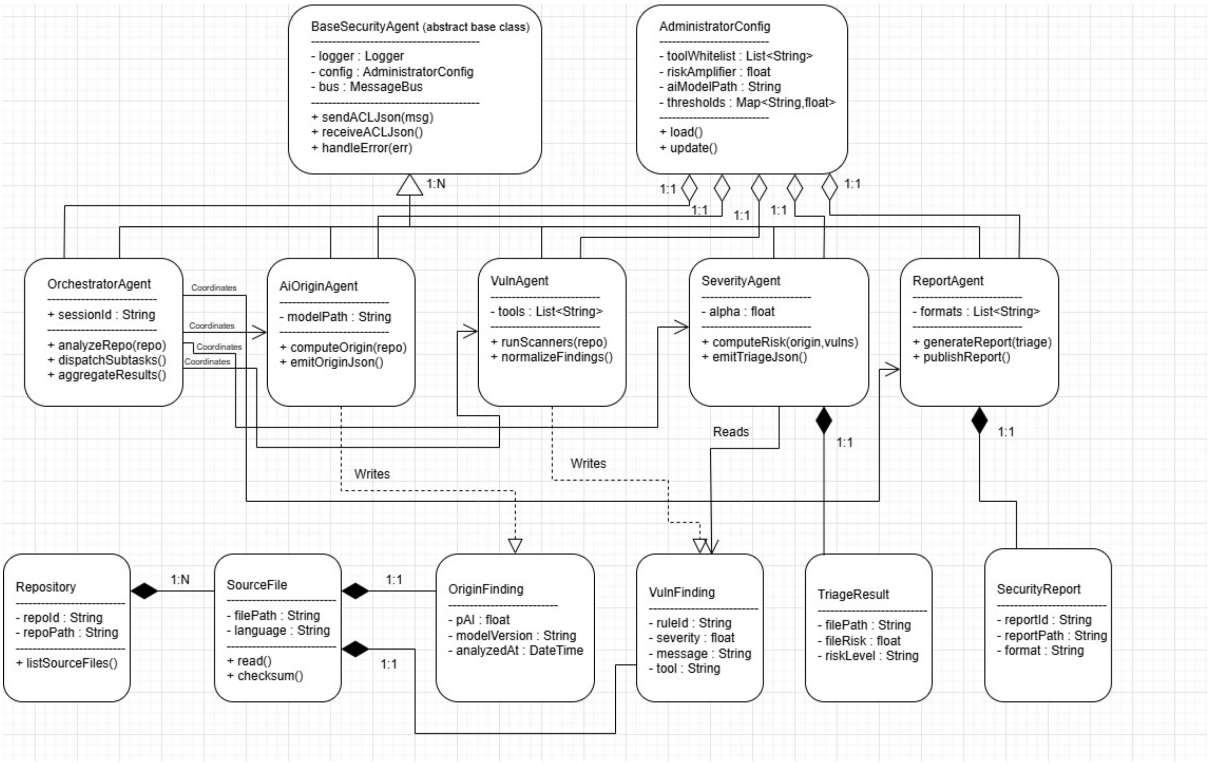


Figure 3.1: UML Class Diagram for Multi-Agent Security-Risk Triage System

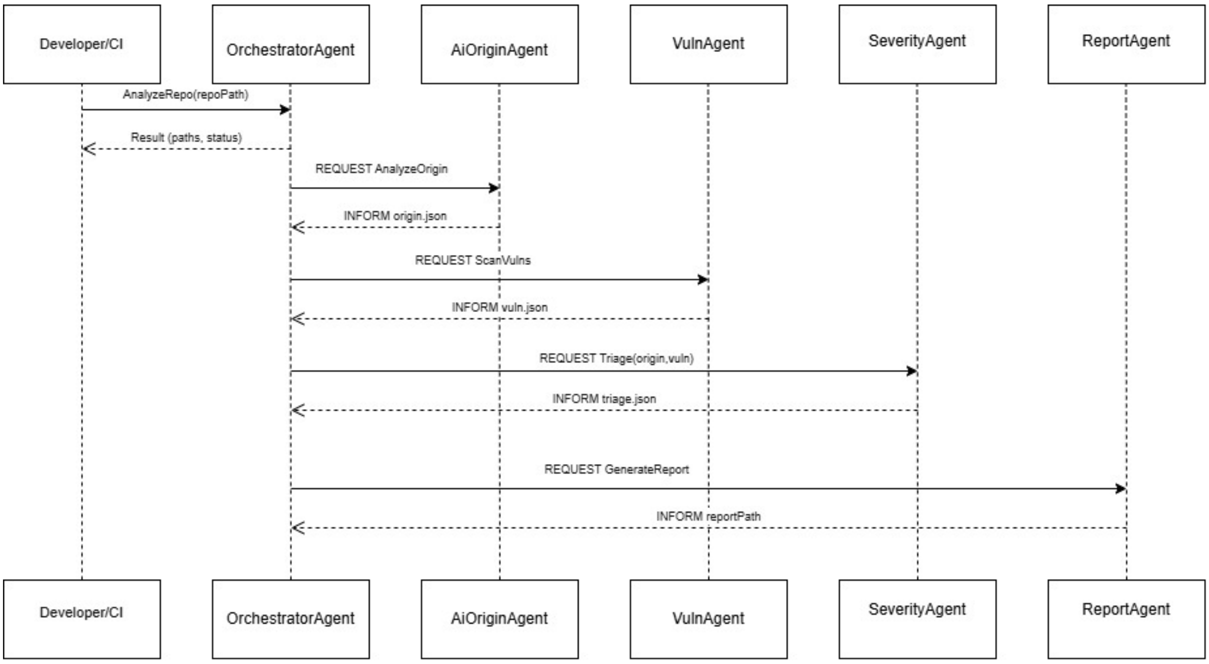
Chapter 4

Message Sequence Chart

4.1 Interaction Chart

The following sequence diagram captures the main interaction for a single analysis request.

- Logging and configuration;
- FIPA-ACL messaging;
- Error handling and message passing.



4.2 Message Exchange

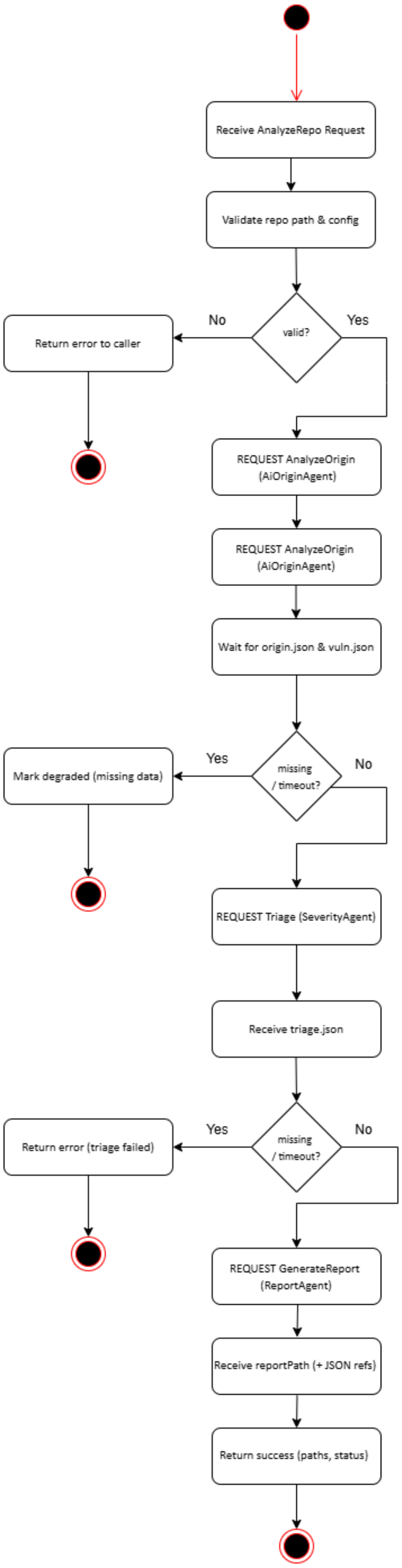
Agents use JSON-based FIPA-ACL performatives: `REQUEST`, `INFORM`, and `FAILURE`, ensuring interoperability between Java (JADE) and Python modules.

Chapter 5

Activity Diagrams

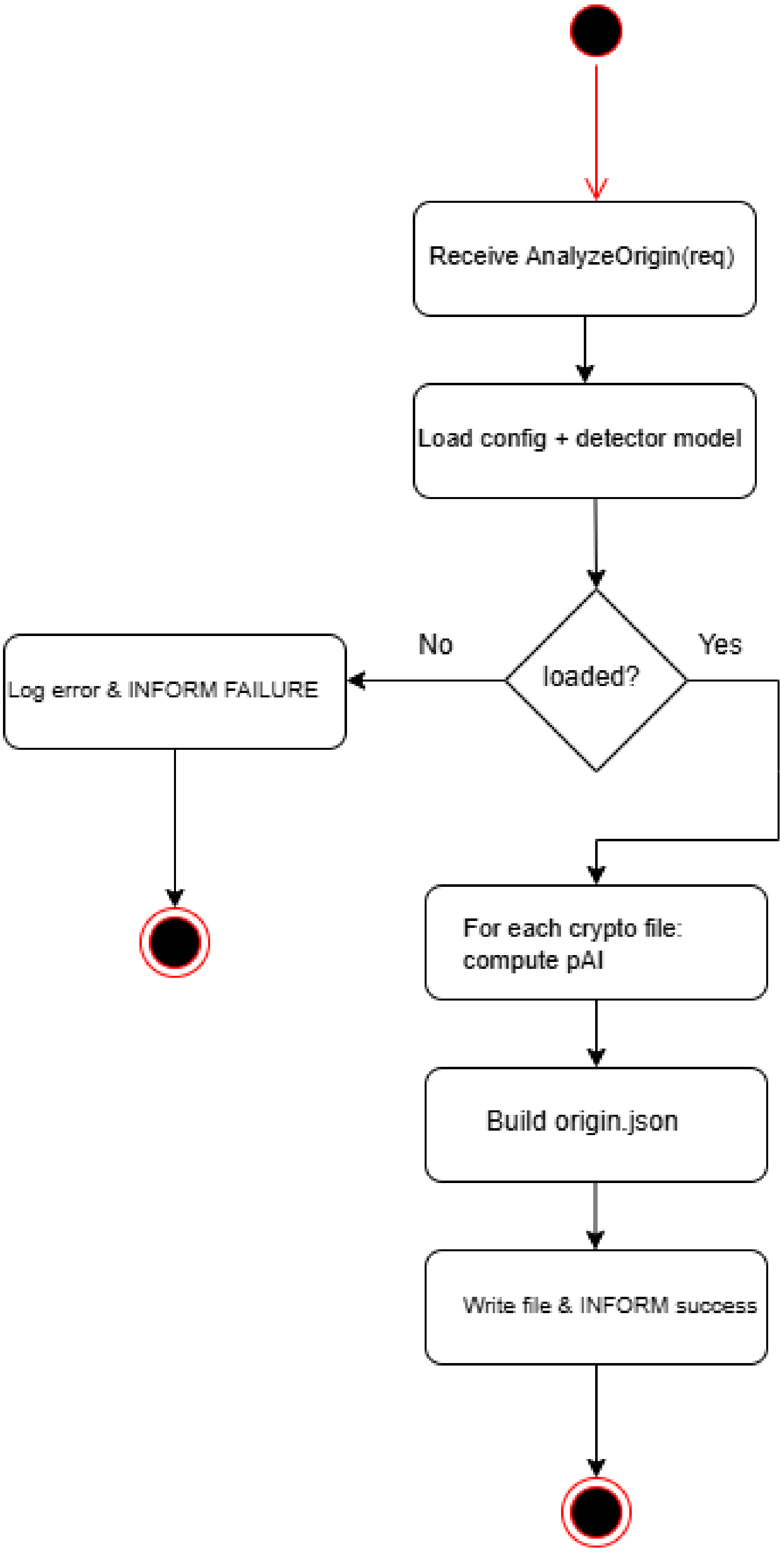
5.1 Repository Analysis Activity (End-to-End Workflow)

This diagram captures how a Developer/CI request is processed by the OrchestratorAgent, which delegates to AiOriginAgent, VulnAgent, SeverityAgent, and ReportAgent. Transient failures lead to controlled degraded modes or final error responses.


Receive AnalyzeRepo Request
Validate repo path & config
No
valid?
Yes
Return error to caller
REQUEST AnalyzeOrigin (AiOriginAgent)
REQUEST AnalyzeOrigin (AiOriginAgent)
Wait for origin.json & vuln.json
Yes
missing / timeout?
No
Mark degraded (missing data)
REQUEST Triage (SeverityAgent)
Receive triage.json
Yes
missing / timeout?
No
Return error (triage failed)
REQUEST GenerateReport (ReportAgent)
Receive reportPath (+ JSON refs)
Return success (paths, status)

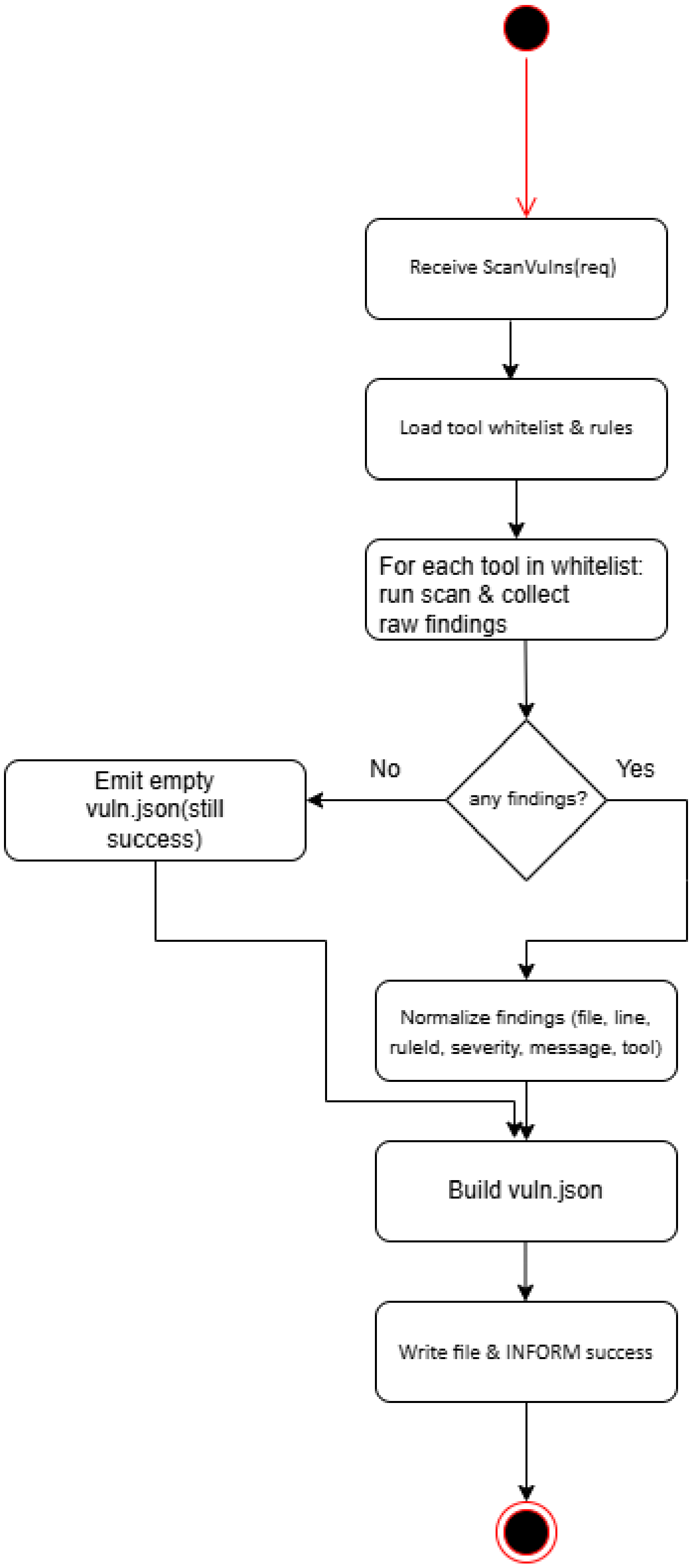
5.2 AI-Origin Analysis Activity (AiOriginAgent)

AiOriginAgent receives a request from the Orchestrator, loads its configuration and model, computes per-file pAI values for crypto-relevant files, and emits origin.json. Failures are logged and reported back to the Orchestrator.


Receive AnalyzeOrigin(req)
Load config + detector model
loaded?
No
Yes
Log error & INFORM FAILURE
For each crypto file:
compute pAI
Build origin.json
Write file & INFORM success

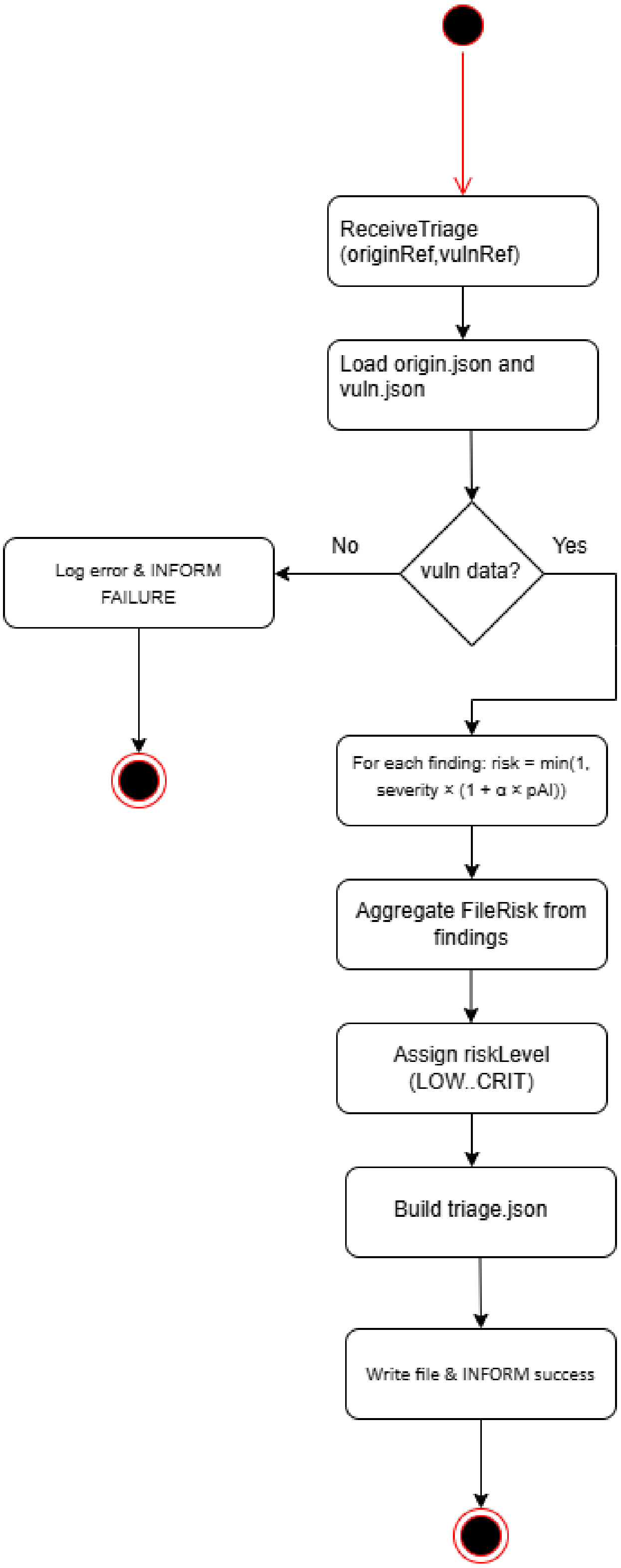
5.3 Vulnerability Scan Activity (VulnAgent)

VulnAgent runs whitelisted static analysis tools on the repository, aggregates and normalizes findings, and emits vuln.json. Failures per tool are tolerated; a global failure is reported if no usable results are produced.


Receive ScanVulns(req)
Load tool whitelist & rules
For each tool in whitelist: run scan & collect raw findings
any findings?
No
Yes
Emit empty vuln.json(still success)
Normalize findings (file, line, ruleId, severity, message, tool)
Build vuln.json
Write file & INFORM success

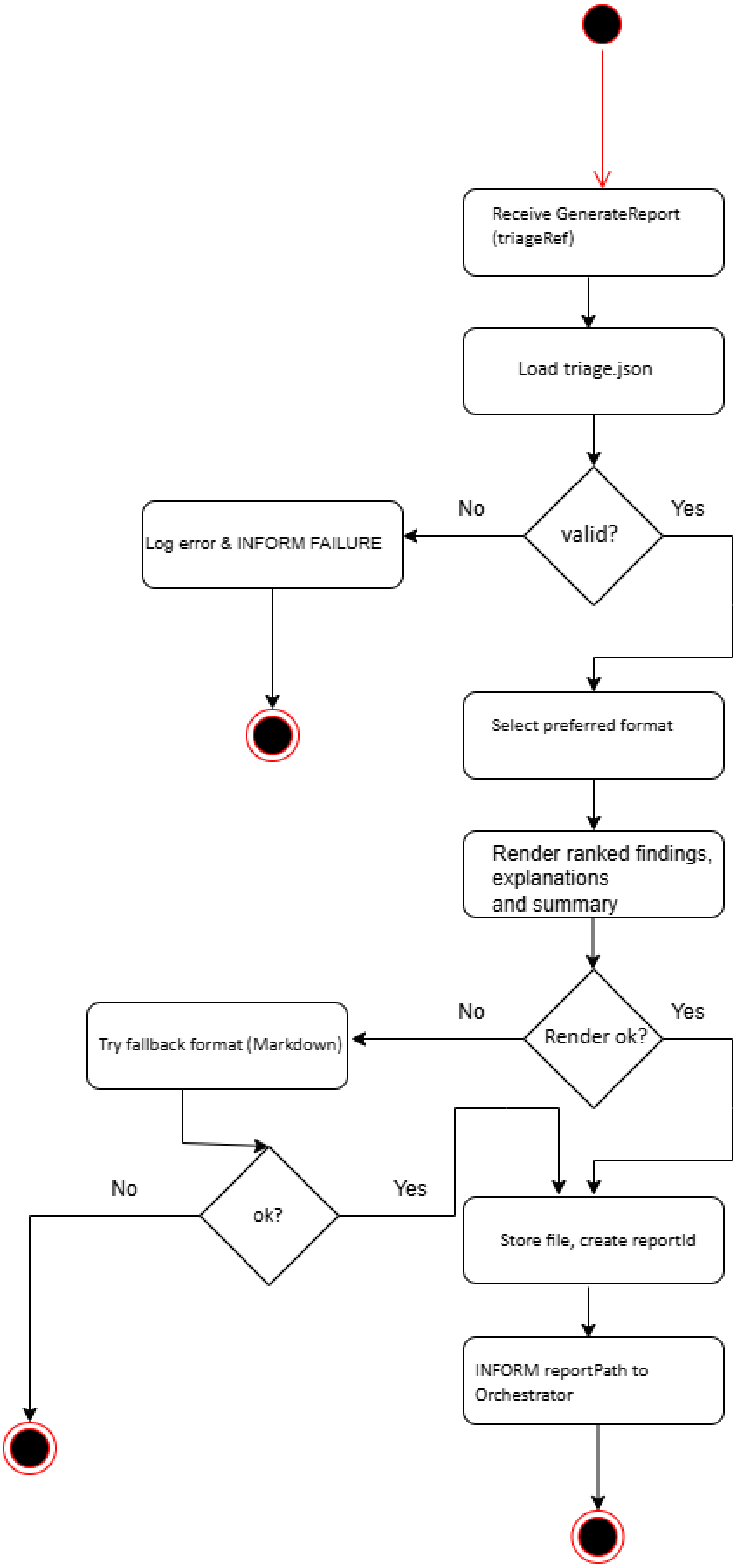
5.4 Risk Triage Activity (SeverityAgent)

SeverityAgent fuses origin and vulnerability data. It tolerates missing origin information (by disabling AI amplification), assigns risk levels, and emits triage.json.


ReceiveTriage (originRef,vulnRef)
Load origin.json and vuln.json
vuln data?
No
Yes
Log error & INFORM FAILURE
For each finding: risk = min(1, severity × (1 + α × pAI))
Aggregate FileRisk from findings
Assign riskLevel (LOW..CRIT)
Build triage.json
Write file & INFORM success

5.5 Report Generation Activity (ReportAgent)

ReportAgent consumes triage.json, renders a ranked security report, and exposes its location. If rendering fails for the preferred format, it falls back to a simpler format before signaling failure.


Receive GenerateReport (triageRef)
Load triage.json
No
valid?
Yes
Log error & INFORM FAILURE
Select preferred format
Render ranked findings, explanations and summary
No
Render ok?
Yes
Try fallback format (Markdown)
No
ok?
Yes
Store file, create reportId
INFORM reportPath to Orchestrator

Chapter 6

Data Specification

The Multi-Agent System (MAS) exchanges structured JSON artifacts that capture the outputs of the AiOriginAgent, VulnAgent, and SeverityAgent, representing AI-origin estimates, vulnerability findings, and triage decisions, respectively. Each agent produces or consumes these files in a standardized format, ensuring traceability, reproducibility, and easy integration with external analysis or reporting tools.

6.1 JSON Schemas (Conceptual)

The following schemas illustrate the structure of the primary JSON artifacts exchanged between agents. Each schema represents a conceptual data model rather than a strict implementation specification. Field names and datatypes are consistent with the system's design.

6.1.1 `origin.json`

The origin.json file contains per-file AI-origin likelihood estimates produced by the AiOriginAgent. Each record includes the file path, AI-origin probability (p_{AI}), the model version, and the timestamp of analysis.

```
[
  {
    "filePath": "src/aes.c",
    "pAI": 0.82,
    "modelVersion": "aes-detector-v1.2",
    "analyzedAt": "2025-10-01T14:23:00Z"
  }
]
```

6.1.2 `vuln.json`

The vuln.json file represents normalized static analysis results generated by the VulnAgent. Each record corresponds to a detected vulnerability, including the affected file, line number, rule identifier, severity, and message.

```
[
  {
    "filePath": "src/aes.c",
    "line": 128,
    "ruleId": "CWE-327",
    "severity": "HIGH",
    "message": "Use of weak or custom crypto mode"
  }
]
```

6.1.3 `triage.json`

The triage.json file captures the final fused results produced by the SeverityAgent, integrating both AI-origin probabilities and vulnerability severities into a unified risk assessment per file.

```
[
  {
    "filePath": "src/aes.c",
    "fileRisk": 0.93,
    "riskLevel": "CRITICAL",
    "findings": [
      {
        "ruleId": "CWE-327",
        "baseSeverity": 0.85,
        "pAI": 0.82,
        "risk": 0.93,
        "explanation": "High severity crypto issue amplified by high AI-origin likelihood."
      }
    ]
  }
]
```

6.2 Artifact Relationships

These JSON artifacts follow the 1:1 relationships established in the class diagram: each `SourceFile` instance produces exactly one `OriginFinding` and one `VulnFinding`, both of which contribute to a single `TriageResult`. The ReportAgent consumes the triage results and generates a corresponding `SecurityReport` in a human-readable format (e.g., Markdown or PDF). This ensures clear traceability between each agent's outputs and their downstream consumers.

Artifact	**Produced by**
`origin.json`	AiOriginAgent — AI-origin likelihoods per file
`vuln.json`	VulnAgent — normalized vulnerability findings
`triage.json`	SeverityAgent — fused and ranked risk results
`SecurityReport`	ReportAgent — human-readable ranked output

6.3 Integration and Traceability

Each artifact is timestamped and versioned, allowing complete traceability throughout the pipeline. A single repository analysis session thus generates a reproducible chain:

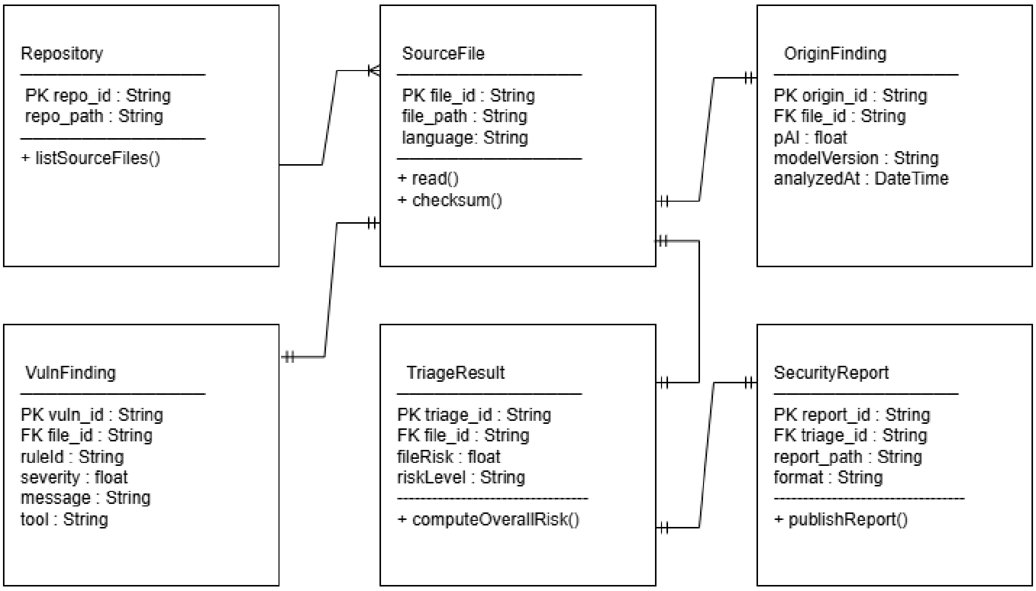
$$\texttt{origin.json} \rightarrow \texttt{vuln.json} \rightarrow \texttt{triage.json} \rightarrow \texttt{SecurityReport}.$$

This design supports both automated pipelines and manual review workflows, ensuring auditability and consistency across multiple analysis runs.

Chapter 7

Data/Knowledge Sharing Specification

7.1 E-R Diagram


Repository
PK repo_id : String
repo_path : String
+ listSourceFiles()
SourceFile
PK file_id : String
file_path : String
language: String
+ read()
+ checksum()
OriginFinding
PK origin_id : String
FK file_id : String
pAI : float
modelVersion : String
analyzedAt : DateTime
VulnFinding
PK vuln_id : String
FK file_id : String
ruleId : String
severity : float
message : String
tool : String
TriageResult
PK triage_id : String
FK file_id : String
fileRisk : float
riskLevel : String
+ computeOverallRisk()
SecurityReport
PK report_id : String
FK triage_id : String
report_path : String
format : String
+ publishReport()