# Individual Report for Group 9

## CyberSage: Multi Ai Agent system for Cyber Threat Intelligence.

## Author: Lay Wee Lye

Contents

[Individual Report 1](#_Toc210682728)

[1. Introduction 3](#_Toc210682729)

[2. Background and Motivation 3](#_Toc210682730)

[3. Main scope of work 3](#_Toc210682731)

[4. Objectives 3](#_Toc210682732)

[5. Review of existing systems 4](#_Toc210682733)

[a. Existing CTI Ingestion Systems 4](#_Toc210682734)

[b. Complexity of ingestion systems 4](#_Toc210682735)

[c. High cost of Threat Intelligence feeds 4](#_Toc210682736)

[d. Adaptive AI in Data Normalization 4](#_Toc210682737)

[6. Design and architecture of Ingest Agent 5](#_Toc210682738)

[7. Data Flow Summary 5](#_Toc210682739)

[8. Core Architectural Components and Challenges 6](#_Toc210682740)

[9. Challenges faced 8](#_Toc210682741)

[ Architecture Complexity 8](#_Toc210682742)

[ Support for multi-formats 8](#_Toc210682743)

[ IOC Extraction 8](#_Toc210682744)

[ CVE Enrichment 9](#_Toc210682745)

[ Deduplication 9](#_Toc210682746)

[ Testing infrastructure 9](#_Toc210682747)

[10. Notable Accomplishments 9](#_Toc210682748)

[Agentic Orchestration Innovation 9](#_Toc210682749)

[Multi-Format Ingestion 9](#_Toc210682750)

[High-Quality IOC Extraction 10](#_Toc210682751)

[Comprehensive CVE Enrichment 10](#_Toc210682752)

[MITRE ATT&CK Mapping 10](#_Toc210682753)

[11. Features and Improvements 10](#_Toc210682754)

[ Parallel RSS feed processing 10](#_Toc210682755)

[ Parsing STIX and MISP more accurately. 10](#_Toc210682756)

[ Automatic creation of YARA rules 10](#_Toc210682757)

[ LLM Guardrails to Prevent Prompt Injection Attacks 10](#_Toc210682758)

## Introduction

CyberSage is a multi-agent AI system for Cyber Threat Intelligence, intended to:

* **Bridging the Cybersecurity Talent Gap:** Due to a shortage of skilled cybersecurity professionals, organizations struggle to analyse and respond to threats effectively. Automating the first level of threat analysis reduces the workload and improves response times.
* **Cost Efficiency:** Hiring dedicated threat intelligence analysts is costly. The proposed platform offers an affordable alternative by automating initial threat assessments using AI-driven analysis.
* **Timely and Scalable Analysis:** The platform will continuously monitor open-source intelligence, ensuring organizations stay ahead of emerging threats and reducing the reliance on delayed human analysis.
* **Improved Threat Awareness:** The platform will help organizations identify vulnerabilities in their technology stacks, monitor threat actors targeting their industry, and assess geopolitical risks associated with state-sponsored cyber activities.
* **Actionable Intelligence:** By structuring the threat intelligence output in JSON format, organizations can seamlessly integrate the platform’s results into their existing security operations.

## Background and Motivation

The rapid digitalization of industries has significantly increased the volume and sophistication of cyber threats worldwide. However, the cybersecurity workforce has not grown at the same rate, resulting in a severe global talent shortage.

Organizations often depend on human analysts to interpret threat reports, extract indicators of compromise (IOCs), and contextualize adversarial behaviours. This manual process is slow, inconsistent, and expensive. The absence of automation leads to delayed detection and poor scalability, especially in small and medium enterprises that cannot afford dedicated CTI teams.

## Main scope of work

The project is made up of 4 components. This report details the work done on Agent A, the intelligence ingestion module, automate the **acquisition, verification, enrichment, and normalization** of intelligence artifacts. It represents the system’s first stage of data processing, ensuring that subsequent analytical agents (B–D) receive clean, structured, and deduplicated content.

## Objectives

Agent A was designed to:

* Continuously acquire intelligence from diverse OSINT and structured sources.
* Detect and classify data format (RSS, STIX, MISP, HTML, JSON, text).
* Normalize data into CyberSage’s unified JSON schema.
* Validate and enrich source metadata (timestamps, URLs, titles).
* Provide reliable, structured output to downstream agents.

## Review of existing systems

### Existing CTI Ingestion Systems

Traditional CTI ingestion relies on rule-based connectors or static parsers (e.g., MISP Feed, TAXII 2.1, OpenCTI). These systems are robust for predefined schemas but struggle with unstructured or evolving sources.

### Complexity of ingestion systems

Despite the availability of established CTI platforms such as MISP, OpenCTI, and Anomali ThreatStream, the ingestion and normalization of threat intelligence remain highly complex. These platforms typically require users to configure feeds, define schemas, and manage synchronization through manual TAXII or RESTintegrations. For instance, MISP supports diverse data formats (STIX, JSON, CSV, XML) but demands extensive configuration and administrative knowledge to manage event synchronization, taxonomy mapping, and object correlation.  
As a result, many organizations, especially small and medium enterprises (SMEs) struggle to operationalize these tools effectively.

Analysts often face **steep learning curves**, inconsistent data ingestion pipelines, and difficulties maintaining **format compatibility** across different feeds. This complexity limits scalability and reduces the timeliness of intelligence dissemination.

### High cost of Threat Intelligence feeds

Commercial threat intelligence feeds offered by vendors such as Recorded Future, Anomali, and ThreatConnect provide rich contextual data on threat actors, campaigns, and indicators of compromise (IOCs). However, these services often come with **high subscription costs**, making them inaccessible to many small and medium-sized organizations. Licensing models are typically based on **per-seat access**, **data volume**, or **API usage quotas**, which can quickly escalate operational expenses.

### Adaptive AI in Data Normalization

Recent advances in AI-driven parsing (LangChain, LlamaIndex) have enabled context-aware content extraction and classification. Few systems, however, integrate multi-agent orchestration for autonomous data ingestion in CTI.

## Design and architecture of Ingest Agent

The system is designed around a **LangGraph-based Orchestrator** that governs task execution through a **state-machine flow**, coordinating multiple **AI agents** and **MCP (Model Control Plane) tools**. The design ensures modularity, scalability, and interoperability across diverse intelligence data formats (STIX, MISP, RSS, HTML, JSON, PDF, TEXT).

A diagram of a document

AI-generated content may be incorrect.

Fig 1. System Architecture

## Data Flow Summary

1. **Input**: A URL or feed item is submitted to the Orchestrator.
2. **Detection**: The Orchestrator queries MCP’s detect\_content\_type to determine format (MISP/STIX/HTML/PDF/etc.).
3. **Delegation**: Based on type, it selects the corresponding parsing agent from the Agent Repository.
4. **Parsing & Normalization**: The chosen agent calls relevant MCP tools (e.g., parse\_stix\_contents) and returns a structured JSON.
5. **Enrichment**: Analytical agents (EntityAgent, IOCAgent, CVEAgent, etc.) process the normalized content for deeper insights.
6. **Validation & Storage**: The ValidatorAgent ensures compliance with the fixed JSON schema; StoreAgent persists results to the CTI database.
7. **Output**: The final normalized intelligence artifact is made available for downstream analytics or orchestration.

## Core Architectural Components and Challenges

**a. Input Layer**

* **URL Item**: Represents an input source, typically an OSINT feed or CTI report URL.
* Each URL acts as an independent task unit, enabling concurrent ingestion and processing.
* The input must be flexible and allow various formats:
  + **STIX and MISP Events** are well structured and formatted intelligence content. Intelligence content can contain too many details and require interpretation to make it useful.
  + **HTML documents** often contain irrelevant information and data (Navigation ads, Stylesheets, JavaScripts). This could not be hard coded. Using LLM to interpret HTML code to filter out irrelevant sections is a possible option but large pages with a lot of content will use a lot of tokens.
  + **PDF documents** need to be extracted to plain text format.
  + **Text** are freeform words which require to be interpreted.
  + **JSON files** are considered semi-structured because while the data is structured, the format is unknown and must be interpreted.
  + **RSS / ATOM feeds**. These feeds are ubiquitous and very commonly provided by news media, and threat intelligence vendors:
    - https://www.straitstimes.com/news/singapore/rss.xml
    - https://www.bleepingcomputer.com/feed/
    - https://feeds.bloomberg.com/markets/news.rss
* Many feed aggregators support RSS / ATOM feeds, e.g Feedly, Inoreader, and are very low cost compared to Threat Intelligence Feeds.

**b. Orchestrator (LangGraph State Machine)**

* Serves as the **central decision-making and control layer**.
* Uses a **state graph model** to define conditional flows between agents (e.g., Fetch → Parse → Enrich → Normalize).
* Responsible for:
  + Invoking appropriate agents based on content type.
  + Enforcing the **Fixed JSON Schema** for data normalization.
  + Maintaining task context across sequential operations.
* The Orchestrator ensures consistency and error recovery, similar to an event-driven microservice coordinator.

**c. Fixed JSON Schema**

* Acts as the **canonical data model** for structured intelligence output.
* Ensures uniformity across diverse sources and formats (MISP, STIX, RSS, PDF).
* Fields typically include title, source\_url, published\_date, iocs, threat\_actor, cve\_list, and summary.
* This schema enables downstream integration with CTI databases or SIEM/XSOAR systems.

**d. Agent Repository**

* A modular collection of specialized agents, each designed for a specific intelligence processing task.
* **Parsing Agents:**
  + MISPParseAgent, STIXParseAgent, PDFParseAgent, TextParseAgent, JSONNormaliseAgent handle raw data extraction and parsing logic.
* **Analysis Agents:**
  + SummarizeAgent, EntityAgent, IOCAgent, CVEAgent, MITREAgent, GeoMotivationAgent, CyberAgent perform semantic interpretation and enrichment using LLM reasoning.
* **Validation & Storage Agents:**
  + ValidatorAgent, StoreAgent ensure schema compliance and persistence into storage layers.

This separation of concerns promotes **maintainability** and **parallel development**, allowing individual agents to be upgraded or replaced without affecting others.

**e. MCP Server**

* Provides the **execution environment for deterministic tools** accessible to agents and orchestrator.
* Implements low-level operations such as:
  1. fetch\_contents : retrieves raw data from a URL or API.
  2. detect\_content\_type : determines file or feed type (RSS, STIX, PDF, HTML, etc.).
  3. parse\_\*\_contents : performs structured parsing of known formats.
  4. pdf\_extract : handles OCR and text extraction from PDFs.

This MCP layer **decouples tool execution from agent reasoning**, allowing deterministic tasks (e.g., parsing, validation) to run independently of LLM calls. It acts as a “toolbox” that can be reused by any agent in the system.

**f. Agents (Fetch, Parse)**

* Represent the active runtime instances that perform specific data operations delegated by the Orchestrator.
* These agents interact dynamically with the MCP server to retrieve, analyze, and normalize intelligence data in real-time.

## Challenges faced

### Architecture Complexity

* **LLM Decision-Making Overhead**: The agentic orchestrator adds 3-5 LLM decision calls per pipeline run, increasing latency and token costs. The system must balance between autonomous decision-making and deterministic execution.
* **State Management Complexity**: Managing GraphState across 19 specialized agents with proper token tracking, validation, and error handling requires careful coordination.
* **Mandatory Sequence Enforcement**: Overriding LLM decisions to prevent illogical operations (e.g., extracting before fetching) adds code complexity but is necessary for reliability.

### Support for multi-formats

* **Content Type Detection Ambiguity**: JSON content could be STIX, MISP, or generic JSON. Required building a scoring system with STIX validator integration and heuristic patterns.
* **STIX Validation Challenges**: STIX 2.x bundles have complex validation requirements. Had to integrate stix2-validator but also maintain fallback parsers for malformed content.
* **RSS Feed Special Handling**: RSS feeds require recursive processing where each item URL goes through the full CTI pipeline again, creating potential for infinite loops if not handled carefully.

### IOC Extraction

* **False Positive Problem**: Initial regex-based extraction captured example domains (example.com), documentation IPs (192.168.x.x), and file extensions mistaken for domains (.exe, .dll).
* **Two-Stage Filtering Needed**: Had to implement validators library + tldextract (Public Suffix List) pre-filtering before LLM filtering to reduce token costs and improve accuracy.
* **Defanged IOC Handling**: Threat reports use defanged formats (hxxp://, example[.]com) requiring normalization before validation.

### CVE Enrichment

* **NVD API Rate Limiting**: NVD enforces 6-second delays between requests. Processing 10 CVEs takes 60+ seconds. Had to implement in-memory caching and offline KEV cache.

### Deduplication

* **Content Hash Collisions**: Initial URL-based deduplication failed when same content appeared at different URLs. SHA-256 content hashing solved this but required careful URL canonicalization.
* **RSS Feed URL Tracking**: Each RSS item URL needs independent tracking while maintaining relationship to parent feed.

### Testing infrastructure

* **MCP Server Lifecycle**: Tests requiring MCP server needed complex fixture management (session-scoped server startup/shutdown).
* **LLM Non-Determinism**: Testing LLM-based agents (EntityAgent, MITREAgent) requires mock responses or actual API calls with validation of result structure, not exact content.

## Notable Accomplishments

### Agentic Orchestration Innovation

* **Hybrid Decision-Making**: Successfully combined LLM autonomy with deterministic safety overrides. The orchestrator can adapt to content (skip CVE enrichment if no CVEs found) while enforcing mandatory sequences (fetch → detect → parse).
* **JSON Completeness Tracking**: Real-time monitoring of 30+ JSON fields with proactive capability selection to fill gaps prevents premature termination.
* **Self-Correcting Logic**: Orchestrator detects incomplete processing (e.g., extraction attempted without parsing) and automatically corrects course.

### Multi-Format Ingestion

* **7 Content Types Supported**: MISP, STIX 2.x, RSS/Atom, HTML, PDF, JSON, plain text with intelligent routing. Multi content types are enough to ensure threat intelligence through OSINT sources, not paid sources. Also supports paid feeds in MISP and STIX formats and threat intelligence reports in PDF format.
* **STIX Validator Integration**: Only CTI pipeline using stix2-validator for definitive STIX detection.
* **RSS Recursive Processing**: Cleanly handles feeds with 50+ items, processing each through full pipeline while maintaining feed metadata.

### High-Quality IOC Extraction

* **Three-Stage Filtering**: Regex → validators/tldextract → LLM filtering achieves 95%+ precision on test data.
* **Public Suffix List Integration**: Using tldextract with PSL prevents false positives like "file.exe" being treated as domain.
* **Automatic Defanging**: Seamlessly handles hxxp://, [.]com, [:]// formats common in threat reports.

### Comprehensive CVE Enrichment

* **NVD + CISA KEV Integration**: Combines NVD CVSS scores (v3.1 > v3.0 > v2.0 priority) with CISA Known Exploited Vulnerabilities for active exploitation status.
* **CPE Parsing**: Extracts affected products from CPE 2.3 strings with version range.

### MITRE ATT&CK Mapping

* **Context-Aware Mapping**: LLM analyzes threat text + extracted entities (malware, actors) + IOCs to map techniques with evidence snippets.
* **Singapore/ASEAN Focus**: Dedicated GeoMotivationAgent analyzes regional impact with confidence scores, motivations (espionage, financial, sabotage), and strategic implications.

## Features and Improvements

### Parallel RSS feed processing

* + The current implementation processes each RSS item sequentially. Using asynchronous or parallel execution can vastly improve performance.

### Parsing STIX and MISP more accurately.

* + The current implementation does not take relationships between items. This can be parsed and stored in a graph database which can allow more complex queries.

### Automatic creation of YARA rules

* + Generate YARA rules which can be used in SIEMs.

### LLM Guardrails to Prevent Prompt Injection Attacks

* + One of the key challenges identified during testing was the **risk of prompt injection**, where malicious or manipulated input content could alter the behaviour of the large language model (LLM) or cause it to execute unintended actions.
  + Since **Agent A** processes raw data from untrusted OSINT and web sources, adversaries could potentially embed hidden instructions within text fields or HTML content that, if interpreted naively, could override the system’s reasoning context or produce hallucinated results.
  + **Implemented Safeguards**
    - **Isolation of Input Contexts**  
      All external content ingested by the system is handled strictly as **user-level prompts**, never as **system-level instructions**.  
      This design ensures that untrusted data cannot modify the underlying operational behaviour, orchestration logic, or security policies of CyberSage.
  + **Guardrail Framework Integration**  
    The project experimented with **off-the-shelf guardrail validators** from [Guardrails AI Hub](https://hub.guardrailsai.com/) to test automated defences against prompt manipulation.  
    Two validators were evaluated:
  + **“Unusual Prompt”**: detects anomalous or contextually inconsistent input structures that may indicate injection attempts.
  + **“Detect Jailbreak”:** flags known jailbreak patterns such as prompt-reversal or obfuscated LLM instructions.
  + **Findings**
    - During evaluation, the Guardrails library frequently **flagged legitimate cybersecurity text** (e.g., exploit code fragments, malware analysis strings, or command-line indicators) as *unsafe*.  
      This indicates that generic LLM guardrails, while useful for general use-cases, require **domain-specific tuning** to distinguish between *malicious text* and *benign security intelligence samples*.
  + **Proposed Enhancements**
    - **Custom Guardrail Fine-Tuning:**  
      Develop bespoke validators trained on labeled CTI datasets (e.g., differentiating benign curl/PowerShell snippets from malicious jailbreak instructions).
    - **Dual-Layer Validation:**  
      Combine static rule-based filtering (regex/heuristic) with semantic guardrails to ensure high recall without suppressing legitimate intelligence indicators.
    - **Sandboxed LLM Execution:**  
      Future versions of CyberSage will route LLM calls through an **isolated inference container**, ensuring that even if prompt injection occurs, it cannot affect the parent orchestrator or MCP environment.