MASTRIA: Automated Threat Modeling and Risk Analysis for LLM Multi-Agent Systems

Anonymous authors
Paper under double-blind review

Abstract

Assessing the security of multi-agent systems (MASs) is challenging because execution paths, task-delegation relationships, and attack surfaces can evolve dynamically through LLM reasoning and inter-agent interactions, limiting the effectiveness of conventional threat modeling approaches. Existing threat modeling frameworks for agentic systems address some of these challenges but still rely on manually authored architecture descriptions, diagrams, or contextual inputs, making their effectiveness sensitive to the completeness and accuracy of the provided system representation. To address this gap, we present MASTRIA, a three-phase automated threat modeling approach for MASs. Given the source code and an inventory of protected assets, MASTRIA first constructs a system model by identifying entry and exit points and generating a data-flow diagram of agents, tools, and data stores, enriched with inter-agent communication, and deriving trust boundaries and segmentation. Next, it identifies threats through complementary deterministic and agentic analyses guided by the OWASP Agentic Security Initiative (ASI) and a MAS-specific threat taxonomy. The identified threats are linked to relevant assets, attack paths, and adversarial techniques, semantically deduplicated, and validated using both deterministic rules and LLM-based modules. Finally, MASTRIA assigns likelihood- and impact-based risk scores and generates mitigation recommendations. The effectiveness of MASTRIA was evaluated using six open-source multi-agent applications, achieving 97.5% recall and 94.8% precision. The consistency of the results across repeated runs further demonstrates the feasibility of reproducibility, implementation-grounded threat modeling for MASs.

1 Introduction

LLMs have evolved into agentic applications that reason, generate content, and interact with tools (Bajoria et al., 2026). These applications increasingly take the form of *multi-agent systems* (MASs), where specialized agents coordinate to perform complex real-world tasks (e.g., operating browsers, writing and reviewing code, and conducting multi-step research) (Tran et al., 2025). These capabilities also introduce new attack surfaces. A key source of this extended attack surface is that, in agentic systems, natural-language content can serve as both data and instructions. Consequently, allowing poisoned inputs (e.g., retrieved documents or tool outputs) or compromised agents to influence downstream decisions and propagate malicious behavior across agents. This can lead to unauthorized actions, privilege escalation, or data exfiltration without directly compromising the underlying infrastructure (Zhang et al., 2025; Miao et al., 2026).

Threat modeling is an essential security practice for systematically identifying, assessing, and prioritizing security risks before deployment. Conventional approaches typically examine system architecture, assets, data flows, and trust boundaries to identify potential threats, rank them, and guide mitigation decisions (Drake). However, widely used traditional methodologies such as STRIDE (Kohnfelder & Garg, 1999) and PASTA (UcedaVelez & Morana, 2015) primarily assume static system architectures and representations in which components, interactions, and execution paths are specified in advance. This assumption does not hold for MASs, where execution is non-deterministic and key security properties,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

including control flow, trust relationships, and attack surfaces, can emerge dynamically from LLM-driven reasoning and inter-agent interactions.

While recent threat-modeling efforts for agentic systems address their dynamic nature, they still rely heavily on manual expert judgment for system decomposition, threat identification, and risk assessment (Huang, 2025). Tool-supported approaches automate parts of this process, but typically still rely on manually specified architectures or unconstrained LLM reasoning (e.g., ASTRIDE (Bandara et al., 2026)). As a result, their outputs can be difficult to reproduce, validate, audit, or compare.

To address this gap, we propose MASTRIA, an automated three-phase threat-modeling approach for MAS that captures both threats arising within individual agents and those emerging from inter-agent communication, delegation, and coordination. Given the source code of a MAS and an inventory of protected assets (detailed in Appendix A.7), MASTRIA first constructs an enriched system model by identifying entry and exit points, modeling the system components and their communication paths in a MAS-aware data-flow diagram (DFD), and deriving trust boundaries and trust segmentation. In the second phase, MASTRIA applies three complementary threat-identification mechanisms: (1) a deterministic rule-based module based on the OWASP Agentic Security Initiative (ASI) (OWASP Gen AI Security Project, 2025), (2) an agentic analysis to validates candidate threats and identifies ASI threats missed by the deterministic module, and (3) an agentic MAS-taxonomy module that identifies MAS-specific threats, including multi-step threats arising from interactions, delegation, and coordination among agents. The resulting threat candidates are linked to relevant assets and attack paths, deduplicated by identifying semantically overlapping threats, and subsequently validated through a combination of deterministic and agentic procedures. Finally, MASTRIA assesses and ranks the validated threats based on likelihood and impact and generates corresponding mitigation recommendations. The resulting structured threat report provides a systematic and reproducible representation of the MAS attack surface, enabling consistent security analysis across systems and over time.

We evaluate MASTRIA on six applications spanning different architectures, scales (number of agents and tools), and domains. We compare MASTRIA 's results against ground-truth threats identified by human security experts, achieving 97.5% recall and 94.8% precision.

The main contributions of this work are summarized as follows: (1) **Framework-agnostic system modeling of MASs:** We introduce a code-driven method that automatically reconstructs the security-relevant structure of a MAS into a representation that is independent of the underlying implementation framework. (2) **Taxonomy-guided threat identification for MASs:** We develop a threat-identification approach that combines deterministic and agentic analysis based on the OWASP ASI taxonomy and its extended MAS guide. (3) **Reproducible and auditable threat models:** Every identified threat is grounded to the system model and linked to the affected assets and attack paths. The result is a fixed, machine-checkable schema that enables systematic validation, auditing, and comparison across systems and repeated analyses. (4) **Open implementation and benchmark dataset:** We release an implementation of MASTRIA together with a manually curated threat dataset for six open-source agentic applications, supporting reproducibility and future evaluation of MAS threat-modeling approaches.

2 Related Work

Traditional threat-modeling approaches, including the OWASP threat modeling process (Conklin), STRIDE (Kohnfelder & Garg, 1999), PASTA (UcedaVelez & Morana, 2015), and LINDDUN (Wuyts et al., 2014), assume systems whose behavior is defined by explicit software logic and configuration, with external inputs treated as data. They therefore rely on architectural representations such as DFDs to make components, assets, data flows, and trust boundaries explicit, enabling identification of attack surfaces, attack paths, and affected system elements. Recent LLM-based approaches automate parts of this process by analyzing system descriptions, architecture diagrams, or DFDs and mapping them to predefined threat taxonomies (Mollaeefar et al., 2024; Bandara et al., 2025; Crossman et al., 2025).

However, when agentic systems are the subject of analysis, traditional threat modeling processes become insufficient. Such systems may act autonomously, make non-deterministic decisions, and face adversarial manipulation that traditional methods do not explicitly model (Huang, 2025). Recent works have therefore adapted threat modeling methods to AI-based systems. Among these adaptations, STRIDE-AI redefines each STRIDE category for AI-specific failure modes through a structured, analyst-driven, multi-phase assessment process (Cyrille & Schwarz, 2026). ATFAA instead proposes a new threat model for autonomous AI agents, identifying nine threats associated with autonomy, persistent context, multi-step reasoning, and external tool use capabilities of an agent, and complements this taxonomy with the SHIELD mitigation framework (Narajala & Narayan, 2025). ASTRIDE extends Deep-STRIDE Bandara et al. (2025) by adding a new category for AI agent-specific attacks (Bandara et al., 2026). MAESTRO extends traditional threat-modeling approaches with AI-specific considerations, organizing agentic systems into seven architectural layers and addressing threats such as unpredictable agent behavior, memory poisoning, and inter-agent collusion (Huang, 2025).

The approaches discussed above can automate threat identification and analysis, but their automation typically begins from an already available system representation, such as an architecture description. Hence, they do not address the preceding step of deriving the threat-modeling representation from the implemented system itself. Our work instead derives the threat model directly from the source code and an asset inventory, reducing human effort and grounding both the threat analysis and the resulting mitigation recommendations in the implemented system rather than an external representation of it. More broadly, MASTRIA extends the OWASP threat-modeling process with concepts designed specifically for MAS, while preserving its overall workflow.

3 Methodology

3.1 Adversary Model

We assume an external adversary with no initial access to host or deployment environment. The adversary knows public interfaces, documentation, and implementation details but cannot directly access or modify source code, configuration, or protected data. Acting anonymously or through a legitimate user account, the adversary may use any interface available at that privilege level to submit controlled inputs, induce retrieval of controlled content, exploit reachable implementation vulnerabilities, or manipulate responses from external dependencies. These capabilities enable direct or indirect prompt injection, influence LLM model mediated decisions, and induce components to misuse application granted authority. The adversary's scope includes components, assets, and actions reachable through feasible data or control-flow paths originating at attacker-controlled entry points. The adversary cannot pre-modify tools or dependencies but may exploit their vulnerabilities or induce the use of malicious components through such paths. We exclude attacks requiring administrative or insider access, modification of source code, configuration, or the local environment, and interfaces available only with those privileges.

3.2 MASTRIA Overview

As shown in Figure 1, MASTRIA operates through three automated phases: system modeling, threat enumeration, and risk assessment and mitigation. Across these phases, deterministic modules perform structural extraction, model traversal, rule application, and deduplication, while LLM-based modules handle tasks that require semantic interpretation, such as model enrichment, contextual threat identification and validation, and risk scoring. All LLM-generated outputs are constrained to be grounded with evidence from the source code or the derived system model; outputs that cannot be supported by such evidence are rejected. Each phase produces a schema-checked artifact that becomes the input to the next phase, resulting with a traceable outputs. Appendix D documents the purpose, inputs, outputs, and validation requirements of each LLM-assisted step.

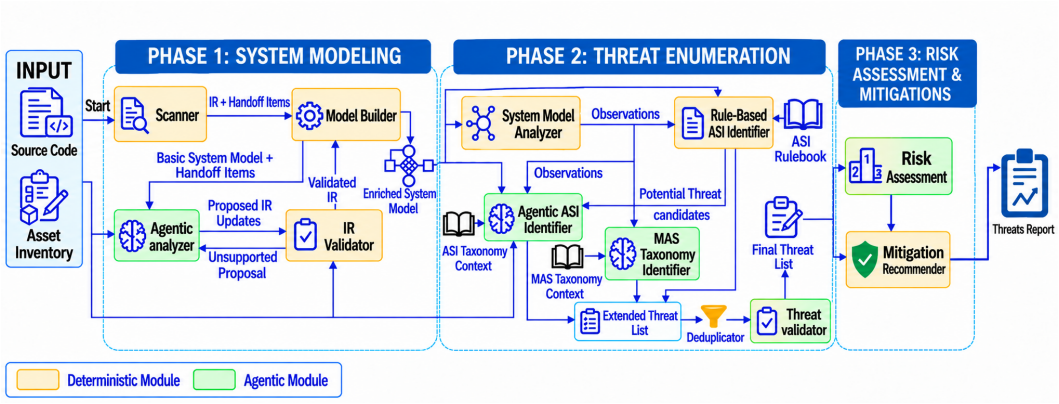

Figure 1: MASTRIA transforms MAS source code and a validated asset inventory into an auditable threat-model report containing an enriched system model, validated threats, risk rankings, and mitigation recommendations. IR denotes intermediate representation.

3.3 Phase 1: System Modeling

The first phase of MASTRIA receives the MAS application source code and asset inventory as input to construct a system model that the following phases use for threat identification and risk analysis. It produces four primary artifacts: (1) a list of entry points, (2) a list of exit points, (3) an enriched DFD, and (4) an asset inventory anchored to the system model. System modeling is done by the following four modules (see Figure 1).

Scanner. The Scanner is a deterministic module that parses the source code into abstract syntax trees (ASTs), traverses them to identify framework-specific elements, and translates them into a framework-agnostic intermediate representation (IR). The IR captures software components, connections, and entry and exit points (see details in Appendix B.1). In some cases, the scanner cannot fully resolve and provide grounded (verifiable) information about an analyzed element. This may be caused by semantic ambiguity, visibility gap, or an unverifiable claim. For these cases, the scanner maintains a list of *handoff items*, which are bounded questions (each refers to one unresolved IR element), where each question contains the relevant source code lines and a closed set of possible outcomes (see details in Appendix B.2).

Model builder. The model builder is also a deterministic module that derives a system model from the IR by applying a fixed set of mapping rules (Appendix B). Specifically, using the components and connections represented in the IR, it constructs the application's DFD, assigns each DFD node an element type (e.g., agent, tool, data store), identifies containment relationships among elements (e.g., a tool shared by multiple agents will be contained by each of these agents), and assigns each DFD edge a flow type based on its source and destination nodes (Appendices A.2 and A.3 provide detailed information on element types and flow types, respectively). It also generates the corresponding entry and exit point lists, and maps each point to its associated DFD node. For each entry point, we provide the external source and mechanism through which data or control enters the application, and it is marked as *agentic* if the incoming data can reach an LLM context. For each exit point, we record the action performed and classify its effect, which we refer to as *action authority*, as *advisory* when it only produces output (e.g., a printed answer to the user), *reversible* when its effect can be undone (e.g., creating a draft of an email), and *irreversible-external* when it cannot (e.g., executing a payment procedure). The action authority is later used to compute the risk level of a threat that reaches the exit point. Finally, the model builder assigns trust levels, partitions the DFD into trust zones, and records each edge between zones as a boundary crossing together with its corresponding category (see Appendices A.5 and A.6). The model builder output is the *basic system model* and the handoff-item list, which together serve as input to the Agentic analyzer, and it is run again on every update to the IR.

Agentic analyzer. The Agentic analyzer is an LLM-based module that produces a list of proposed updates to the IR that (1) resolves the handoff items list and (2) links each asset in the asset inventory to the system model. It receives the basic system model, the handoff item list, the asset inventory, and the source code of the application as input and uses an LLM-based agent to complete the two tasks. For the handoff items, it answers each according to its type. For a *semantic ambiguity*, it chooses one of the admissible outcomes (e.g., identifying the agent(s) that receives and processes the user inputs by reading the agents' system prompts). For a *visibility gap*, it recovers the hidden elements and returns them in IR vocabulary (e.g., listing the tools of a remote MCP server from the code or the asset inventory). For an *unverified claim*, it confirms or replaces the recorded IR fact by reading the evidence the scanner could not resolve (e.g., tracing whether different variable names refer to the same underlying database or distinct data stores). For the asset inventory, the agent proposes for each asset the DFD nodes or entry and exit points the asset lives at, or an empty list with a fixed reason instead of a guess (e.g., `outside_modeled_surface` for a credential cache that only startup code reads). The agent cannot modify the system model or the IR directly; each proposed IR update is later verified by the Validator before applying the updates.

IR validator. Every proposed update of the Agentic analyzer is checked by the deterministic validator module against the source code, the basic system model, and the relevant handoff item. It checks admissibility, not correctness: its purpose is to keep the agentic output inside a closed, checkable set of actions. A proposal is admitted only if the cited source files and lines exist and quote the code as written, every system model element it references exists, and its outcome is one of the handoff item's admissible outcomes. A proposed asset anchor must additionally point at nodes or entry and exit points that exist in the model and fit the asset's class (e.g., a tool capability anchors to a tool node). An unsupported proposal is returned to the agentic analysis once with the failing check, and if it fails again, it is rejected and logged with its reason. Admitted updates are applied to the IR, which is passed to the model builder to regenerate the DFD, the entry and exit point lists, and the asset anchors, producing the *enriched system model* that the next phase consumes.

3.4 Phase 2: Threat Enumeration

The threat enumeration phase analyzes the enriched system model to identify, consolidate, and validate applicable threats.

System model analyzer. First, this deterministic module performs a single traversal of the enriched system model and derives two classes of structured observations: *attack paths* obtained through tainted-propagation analysis, and *action-authority*, which characterizes the potential impact that can result if an attack path is successfully realized (as described in Section 3.3). These observations are then used by three subsequent threat-identification modules. Taint propagation analysis starts from each agentic entry point and follows feasible propagation paths across trust boundary crossings defined in Table 9. Propagation continues until it reaches an external entity. Each resulting attack-path observation is represented as a DFD subgraph rooted at an entry point and terminating at either an exit point or a node containing an asset recorded in the asset inventory. Thus, each observation captures a potential path through which attacker-controlled information can propagate toward a security-relevant target.

Rule-based ASI identifier. This module consumes the observations generated by the system model analyzer and uses a dedicated set of rules (detailed in Appendix C.1) that map observations to the OWASP ASI taxonomy (OWASP Gen AI Security Project, 2025). Each rule includes a structural pattern over the system model, along with a path condition checked against the observations (e.g., paths that include a specific asset class or end in an injection_surface boundary crossing), and produces either a threat or a potential candidate, depending on the rule. When a rule matches, the module produces one of two outputs. If the matched conditions are sufficient to establish a threat, it emits a confirmed threat. If they are necessary but insufficient, it emits a potential threat candidate for the agentic

ASI identifier to evaluate (Appendix C.1). Each output names the affected asset, the ASI category the rule declares, and its evidence in the system model's identifiers (the attack path, the attack surface, and the matched observation attributes), so that every threat and candidate is traceable to the observations and the rule that produced it and can be later checked by the Threat validator.

Agentic ASI identifier. The agentic ASI identifier receives the observations, the potential threat candidates, the system model and the source code, together with an ASI taxonomy context that encodes the OWASP ASI categories and their criteria. An LLM agent performs *candidate verification* and *coverage completion*. In *candidate verification* it confirms, revises, or rejects each potential candidate by checking its unresolved rule conditions against the system model and the source code(e.g., an agent that can trigger a payment is a confirmed threat only if in the verification the code shows no approval step before it) (Appendix C.1.1). In *coverage completion*, it applies the taxonomy criteria to identify ASI threats not covered by the rulebook(e.g., whether one agent's output can carry instructions into another agent's context unchecked, which needs reading how the receiving agent handles it). Each confirmed candidate and newly identified threat follows the threat schema deteailed in Appendix B.6, each with the system model and source code references that show it is possible and a short explanation of how they do, as evidence for the threat validator.

MAS-taxonomy identifier. The MAS-taxonomy identifier runs a second LLM agent in parallel with the agentic ASI identifier and independently analyzes the system model and observations. Its taxonomy context encodes fourteen questions from the OWASP Multi-Agent System Threat Modeling Guide (OWASP GenAI Security Project, Agentic Security Initiative, 2025) (Appendix C.1.2), covering multi-agent scenarios such as tool misuse through delegation and cascading failures, where one agent's unchecked output misleads the next. It gives a second, independent view: the MAS guide frames threats as questions over the system's coordination structure rather than as categories, so its agent reaches multi-agent instances that the category-driven rules and candidate verification did not raise, and its findings map back onto the ASI categories. Each question specifies the conditions for emitting a threat. The agent answers each question and emits the threats the system model and observations support, in the same threat schema. Overlaps with other modules are resolved by the Deduplicator. Additional taxonomies can be supported by providing a taxonomy context following the schema (Appendix B.6) and deploying a corresponding agent, one agent per taxonomy context.

Extended threat list. The union of the three modules' outputs forms the extended threat list: rule-confirmed threats, verified candidates, and newly identified threats from the two agents. Each of them is a specific adverse event on an anchored asset (Appendix A.1), not a category:"prompt injection" alone is not a threat in MASTRIA, whereas "prompt injection leading to customer-data exfiltration through the drafter agent's email tool" is. Each threat identifies the affected asset, attack vector, attack path, entry and exit points, action authority, taxonomy labels, supporting evidence, and module that produced it (Appendix B.6 defines the complete schema). The extended list is passed through the Deduplicator and the Threat validator, which produce the final threat list.

Deduplicator. The Deduplicator is a deterministic module that reduces the extended list to one threat per identity in two stages, calibrated on human-triaged duplicates from a development application (Appendix C.2). First, threats are considered identical when they share the affected asset, ASI label, entry point, the edge over which the threat is realized, and exit point. Their attack paths and vectors are merged as supporting evidence, along with the modules that produced it. Second, because ASI categories are defined at different levels, some naming a mechanism (e.g., tool misuse) and some an outcome (e.g., intent breaking), one exposure can receive different labels from different modules. Threats that differ only in their ASI labels are therefore merged when those labels describe the same outcome class (e.g., a threat labeled Tool Misuse (T2) by the rulebook and Intent Breaking (T6) by the MAS-taxonomy identifier, both describing an agent driven to misuse its tool). This combines the ASI and MAS-taxonomy views, and the merged threat keeps both labels.

Threat validator. The Threat validator determines which generated threats are admitted to the final threat list, combining deterministic structural checks with an LLM agent's correctness judgment. The structural checks verify that (i) the referenced asset, DFD components, entry point, and exit point exist in the system model, (ii) each step in the attack path corresponds to a valid DFD edge, (iii) the action authority attributed to the threat matches that of the referenced exit point, (iv) the quoted code that matches the source, and (v) the threat conforms to the schema defined in Appendix B.6. Each threat is reviewed by the agent against the system model and source code. The agent either rejects it as incorrect or revises missing/incorrect attributes and reruns the structural checks, with one retry allowed if it fails. Threats that fail twice or are judged incorrect are rejected and logged with the rejection reason.

3.5 Risk Assessment and Mitigation

Risk assessment. After threat identification, MASTRIA assesses each threat along two dimensions: *likelihood* and *impact*. Following the risk-assessment framework of NIST SP 800-30 (Initiative, 2012), these dimensions are combined to derive an overall risk rating of low, medium, or high. *Likelihood* is determined from properties of the identified attack path and the conditions required for successful exploitation. In particular, the assessment considers whether attacker-controlled input can reach the affected component and whether any additional preconditions must be satisfied. *Impact* is derived from the sensitivity of the affected asset and the action authority associated with the affected component or tool, including the scope and reversibility of the resulting action and whether its effects can propagate to other components. These signals are obtained directly from the threat record and the system model. MASTRIA supports both deterministic and LLM-based risk assessment. The deterministic procedure derives likelihood and impact ratings from structured evidence already contained in the threat record and system model using a fixed set of scoring rules. Applying the same rules to every threat yields a reproducible assessment in which each rating can be traced to explicit properties of the modeled system. For the LLM-based assessment, one model call is performed for each threat. The model receives the threat record, the relevant system-model context, and instructions to assign likelihood and impact ratings together with supporting justifications. In the hybrid variant, the prompt additionally includes the deterministic ratings, allowing the model to consider the rule-based assessment while producing its final ratings.

Mitigation recommender. The mitigation strategies for the identified threats are recommended by a deterministic module from the OWASP Agentic AI security guidance. Each threat identified in the threat model was mapped to the corresponding OWASP ASI risk category. Based on this mapping, the relevant mitigation recommendations proposed by OWASP were selected and adapted to the system under analysis.

4 Evaluation

Evaluated applications. We evaluate MASTRIA on six open-source multi-agent applications: *Email Assistant* (Gandhi et al., 2026), *HalluCheck* (Solomon et al., 2026), *Finance* (Levi et al., 2026), *Trip Planner*,*Instagram* (CrewAI, Inc., 2026) and *RealEstate* (Hancock, 2024). The applications were selected to cover diverse architectures (e.g., sequential and hierarchical), entry points (e.g., chat input, documents, and web content), tools (e.g., email, RAG, code execution, and MCP tools), and potential impacts (e.g., advisory output, file modification, and external actions). Appendix F provides detailed descriptions of each application.

Ground-truth construction. To construct the ground truth for evaluating threat coverage, three reviewers with expertise in cybersecurity, AI, and agentic systems independently conducted a manual threat modeling of each of the six applications. They followed the established OWASP Threat Modeling Process and used the OWASP *Agentic AI Threats and Mitigations* guide as an agent-specific threat reference (OWASP Gen AI Security Project, 2025). This provided a common threat modeling scope and terminology while each reviewer independently enumerated the threats applicable to each application. Because the reviewers

described threats in free-form language, we manually aligned semantically equivalent threats across their analyses before measuring inter-reviewer agreement using mean pairwise F1. Across the three pairwise reviewer comparisons, the mean F1 score was 0.911, indicating substantial overlap among the independently constructed threat sets. Disagreements (28 in total) were resolved through discussions, and the resulting reconciled threat sets served as the ground truth for the recall evaluation.

Human evaluation of generated threat records. We evaluate the threats generated by MASTRIA along two complementary dimensions: coverage and correctness. Recall measures whether MASTRIA recovers the threats identified in the manually constructed ground truth, but it does not assess the correctness of additional threats identified by the pipeline. We therefore evaluate threat-level precision separately through human review. Each reviewed threat record is assigned one of three labels: (i) **Correct**, when the identified threat description is valid, supported by the system model, and the threat record (metadata) is complete, including a correct ASI mapping; (ii) **Correct (Incomplete)**, when the identified threat description is valid but its threat metadata contains an incomplete/partially incorrect ASI mapping; and (iii) **False Positive**, when the identified threat is incorrect or unsupported by the modeled system. Accordingly, for computing threat-identification precision, both **Correct** and **Correct (Incomplete)** records are counted as correctly identified threats.

LLM-assisted evaluation. Human review is our primary reference for assessing threats generated by MASTRIA. However, manual review can become costly as threat records and evaluated applications increase. We therefore ask whether an automated judge can serve as a scalable validation mechanism that researchers applying MASTRIA to their own systems can use in place of exhaustive manual review. Treating the human-annotated records as ground truth, we measure how closely the panel reproduces human labels; that agreement determines whether, and under what conditions, the panel can be trusted as a substitute for manual validation at scale. We use majority voting across three leading LLMs: Kimi K3, GPT-5.6, and Opus 5, selected to limit correlated judging behaviour within a single model family. Each model independently receives the same system information, generated threat record, and evaluation guidelines used by the human reviewers, and assigns one of the three evaluation labels (prompts are provided in Appendix E). The LLMs neither modify the generated threat records nor contribute to constructing the human ground-truth threat set.

Results. All results reported in this section were obtained using GPT-5.5 as the underlying LLM for MASTRIA. Across the six MAS applications, MASTRIA generated 325 threat records, which were evaluated by human reviewers. We compare these records against the reconciled human ground-truth threat set (which contained 316 threats).

MASTRIA performance: Table 1 summarizes both threat coverage and record-level correctness. Of the 316 ground-truth threats, MASTRIA recovered 308 (266 were fully correct and 42 were correct but incomplete) and missed 8, yielding an overall recall of 0.975. The remaining 17 records were false positives, yielding an overall precision of 0.948, indicating consistently high coverage and record-level correctness across different MAS topologies and application domains. The analysis required 1,232 LLM calls at a total cost of $43.91 (a detailed cost breakdown is provided in Table 24).

Ablation: We conduct two complementary ablation studies to assess the contributions of pipeline individual components and the the choice of underlying LLM. Removing MAS/ASI-generated threats leaves precision unchanged but reduces recall to 0.478; disabling deduplication yields 0.775 precision and 0.961 recall; and removing ASI corrections results in 0.848 precision and 0.885 recall (detailed results are presented in Table 28). We further evaluate MASTRIA using the locally deployable Kimi-2.7 model while keeping the rest of the pipeline unchanged. Since Kimi-2.7 failed on *realestate* due to a token-related error, we compare performance on the remaining five applications, where GPT-5.5 achieves 0.944 precision and 0.978 recall, compared with 0.831 and 0.693 for Kimi-2.7 (see results in Table 27).

LLM panel: Table 2 reports the label distribution and precision assigned by the LLM panel described above across all 325 generated threat records. On the independently human-reviewed records, the panel agrees on 318 human Correct/Correct (I)/FP label. In particular,

Table 1: Threat coverage against reconciled human ground truth and human evaluation of generated threat records. $|T_{\mathrm{GT}}|$ denotes the number of ground-truth threats; *Recovered* and *Missed* denote the number of ground-truth threats recovered and not recovered by MASTRIA, respectively. Correct, Correct–Incomplete (Correct (I)), and False Positive (FP) denote the human-assigned labels for generated threat records.

Application	$\lvert T_{\mathrm{GT}}\rvert$	Recovered	Missed	Correct	Correct (I)	FP	Recall	Precision
Email Assistant	62	62	0	58	4	5	1.000	0.925
HalluCheck	36	35	1	33	2	0	0.972	1.000
Trip Planner	39	38	1	30	8	3	0.974	0.927
RealEstate	39	37	2	30	7	1	0.949	0.974
Finance	85	83	2	73	10	4	0.976	0.954
Instagram	55	53	2	42	11	4	0.964	0.930
Overall	316	308	8	266	42	17	0.975	0.948

all 266 records classified as Correct and 42 Correct (I) by the human reviewers are also classified as Correct, Correct (I) by the panel respectively. The remaining disagreements are concentrated among the small number of human-identified false positives: of seventeen human FPs, the panel also labels ten as FP and other seven as Correct (I). Thus, the panel exhibits a somewhat more permissive judging tendency than the human reviewers. These results support using the LLM panel as an auxiliary mechanism for scaling evaluation, rather than as a replacement for human judgment. Human review remains the reference for false-positive calibration, while the high overall and binary agreement indicates that majority-vote LLM judgments can provide a useful large-scale estimate of generated-record quality.

Table 2: Majority-vote LLM-panel evaluation of generated threat records. Correct, Correct–Incomplete (Correct (I)), and FP report the panel's majority-vote verdicts.

Application	Total	Correct	Correct (I)	FP	Panel Precision
Email Assistant	67	58	6	3	0.955
HalluCheck	35	33	2	0	1.000
Trip Planner	41	30	10	1	0.976
RealEstate	38	30	8	0	1.000
Finance	87	73	12	2	0.977
Instagram	57	42	11	4	0.930

5 Conclusion and Limitations

We presented MASTRIA, an automated threat-modeling approach for MASs that derives a structured, MAS-specific system model from source code and a validated inventory of protected assets. Evaluation across six open-source applications shows that implementation-grounded threat modeling is feasible across heterogeneous MAS architectures and can achieve high threat coverage and precision. By combining a source-derived system model with evidence-grounded agentic analysis, MASTRIA provides a systematic, reproducible, and auditable approach to threat modeling for systems whose behavior is partly determined by LLM reasoning.

Nevertheless, MASTRIA assumes access to application source code and relies on framework-specific scanners to translate implementation constructs into its normalized representation. Runtime-generated components, external configuration, remote services, and behaviors not represented in the available source code may therefore be missed. The handoff mechanism allows agentic analysis to address visibility gaps, but it cannot recover behavior for which no relevant evidence is available. Moreover, the identified attack paths represent feasible routes for attacker influence rather than observed executions, since MAS behavior may also depend on model outputs, external content, persistent state, and interaction history. Future work can complement the static analysis with runtime traces, dynamic instrumentation, and adversarial testing to better assess exploitability and distinguish structurally possible threats from those observed in practice.

AI USE STATEMENT

(LLMs were used as components of proposed MASTRIA framework and also served as research tools in its evaluation. Within the framework, LLMs perform semantic enrichment of the system model and asset anchoring, taxonomy guided threat identification and candidate verification, threat validation, and risk assessment. To support reliability and reproducibility, every LLM output is constrained by a predefined schema and grounded in the application source code or the derived system model, with deterministic validation applied at the stages specified by the framework.

Our primary evaluation uses GPT-5.5 as MASTRIA's underlying model, and we additionally evaluate Kimi-2.7 as an alternative backend to examine portability across models. To complement human review, which serves as the primary evaluation reference, an independent panel of three LLM judges (Kimi K3, GPT-5.6, and Opus 5) assessed the generated threat records. The panel acted only as evaluators and human ground-truth threat sets were constructed independently of it, all records were assessed exactly as generated.

Generative AI tools also supported language editing to improve the clarity and readability of the manuscript. The authors reviewed and validated all AI-assisted text and research outputs following the procedures described in the paper, and take full responsibility for the manuscript's content, claims, results, and released artifacts.

6 ETHICS STATEMENT

Data and responsible disclosure. Our valuation uses six open-source applications, analyzed in accordance with their licenses. No personal or proprietary data were processed, and only publicly available source code was submitted to commercial LLM APIs.

Human involvement. The ground-truth threat sets were constructed by three security practitioners, each with three to five years of experience in cybersecurity and two years of hands-on experience with AI and agentic systems. Their role was expert annotation of the evaluated applications, no data about the annotators themselves were collected, so the work does not constitute human-subjects research and IRB review was not required.

References

Sparsh Bajoria, Shreyanshu Ranjan, Adhitya M, Vineet Suri, Vikas Hassija, Vinay Chamola, and Amir Hussain. From language models to agentic ai: A survey of autonomous, action-enabled, and collaborative llm agents. *Cognitive Computation*, 18(1):103, 2026.

Eranga Bandara, Amin Hass, Sachin Shetty, Ravi Mukkamala, Ross Gore, Abdul Rahman, and Safdar H Bouk. Deep-stride: Automated security threat modeling with vision-language models. In *2025 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–7. IEEE, 2025.

Eranga Bandara, Amin Hass, Sachin Shetty, Ravi Mukkamala, Ross Gore, Sachini Rajapakse, Xueping Liang, and Safdar H Bouk. Astride: A security threat modeling platform for agentic-ai applications. In *2026 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 386–393. IEEE, 2026.

Larry Conklin. Threat Modeling Process (Historical) | OWASP Foundation — owasp.org. `https://owasp.org/www-community/{T}hreat_{M}odeling_{P}rocess`. [Accessed 13-07-2026].

Larry Conklin, Victoria Drake, Sven Strittmatter, Zoe Braiterman, and Adam Shostack. Threat modeling process. OWASP Foundation, Community Pages, 2024. `https://owasp.org/www-community/Threat_Modeling_Process`, accessed 2026-06-28.

CrewAI, Inc. CrewAI: Framework for multi-agent automation. `https://github.com/crewAIInc/crewAI`, 2026. Accessed: 2026-09-19.

Andrew Crossman, Andrew R Plummer, Chandra Sekharudu, Deepak Warrier, and Mohammad Yekrangian. Auspex: Building threat modeling tradecraft into an artificial intelligence-based copilot. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, pp. 1160–1167. IEEE, 2025.

Tsafac Nkombong Regine Cyrille and Franziska Schwarz. Stride-ai: A threat modeling framework for generative ai security assessment. *arXiv preprint arXiv:2605.17163*, 2026.

Victoria Drake. Threat modeling. URL `https://community.owasp.org/Threat_Modeling`.

Parth Atulbhai Gandhi, David Tayouri, Akansha Shukla, Beni Ifland, Yuval Elovici, Rami Puzis, and Asaf Shabtai. Atag: Ai-agent application threat assessment with attack graphs. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, ASIA CCS '26, pp. 805–819, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723568. doi: 10.1145/3779208.3785380. URL `https://doi.org/10.1145/3779208.3785380`.

Hardik Goel. Security risks in tool-enabled ai agents: A systematic analysis of privileged execution environments. In *2026 IEEE 50th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 3140–3145, 2026. doi: 10.1109/COMPSAC69091.2026.00466.

Brandon Hancock. AI Real Estate Offers. `https://github.com/bhancockio/ai-real-estate-offers`, 2024. Accessed: 2026-09-19.

Ken Huang. Agentic AI threat modeling framework: MAESTRO. Cloud Security Alliance Blog, February 2025. `https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro`, accessed 2026-06-28.

Joint Task Force Transformation Initiative. Guide for conducting risk assessments, Sep 2012. URL `https://csrc.nist.gov/pubs/sp/800/30/r1/final`.

Loren Kohnfelder and Praerit Garg. The threats to our products. *Microsoft Interface, Microsoft Corporation*, 33:1–8, 1999.

Yarin Yerushalmi Levi, Roy Betser, Amit Giloni, Lidor Erez, Itay Gershon, Oren Rachmil, Sindhu Padakandla, and Roman Vainshtein. Rift-bench: Dynamic red-teaming for agentic ai systems. *CoRR*, abs/2606.23927, June 2026. URL `https://doi.org/10.48550/arXiv.2606.23927`.

Rui Miao, Yixin Liu, Yili Wang, Xu Shen, Yue Tan, Yiwei Dai, Shirui Pan, and Xin Wang. BlindGuard: Safeguarding LLM-based multi-agent systems under unknown attacks. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39215–39234, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1819. URL https://aclanthology.org/2026.acl-long.1819/.

Majid Mollaeefar, Andrea Bissoli, and Silvio Ranise. Pillar: an ai-powered privacy threat modeling tool. *arXiv preprint arXiv:2410.08755*, 2024.

Vineeth Sai Narajala and Om Narayan. Securing agentic ai: A comprehensive threat model and mitigation framework for generative ai agents. In *2025 8th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI)*, pp. 1–6, 2025. doi: 10.1109/ACAI68217.2025.11406310.

OWASP Gen AI Security Project. Agentic AI – threats and mitigations, version 1.0. OWASP Agentic Security Initiative (ASI) Whitepaper, February 2025. URL https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/. Accessed: 2026-09-19.

OWASP GenAI Security Project, Agentic Security Initiative. Multi-agentic system threat modelling guide. Guide Version 1.0, OWASP Foundation, April 2025. https://genai.owasp.org/, accessed 2026-06-28.

Ron Solomon, Yarin Yerushalmi Levi, Lior Waknin, Eran Aizikovich, Amit Baras, Etai Ohana, Amit Giloni, Shamik Bose, Chiara Picardi, Yuval Elovici, and Asaf Shabtai. Lumimas: A comprehensive framework for real-time monitoring and enhanced observability in multi-agent systems. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '26, pp. 3749–3757, Richland, SC, 2026. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9798400723179. doi: 10.65109/PNWL9707. URL https://doi.org/10.65109/PNWL9707.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of llms, 2025. URL https://arxiv.org/abs/2501.06322.

Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: process for attack simulation and threat analysis*. John Wiley & Sons, 2015.

Kim Wuyts, Riccardo Scandariato, and Wouter Joosen. Empirical evaluation of a privacy-focused threat modeling methodology. *Journal of Systems and Software*, 96:122–138, 2014.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=V4y0CpX4hK.

Supplementary Material

A Modeling Vocabulary

This section defines the structured vocabulary used to construct a system model and record threats. Architecture primitives are specified in A.2–A.4, trust and boundary primitives in A.5–A.6, and assets and threat paths in A.7–A.8. Table 3 is an index; the rules and fields appear in the subsequent subsections.

A.1 Modeling primitives

Table 3: Modeling primitives used by MASTRIA to represent MAS architecture, trust, and threats.

Group	Role	Primitives
Architecture	Represents system components, interaction points, and data movement.	DFD; element type; flow type; entry point; exit point
Trust	Represents trust assignments and changes across system interactions.	Trust level; trust zone; trust boundary; boundary crossing
Threat analysis	Connects protected assets to their exposure and possible attack progression.	Asset; attack surface; attack vector; attack path

Architecture primitives. MASTRIA retains the standard DFD elements (external entities, processes, data stores and data-flow edges) and extends them with MAS-specific element types and flow types (Appendices A.2 and A.3). A containment hierarchy records ownership and dependency: an agent may contain an LLM engine and a set of tools, so a threat affecting an internal component can be traced to the agent and the capabilities that depend on it. Entry and exit points (Appendix A.4) delimit how influence enters the system and how its actions take effect. To connect these data paths to agent behavior, each routing or action-selection decision is annotated with a *control mechanism*, `deterministic` or `llm_driven`. The control mechanism and the action-authority tier of an exit point are complementary: the former identifies how an action is selected and whether attacker-controlled content can influence that selection, whereas the latter characterizes the potential effect of the selected action. Together, these primitives let MASTRIA trace a security-relevant path from an input channel, through the component and decision it may influence, to the exit point through which the resulting action takes effect.

Trust primitives. Conventional DFD-based threat modeling represents trust changes through boundaries separating authentication, privilege, or administrative domains (Conklin et al., 2024). MASs complicate this abstraction: an agent may hold legitimate system privileges yet remain steerable by content entering its model context (Goel, 2026). Privilege alone therefore does not capture exposure to attacker influence. MASTRIA assigns each node a trust level: *untrusted* for externally controlled components and *semi-trusted* for system components influenced by processed content. A third level, *trusted*, is reserved for deterministic code and libraries, which are not modeled as nodes (Table 8). Trust levels and the containment hierarchy partition the architecture into trust zones. Agents at the same trust level are placed in separate zones, because attacker influence can propagate laterally without a privilege change. An edge between different levels crosses a *level boundary*; an edge between same-level zones crosses a *lateral boundary*. Each crossing is recorded with its direction and a category (Appendix A.6), exposing both conventional trust transitions and inter-agent propagation paths for threat identification.

Threat primitives. Adapting the risk definition of NIST SP 800-30 (Initiative, 2012), MASTRIA defines a *threat* as a potential adverse event affecting an anchored asset. The *attack surface* comprises the boundary crossings through which an attacker can engage the system (Appendix A.6). An *attack vector* combines the attack type, the delivery mechanism,

and the targeted entry point. An *attack path* records the ordered DFD nodes and edges through which attacker influence propagates toward the asset, or toward an exit that affects a downstream asset. Assets are defined in Appendix A.7; the fields that record vectors and paths in a threat are given in Appendix B.6.

A.2 Element types and subtypes

An element type classifies a DFD node by a standard DFD category and a MAS-specific subtype. Table 4 lists the closed vocabulary.

Table 4: DFD element categories and subtypes. Deterministic orchestration code (routers, drivers) is not given a node: the DFD models the agentic attack surface only.

Category	Subtype	What it represents
`external_entity`	`user`	The human principal; inferred, only when an agent sends to or receives from it.
	`llm_provider`	The hosted model API behind an LLM engine; inferred, one per provider identity.
	`external_service`	An API, host, or remote MCP server outside the application; one shared node per endpoint host, flagged `mcp_endpoint` when it is a remote MCP server.
`process`	`agent`	A component that receives input, reasons through an LLM, and produces output.
	`llm_engine`	The model client an agent calls; child of its agent. Represents the agent's capability to call an LLM.
	`tool`	A function an agent can invoke, including each statically named MCP tool; child of its agent.
	`mcp_server`	A local MCP server that executes tool code on the host; standalone in its own zone.
	`mcp_client`	The in-system MCP client for a remote MCP server whose tools are discovered at run time; child of its agent.
`data_store`	`data_store`	Persistence between steps or runs: files, databases, vector indexes, shared memory.

Conventions. (1) *Containment.* An agent owns its LLM engine, its tools and its remote MCP clients (parent and children). A local `mcp_server` is not owned. A component bound by several agents is duplicated into one copy per owner, `<id>@<agent>`, carrying `original_id`, so that a tool invocation stays inside the owner's zone and compromising one agent's copy does not hand over another's. (2) *Inferred nodes.* The `user` node exists only when some agent sends to or receives from it. Two engines on the same vendor share one `llm_provider` node; engines of different vendors do not. (3) *Capability flags.* Every node carries `code_execution`, `read_external`, `write_external`, `read_internal` and `write_internal`. A tool carries the flags of its own code. A tool served by an MCP server carries the server's union, because a compromised server can answer any tool call with any other tool it exposes. An agent carries the union of its children. LLM engines and inferred entities carry none. (4) *Stores.* A data store earns a node only in an attacker-relevant flow, that is, when it is written and read back, or read by something that reaches an exit point. Credential caches and write-only outputs therefore enter the asset inventory (Appendix A.7) but not the DFD.

Examples. Email Assistant: agents `fetcher` and `drafter`, tools `GmailSearchTool@fetcher` and `GmailDraftCreator@drafter`, and one `external_service gmail_api`. Finance: a connection to a remote MCP server yields the `mcp_client financial_mcp@market_worker` in the agent's zone, plus an `external_service` host flagged `mcp_endpoint`; the portfolio file written by `buy_stock` and read back by the same agent is a `data_store` node `portfolio_state`.

A.3 Flow types and endpoint typing

The builder assigns an edge type from its endpoint subtypes using Table 5 in order; the first match wins. The fallback `data_flow` did not occur in the evaluated applications.

A recorded call induces its paired return edge. The builder also connects every LLM engine to its provider with an API-call/API-response pair. Agent-to-agent transfers use `inter_agent` whether a fixed task chain or an LLM-selected delegation produced the handoff; the deciding agent is the edge source.

Table 5: Flow-type inference from source and target subtypes.

Source	**Target**	**Flow type**
user	agent	`user_input`
agent	user	`system_output`
agent	llm_engine	`prompt_construction`
llm_engine	agent	`llm_response`
llm_engine	llm_provider	`llm_api_call`
llm_provider	llm_engine	`llm_api_response`
agent or llm_engine	tool	`tool_invocation`
tool	agent or llm_engine	`tool_result`
agent, llm_engine, tool, or mcp_client	mcp_server	`mcp_request`
mcp_server	agent, llm_engine, tool, or mcp_client	`mcp_response`
agent or llm_engine	mcp_client	`mcp_request`
mcp_client	agent or llm_engine	`mcp_response`
tool, mcp_server, or mcp_client	external_service	`external_request`
external_service	tool, mcp_server, or mcp_client	`external_response`
any	data_store	`data_write`
data_store	any	`data_read`
agent	agent	`inter_agent`
any	any	`data_flow`

A.4 Entry Points and Exit Points

An entry point is the channel through which data enters the system from outside it, pre-existing local data included; an exit point is the channel through which the system acts on or releases data to the outside. Both catalogs are produced beside the DFD and anchored to it by node id. Table 6 lists the closed vocabularies of their classifying fields. Two fields classify an entry point: `external_source` gives the origin (who authored the content) and `mechanism` the channel it arrives through.

Conventions. (1) *Anchoring.* Entry is the first node inside; exit is the last node inside. For an in-process tool, both are the tool. For a tool served by a local MCP server, both are the server (the executor). For a remote MCP binding, the last node inside is the `mcp_client` node. The pair (`external_source_node`, `receiving_component_id`) is a real DFD edge. (2) *Agentic.* A chat message or task input is agentic. A CLI flag that only steers control flow is not, nor is an environment variable holding a key. For an exit, agentic is a property of the attribution, not of the type: the same `file_write` is agentic when a tool the LLM invoked performs it and non-agentic in startup code. (3) *Action authority.* This is the authoritative definition of the three tiers used throughout the paper. `advisory` produces output only, including a read-only outbound call whose request arguments leave the system (search, scrape, API GET); `reversible` effects an undoable change (a local file, a draft, a revertible record); `irreversible_external` effects a change the operator cannot undo (an email sent, a payment, a public post, a destructive delete on an external system). The tier of an exit bounds the impact of every threat that reaches it in the risk assessment. Every exit point starts at its per-type default, and the Agentic analyzer refines it (Appendix B.2).

Table 6: Classifying fields of the entry-point and exit-point catalogs. Both catalogs also carry name, source file, source line, description, and provenance fields.

Field	Values
Entry point (`EP-<n>`)	
`mechanism`	`user_chat_interface_input`, `tool_response`, `file_read`, `api_request`, `llm_api_response`, `env_variable`, `cli_argument`, `function_parameter`, `remote_dataset_read`, `other` (the channel it arrives through)
`external_source`	`direct_user_input`, `external_api_input`, `external_data_store`, `local_stored_data`, `external_entity_other` (the origin, never the channel)
`is_agentic`	True when the data reaches an LLM context.
`receiving_component_id`	The first in-system node the data reaches.
`external_source_node`	The external node it comes from (null when the origin is not modeled).
`data_type`	`string`, `file_path`, `json`, `boolean`, `numeric`, `tabular`, `other`
Exit point (`XP-<n>`)	
`exit_type`	`file_write`, `http_outbound`, `email_outbound`, `database_write`, `tool_external_write`, `system_output`, `subprocess`, `other` (the effect)
`mechanism`	`tool_invocation`, `llm_api_call`, `direct_code`, `system_output`, `other` (the channel)
`is_agentic`	True when the exit's component is one an agent invokes (a tool, MCP server, or MCP client).
`action_authority`	`advisory` < `reversible` < `irreversible_external`
`component_id`	The last in-system node.

Example. Trip Planner has 21 entry points, 12 of them agentic. By mechanism: `env_variable` 9, `tool_response` 5, `user_chat_interface_input` 4 and `llm_api_response` 3; by origin: `local_stored_data` 9, `external_api_input` 8 and `direct_user_input` 4. Its exit-point catalog is shown in Table 7.

Table 7: Trip Planner exit-point catalog, grouped by exit type (10 per-agent rows). These are the classifications reported in the supplied appendix; nine of ten rows are agentic. The code-execution classification requires reconciliation with the authority definition.

Exit	Copies	Type	Authority recorded	Rationale
`search_internet`	3	HTTP outbound	advisory	Read-only query to the documented search endpoint.
`ScrapeWebsiteTool`	2	HTTP outbound	advisory	Read-only page fetch.
`llm_provider_request`	3	HTTP outbound	advisory	The request itself is the effect.
`final_response`	1	System output	advisory	Printed answer; the one non-agentic exit.
`calculate`	1	Subprocess	`irreversible_external`	Evaluation of agent-supplied text as code.

A.5 Trust Levels, Zones and Boundaries

A trust level is assigned to every DFD node; a trust zone groups the nodes that share a level under the containment hierarchy, with a separate zone per agent; and a trust boundary separates two zones. Table 8 gives the complete assignment rules used by the Model builder (Section 3.3); the rationale for level and lateral boundaries is given in Appendix A.1. Rules are priority-ordered; the first match wins.

Table 8: Trust-level and zone assignment. Children inherit the level of their parent. The level *trusted* is reserved for deterministic code and libraries assumed to behave as written; these are not modeled as nodes, so the builder emits only *untrusted* and *semi-trusted*.

Priority	Subtype	Trust level	Zone
1	`llm_provider`	untrusted	own zone
2	`external_service`	untrusted	own zone
3	`user`	untrusted	own zone
4	`data_store`	semi-trusted	own zone
5	`agent`	semi-trusted	`zone_<agent>`
6	`llm_engine`	inherits its owner	owner's zone
7	`tool`	inherits its owner	owner's zone
8	`mcp_server` (local executor)	semi-trusted	own zone
9	`mcp_client` (conduit, local or remote)	semi-trusted	owner's zone

A.6 Boundary-Crossing Categories and the Attack Surface

A crossing category classifies the security relevance of a boundary crossing from its flow type and endpoints, and the attack surface is the set of crossings whose category admits attacker engagement. Table 9 is the category rule table; the first match wins.

Table 9: Crossing-category rules in evaluation order.

Source	Target	Category	Meaning
`external_service` with `mcp_endpoint`	`tool`, `mcp_server`, `mcp_client`	`tool_poisoning`	A remote MCP endpoint defines the tool functionality the system lists and calls; the capability mutates before any content flows (poisoned descriptions, rug pulls, shadowing).
`user`, `external_service`, `tool` (on `tool_result`), `mcp_server`	`agent`, `tool`, `llm_engine`, `mcp_client`	`injection_surface`	Untrusted or less-trusted content arriving at run time.
`agent`, `tool`, any caller	`agent`, `tool`, `mcp_server`	`taint_propagation`	Tainted data moving between first-party components.
`agent`, `tool`, `mcp_server`, `mcp_client`	`external_service`	`exfiltration_surface`	Data leaving the system.
`agent`	`user`	`return_flow`	The system's answer to the human.
any	`data_store`	`memory_poisoning`	A write that a later read will trust.
`data_store`	any	`data_retrieval`	The read-back half of a memory-poisoning loop.
`llm_engine`, `agent`, `tool`	`llm_provider`, and back	`llm_boundary`	Provider-bound flows in both directions, kept separate so vendor-specific threats can attach.

Conventions. (1) Only crossing edges are categorized. The category is attached to the edge together with its source zone, target zone, direction and flow type. (2) `tool_poisoning` keys on the source, not the target: a crossing is `tool_poisoning` when its untrusted source is a remote MCP endpoint, whatever in-system node it feeds. A search API answering a first-party tool or local server is run-time content and stays `injection_surface`. (3) *Attack surface.* The attack surface comprises the crossings categorized `injection_surface`, `tool_poisoning`,

`taint_propagation`, `exfiltration_surface`, `memory_poisoning`, `data_retrieval` and `llm_boundary`. `return_flow` is an operational crossing and is excluded. Each category constrains which attack types apply at that crossing; a threat's `attack_surface` field (Table 11) names the category of the crossing it exploits.

Examples. Finance: the edge from the `mcp_endpoint` host to `financial_mcp@market_worker` is `tool_poisoning`, because the remote server chooses which tools exist and how they describe themselves. The same agent's `buy_stock` write to `portfolio_state` is `memory_poisoning`, and the read-back is `data_retrieval`. Trip Planner: a search tool's call to its API is an outbound `exfiltration_surface`, and the response an inbound `injection_surface`. Its 16 crossings split into `injection_surface` 4, `llm_boundary` 6, `exfiltration_surface` 3, `taint_propagation` 2 and `return_flow` 1 .

A.7 Asset Classes and the Asset Inventory

An asset is anything of value to defend, or of value to an attacker to read, steal, influence, abuse, trigger, poison or route through. The asset inventory is the per-application catalog of assets that the system-modeling phase anchors to the system model (Section 3.3). Table 10 gives the four classes and their subtype vocabulary.

Table 10: The four asset classes and their subtypes. The classes are closed; the subtypes are the vocabulary used in the evaluated applications and may be extended. A qualifier in parentheses refines a subtype, e.g., `action-tool (booking)`, and never replaces it.

Class (id prefix)	Definition	Subtypes
Information (`INFO`)	Data at rest, in transit, or in use worth protecting	`credential`, `config`, `control-parameter`, `control-template`, `system-prompt`, `user-data`, `agentic-memory`, `agentic-state`, `learned-memory`, `retrieved-docs`, `trust-anchor`, `log-artifact`
Capability (`CAP`)	A tool, API, or execution ability an agent or the system can exercise	`retrieval-tool`, `rag-tool`, `egress-tool`, `action-tool`, `write-tool`, `computation-tool`, `code-execution`, `mcp-endpoint`, `decision-engine`, `persistence-ability`, `shared-primitive`
Communication (`COM`)	An inter-component channel	`routing-message`, `result-channel`, `control-message`, `llm-channel`, `tool-channel`, `ipc-channel`, `external-channel`, `injection-channel`, `shared-memory`
Downstream (`DOWN`)	An outward, consumer-facing effect	`user-output`, `artifact`, `operational-outcome`, `staged-action`

Inventory row. Each asset is one row with `asset_id` (`CLASS-NN`, stable across revisions), `name`, `class`, `subtype`, `location`, `data_state` (one or more of `at_rest`, `in_transit`, `in_use`, or `n/a` for a pure ability), `producer`, `consumers`, `sensitivity` (`low`, `medium`, `high`, `critical`: the worst-case value at risk, not the likelihood) and `dfd_node_ids` (the link between the asset and the model).

Authoring. An inventory is supplied by the system owner. Controls and approval gates are mitigations, not assets. An agent's goal or intent is not an asset as such; its value lives in its prompts (Information) and its tools (Capability). Every tool is its own Capability row. The same tool class constructed with different parameters is two assets when the parameters change its capability. A remote MCP endpoint with server-defined tools is one `mcp-endpoint` row. A store used as a cross-component channel yields two rows: the content (Information) and the conduit (Communication, `shared-memory`). Dormant credentials and capabilities are inventoried. For example, in Finance the portfolio file is both an `INFO` row (`agentic-memory`, `critical`: once poisoned, it steers every later decision) and a `COM` row (`shared-memory`), and the remote market-data server is one `CAP` `mcp-endpoint` row. How the inventories of the evaluated applications were produced is reported in Appendix F.

Asset anchoring. An authored inventory carries no anchors (every `dfd_node_ids` cell is empty). The Agentic analyzer proposes them during system modeling (Section 3.3): for each row, one or more DFD node ids, one or more entry-point or exit-point ids, or an empty list with a fixed reason, either `outside_modeled_surface` (the asset sits in no attacker-relevant flow) or `not_present_in_variant` (the code path does not exist as scanned), with a one-line rationale. The anchoring rules are: a system prompt or in-run state anchors to its agent; store contents to the `data_store`; a tool capability to the tool node; user data to every entry point it arrives on and the first node behind each; and a downstream asset to the exit point it leaves through, which for an advisory exit is often its only anchor. A threat's `asset_at_risk` copies `asset_id`, `name`, `class` and `subtype` verbatim from the inventory row, and its sensitivity is used in the risk assessment. For example, in Email Assistant, `INFO-01` (inbox content, `user-data`, `high`) is anchored to the `tool_response` entry point of `GmailSearchTool` and to the tool; `CAP-02` is the mailbox read (`retrieval-tool (gmail-read)`: the subtype encodes the tool's primary function, and its outbound query is carried by the advisory exit point, not by the subtype); and `DOWN-01` is the created draft (`staged-action`).

A.8 Attack vectors and paths

An attack vector describes one exploited component and mechanism; an attack path is the ordered DFD route from an entry point to an exit point or an internally completed adverse outcome. A threat may contain several vectors on one route.

Table 11: Vector and path fields within a threat record. The full schema is in Table 18.

Field	Meaning
`attack_vectors[]` **(one or more)**	
`exploitable_component`	DFD node at this step.
`component_type`	Its category and subtype.
`control_mechanism`	`llm_driven` or `deterministic` for a process node; otherwise absent.
`attack_mechanism`	How the step is exploited, such as indirect prompt injection or tool-invocation abuse.
`attack_surface`	Boundary-crossing category at this step; absent for an intra-zone step.
`attack_path` **(one per threat)**	
`path_components`	Ordered node IDs; each consecutive pair must be a DFD edge.
`entry_point_id`	Originating entry; absent only for an explicitly internal-origin finding.
`exit_point_id`	Exit through which the adverse effect occurs; absent for internal termination.
`action_authority`	Must equal the referenced exit's tier; absent for internal termination.
`internal_termination`	Sentence explaining an adverse effect completed inside the system; mutually exclusive with an exit ID.

The route, its entry, and its termination are part of a specific adverse-event claim. Two records with the same asset and exit but different entry points may call for different controls and remain separate under the deduplication identity of Appendix C.2. The source PDF's Email Assistant example calls draft creation "customer-data exfiltration"; that claim requires a recipient or attacker-accessible route beyond the shown reversible draft. In this appendix, the demonstrated route is described as attacker-induced creation of a draft until such a route is supplied.

B System Modeling Details

B.1 The intermediate representation(IR)

The framework-specific Scanner emits a normalized intermediate representation (IR) that the framework-independent builders consume. Table 12 lists its fields and closed vocabularies.

The IR is framework-neutral by construction: framework nouns (class names, decorators, configuration keys) may appear only inside quoted evidence text, never as field values, so nothing in the builders knows which framework the application was written in.

Table 12: Fields and closed vocabularies of the IR.

Field	Content and vocabulary
`components`	One entry per implementation-derived component. Type: `agent`, `llm`, `tool`, `mcp_server`, `mcp_client`, `external_service`, `data_store`. Capability flags `code_execution`, `read_external`, `write_external`, `read_internal`, `write_internal`; agent role, goal and system prompt; LLM provider and model; tool class; source file and line.
`connections`	Directed `source_id` to `target_id`, typed `tool_use`, `delegation`, `data_flow`, `llm_call`, `external_access` or `task_flow`. One direction only; the builder adds the return edge of every call.
`entry_points`	`is_task_input` (the value reaches an LLM context); `receiving_component_id` (first component inside the system); `external_source_node` (the origin, or null).
`exit_points`	Type: `file_write`, `http_outbound`, `email_outbound`, `database_write`, `tool_external_write`, `system_output`, `subprocess`, `other`; `component_id` (last component inside the system; empty when the exit sits outside of the modeled surface).
`process_type`	`sequential`, `hierarchical`, `graph`, `selector` or `unknown`; `unknown` is carried as such when the topology is not statically decidable.
`handoff_items`	The worklist: one item per decidable subject (Appendix B.2).

The Scanner (i) never executes or imports the analyzed application; (ii) never reads secret files: the names of environment variables are evidence, their values are not; (iii) never guesses an entry anchor: an external-origin entry it cannot anchor keeps an empty `receiving_component_id` and raises a handoff item; and (iv) never enumerates components built inside a loop or comprehension, whose number and identity depend on run-time data; it marks the region as a lower bound and raises a `completeness_limit` item instead.

The Model builder consists of a DFD builder, an entry-point builder and an exit-point builder. Every artifact is a deterministic function of the IR, obtained by applying the rule tables of Appendix A. When the agent graph has no clear source or sink (e.g., a cyclic supervisor loop), the builder raises a handoff item instead of naming a supervisor. Correcting the model therefore means editing the IR and rebuilding, never patching an artifact. Repeated builds of the same IR are byte-identical, which makes the base system model identical across runs over the same source code. For example, the deterministic pass over Trip Planner yields a DFD of 15 nodes (3 agents, 6 tools, 3 LLM engines, 1 LLM provider, 1 external service and the user) in 6 trust zones, with 34 edges and 16 boundary crossings; 15 entry points (12 agentic); 10 exit points; `process_type` `sequential`; and 7 open handoff items.

B.2 The handoff protocol

A handoff item is one question with a standing default, in a fixed schema. Table 13 shows the schema on one item, field by field. This subsection gives the schema, the six item kinds, the evidence-basis labels and the item states.

Evidence basis. Each default carries one of four evidence-basis labels (Table 15). There is no numeric confidence, because no calibration source exists. The basis carries evidence strength and `effects` carries impact, and the Agentic analyzer orders its work by kind and impact.

Item states. Every item starts `open`. After the Agentic analyzer propose an answer it is resolved after it passes validation and applied to the IR by the IR validator, and stays `open` otherwise: a rejected or unresolved answer changes nothing. An item the agentic analysis declines to decide, because the answer is a policy rather than a fact of the code, stays open with the module's rationale attached for the analyst. In the evaluated applications one such

Table 13: The handoff item schema, with one item as the example.

Field	Meaning	Example: Trip Planner's web-scraping tool and its outbound call
`id`	Deterministic identifier built from kind and subject, not an ordinal, so items can be compared across runs of unchanged source.	`cls-exit/ScrapeWebsiteTool/ http_outbound/authority`
`kind`	Selects the answer schema (Table 14).	`classification`
`state`	`open` or `resolved`; an open item may carry the module's note for the analyst.	`open`
`subjects`	References into this run's artifacts; a component duplicated into N copies becomes N subjects.	exits `XP-001` and `XP-001-2` in `exit_points` (one per agent)
`question`	One sentence, readable by analyst and agent.	"Does `ScrapeWebsiteTool`'s outbound call change any state outside the system, or is it a read-only query?"
`action_taken`	What the deterministic pass already did.	"exit point kept as `irreversible_external` (the safe side)"
`default`	The standing answer if the Agentic analyzer never runs, with its evidence basis (Table 15).	value `state_changing_write`; basis `heuristic`; rationale "the code shows an outbound request, not whether it changes anything"
`candidates`	The admissible set; closed-choice kinds only.	`read_only_query` (the response is consumed, nothing outside is created, updated or deleted); `state_changing_write` (the typing stands)
`evidence`	File and line anchors with notes, for the agent's reading.	`agents.py:23`, "agent-invoked outbound call, matched by pattern `ScrapeWebsiteTool`"
`resources`	Hints: `source`, `asset_inventory` or `runtime`.	`source`
`effects`	Per-answer operations (Appendix B.3)	`read_only_query`: `set_exit_read_only` on `XP-001` and `XP-001-2`; `state_changing_write`: none

item occurred: whether creating a draft in the user's mailbox is a reversible effect. This is the system owner decision; we decided to make it irreversible, simulating an email box that managed using agents that might trusts drafts created by fellow agents and send it later, or even be compromised by reading it alone.

B.3 How answers change the system model

An accepted answer changes the IR through one of eight operations (Table 16 gives each operation's effect on the IR and its restriction). After the operations are applied to the IR, the Model builder regenerates every artifact. Every operation target is pre-bound in the item's `effects` at emission, so the agent pick an answer from existing options and does not name a new target.

E.g., Finance: the deterministic analysis catalogued the command-line query as an entry point but could not tell whether its value reaches an agent (default `not_task_input`, basis `none`). The agentic analysis traced it into the router agent's task and answered `task_input`. The pre-bound `set_entry_task_input` set `is_task_input` on that entry point and the rebuild marked it agentic.

Table 14: The six item kinds, mapped onto the three sources of Section 3.3.

Kind	Source	Question shape; action already taken	Answer and operations
`classification`	semantic ambiguity	The subject is one of N enumerated categories and the discriminator is semantic; the conservative category was kept (write kept as consequential, exit graded as state-changing, entry unanchored).	A choice; `drop_exit_point`, `drop_entry_point`, `set_entry_task_input`, `set_entry_receiver`, `set_exit_read_only`.
`inference_disclosure`	low-confidence assertion	A field was asserted on stated evidence (LLM provider, store backend). Confirm or replace.	A verdict and replacement; `set_field`.
`completeness_limit`	visibility gap	A system model region is a stated lower bound (dynamic dispatch, wiring in loops and factories, user connectivity the builder refused, tools a remote MCP server exposes); modeled as far as visible.	The missing elements in IR vocabulary, and whether they complete the region or leave a stated lower bound; `add_elements`.
`merge_split_proposal`	low-confidence assertion	Stores were kept separate because identity is unproven. Do they share one backend?	Merge or keep separate, and which stores form one store; `merge_nodes`.
`exclusion_taken`	low-confidence assertion	The scanner saw an element and left it out for a stated reason (dead-code tool, credential cache, write-only store)	Uphold or reinstate; `add_elements` on reinstate.
`conflict`	semantic ambiguity	Two definitions of one name collided; the last one won.	The live definition; `set_field`.

Table 15: Evidence-basis labels.

`default.basis`	Meaning
`deterministic`	The deterministic pass established this.
`heuristic`	Asserted on stated but defeasible evidence.
`default`	A stated default, not a finding.
`none`	Static analysis hit a wall; the region is a lower bound.

B.4 Agentic analyzer

Table 17 gives, per handoff item type and for the asset inventory, what the agent decides and how its answer enters the IR. The agent has read-only access to the source and the run's artifacts, and built as a loop over the LLM API, not a framework's agent, so the analysis is never coupled to the framework of the system it audits. The agent also receive as context the framework pack of Appendix B.5, that tells it how to identify different MAS elements in systems implemented by it.

B.5 The framework pack.

The Agentic analyzer receives a framework pack containing framework-specific guidance. The pack explains how applications built with the framework, what evidence should be used to answer each type of question raised by the scanner, and where relevant behavior is usually implemented. It also includes worked examples, cautions about common sources of error, guidance for checks that are performed on every application, and the vetted library facts listed in Appendix B.4.

Each claim about the framework is identified either as an observation from the corpus, with a reference to the supporting example, or as documented library behavior. Documented

Table 16: The closed operation vocabulary.

Operation	Effect on the IR	Restriction
`merge_nodes`	Merges component entries, rewrites connection endpoints, records absorbed-to-survivor in the run's identifier map.	The only operation that renames anything; the scanner never merges, only the agentic analysis may.
`drop_exit_point`	Removes an exit-point row.	Matched on the row's identifying fields, never on its positional catalog id.
`drop_entry_point`	Removes an entry-point row.	As above.
`set_field`	Sets one field on the component backing the reference.	A field that is not an IR attribute lands in `metadata`; never renames a node.
`add_elements`	Appends an IR fragment (components, connections, entry points, exit points).	Fragment in IR vocabulary; screened before accepted.
`set_entry_task_input`	Sets `is_task_input`, from which `is_agentic` is re-derived.	Boolean only.
`set_entry_receiver`	Sets `receiving_component_id`, the entry's anchor.	Anchor-if-unanchored, idempotent, never a silent re-target: anchoring over a different existing receiver is refused. The reference must exist in the catalog and in the re-scanned IR.
`set_exit_read_only`	Flags a catalogued exit as a read-only query; the builder re-derives `action_authority` as `advisory`.	No value field and no upgrade direction; never writes `action_authority` itself; refused on `email_outbound` whatever the answer says.

Table 17: What the Agentic analyzer decides, by handoff item type and for the asset inventory.

Task	What the agent decides	Item kinds	Output
Semantic ambiguity	One of the item's admissible outcomes: is a write a persist or temporary, is an exit read-only, which agent first receives an input, does a tool return content, which colliding definition is live, does an exclusion stand.	`classification`, `conflict`, `exclusion_taken`	A choice per item, applied through the item's pre-bound operation.
Visibility gap	The elements a region marked as a lower bound is missing, in IR vocabulary: edges of dynamically dispatched hand-offs, attributed to the deciding agent; edges built in loops and factories; the user edges the model builder could not place; the tools a remote MCP server exposes.	`completeness_limit`	The missing elements, applied through `add_elements`; the answer may declare the region still incomplete.
Unverified claim	Confirm or replace an IR fact recorded on stated but unproven evidence (the LLM provider behind an engine, whether two stores share one backend).	`inference_disclosure`, `merge_split_proposal`	A verdict and the replacement value, applied through `set_field` or `merge_nodes`.
Asset anchoring	Not a handoff item: for each inventory row, the DFD nodes or entry and exit points the asset lives at, or an empty list with a fixed reason.	every row	The anchored inventory: every row linked to DFD node, entry point or exit point ids, or marked outside the modeled surface with its reason (Appendix A.7).

behavior may help the analyzer decide where to look, but it does not by itself establish that a particular question has been resolved. The framework pack does not change the analysis procedure. Its purpose is only to provide framework-specific context and to clarify what evidence is relevant. If no framework pack is available, the analyzer uses the generic procedure and records this in the run report.

B.6 Threat-record schema and an example threat

Every step in the threat enumeration phase follows the threat schema described in Table 18: the three identification modules emit threats in it, the Deduplicator merges within it, and the Threat validator checks conformance to it. Two conventions hold: (1) every identifier a threat carries (asset, DFD node, entry point, exit point, edge) must resolve in the system model of the same run, which is the first structural check of the Threat validator, (2) an optional field is absent when the producing module holds no value for it.

Table 18: Fields of a threat.

Field	Content
Identifier and title	Unique within the run, and a one-line name.
Producing modules	One entry per module that produced or confirmed the threat: the rule (Appendix C.1) or the module name. A merged threat carries the union.
Candidate flag	Set for a potential threat candidate until the Agentic ASI identifier resolves it.
Confidence	Low, medium or high, set by the rule or the agentic module. Not a risk score.
ASI labels	The ASI categories of the threat. Empty only for a threat of the MAS-taxonomy identifier with no ASI counterpart.
Adverse outcome	One or two sentences: what goes wrong, to which asset, and why it matters.
Asset at risk	One row of the asset inventory (identifier, name, class, subtype), its class one of the four of Appendix A.7.
Attack vectors	One or more, with the fields of Table 11.
Attack path	The route, entry point, exit point and action authority, or an internal termination, with the fields of Table 11.
Edge the threat is realized over	Optional. The edge the producing module names as the one the threat is realized over: the boundary crossing a rule matched, or the edge an agentic module cited. Read by the Deduplicator (Appendix C.2).
Severity signals	Inputs to the risk assessment (Section 3.5), not a score: the sensitivity of the asset from the inventory, the action authority reached, and the capability flags of the components on the path.
Evidence	Model references, one system-model identifier each, and code references, a file, a line and a verbatim quote. The Threat validator checks that each identifier exists and each quote matches the source.
Provenance	What later stages did to the threat: the titles and routes a merge absorbed, and the verdict and revisions of the Threat validator.

Table 19 transcribes one threat from the stored run of Instagram: the user's request enters the lead market analyst, an LLM-driven agent that owns a scraping tool, so injected instructions in the request can drive that tool within its granted permissions, and the tool's outbound request is the exit point through which the effect leaves the system. One rule produced it, `R5-T2-TOOL-MISUSE`, which labels T2 and adds T6.

C Threat Enumeration Details

C.1 The rulebook

The rulebook is the fixed rule set of the Rule-based ASI identifier (Section 3.4): it reads the observations of the system model analyzer and writes ASI-labeled confirmed threats and

Table 19: One threat of Instagram, from the stored run (abridged, without the risk scores and mitigations of the third phase).

Identifier, title	`THREAT-001`, Tool Misuse & Intent Breaking and Goal Manipulation
Producing modules	the Rule-based ASI identifier (rule `R5-T2-TOOL-MISUSE`)
Candidate flag	not set
Confidence	high
ASI labels	T2, T6
Adverse outcome	"An attacker steering 'Lead Market Analyst' can drive its side-effecting tool `scrape_and_summarize_website` within granted permissions, abusing that capability."
Asset at risk	`CAP-01`, the `scrape_and_summarize_website` tool, class capability, subtype scraping tool
Attack vector 1	`product_competitor_agent` (agent, LLM-driven), surface injection surface, "direct prompt injection: instructions in the user's request steer the agent's tool use"
Attack vector 2	`scrape_and_summarize_website@product_competitor_agent` (tool), surface exfiltration surface, "tool invocation abuse within granted permissions"
Attack path	`user`, `product_competitor_agent`, `scrape_and_summarize_website@product_competitor_agent`
Entry point	`EP-001`, the user's answer to "What is the product website you want a marketing strategy for?" at the chat interface
Exit point, authority	`XP-001`, the outbound HTTP request of the scraping tool, advisory
Internal termination	none declared
Edge realized over	`product_competitor_agent` to `scrape_and_summarize_website@product_competitor_agent`, tool invocation
Severity signals	asset sensitivity medium, action authority advisory, capability flags write external
Evidence	model references `product_competitor_agent`, `scrape_and_summarize_website@product_competitor_agent`, `EP-001`, `XP-001`, `CAP-01`, no code reference
Provenance	none (no merge, no revision)

potential threat candidates. A rule searches for a structural pattern over the system model (a component subtype, a capability, a boundary crossing of a given category, a bounded path shape) that satisfies a path condition over the observations (e.g., the path includes an asset of a given class, or ends in an `injection_surface` crossing). Its result carries the ASI category label the rule declares, a possible additional category label (e.g., `T2 Tool Misuse + T6 Intent Breaking and Goal Manipulation`) and links to the system-model elements and the asset it matched. It emits a confirmed threat when all the conditions required for the threat to be realized can be checked on the observations. It emits a potential threat candidate when the conditions it checks are necessary but not complete: the remaining are facts the system model does not record, such as whether a human approves an action before it is taken or whether an agent validates what it receives from a peer. A potential threat candidate carries the unresolved conditions to the candidate verification of the Agentic ASI identifier. Every output names the rule and the observation and system-model identifiers it matched (e.g., entry and exit point, attack path, attack surface), which is what the Threat validator later checks.

Table 20 lists the rules: what each fires on, the ASI labels it emits and its output kind. Table 21 gives the same catalog per ASI label.

Table 20: The rule catalog. Output T is a confirmed threat, C a potential threat candidate. A label in parentheses is added only under the condition the row gives. Table 21 details each ASI label.

Rule	ASI labels	Fires on	Output
`R1-T1-MEMORY-WRITE`	T1	a data store written over a `memory_poisoning` crossing by a tainted writer	T

Table 20, continued

Rule	ASI labels	Fires on	Output
R2-T1-MEMORY-RECALL	T1, T5	a poisoned store read back over `data_retrieval` into an agent, engine or tool	T
R3-T1-MEMORY-SHARED	T1, T12	a poisoned store read by two or more agent zones	T
R4-T1-UNTRUSTED-STORE	T1, T6	a store holding content authored outside the system, read into the reach of an LLM-driven agent (standing)	T
R5-T2-TOOL-MISUSE	T2, T6	a tainted LLM-driven agent owning a tool with a write or code-execution capability	T
R6-T3-PRIVILEGE-DELEGATION	T3	a tainted handoff into an agent with a larger capability set or a higher reachable authority	T
R7-T3-CONFUSED-DEPUTY	T3 (T14)	a user input through agent A into agent B that writes externally or reaches an irreversible exit, with no re-authorization at B (unobservable), T14 added when the path passes two or more agents	T
R8-T4-RESOURCE-LOOP	T4	a hierarchical or graph process, or a cycle through a tainted agent, with external or tool calls per iteration	T
R9-T4-FANOUT	T4	a tainted agent delegating to three or more agents	T
R10-T4-UNCAPPED-ITERATION	T4	a tainted agent with an iteration-driving setting and no stated bound	T
R11-T5-HALLUCINATION-CHAIN	T5	two chained LLM-driven agents with no observable validating step between them	C
R12-T6-MANIPULATION-POINT	T6 (T2)	an LLM-driven agent reached by taint, whose control decision can be steered, T2 added when the agent owns a side-effect tool	T
R13-T7-AUTONOMOUS-ACTION	T7	an LLM-driven agent reaching an `irreversible_external` exit point with no human gate on the path	C
R14-T9-A2A-TRUST	T9, T12	an agent-to-agent handoff with no identity control on the edge (unobservable)	T
R15-T9-EXTERNAL-INGRESS	T9	an `injection_surface` from an external service, or a `tool_poisoning` crossing, with no authentication of the source (unobservable)	T
R16-T10-HITL-PRESSURE	T10	a human-approval gate with three or more agents fanning in, or downstream of a T4 loop or fan-out	C
R17-T11-CODE-EXECUTION	T11, T2	a tainted component with code execution (eval, exec, subprocess, interpreter)	T
R18-T12-COMM-POISONING	T12	a tainted agent handing off to a peer agent over `taint_propagation`	T
R19-T13-ROGUE-ORIGIN	T13	a semi-trusted agent with a lateral handoff to a peer or write access to a shared store, with no entry-point taint (standing), one threat per asset it reaches	T
R20-T15-HUMAN-OUTPUT	T15	a tainted LLM-driven agent whose output reaches the user	T
R21-T16-PROTOCOL-ABUSE	T16, T9, T12	an MCP or A2A protocol channel into a tainted receiver, or a `tool_poisoning` crossing	T
R22-T17-SUPPLY-CHAIN	T17	an MCP server, tool or LLM engine whose code, tool or model is defined by a third party (standing)	T

Conventions and edge cases Three rules are keyed on presence rather than on attacker input: R4-T1-UNTRUSTED-STORE, R21-T16-PROTOCOL-ABUSE on a `tool_poisoning` crossing, and R22-T17-SUPPLY-CHAIN. Each names a dependency outside the system whose vulnerability the adversary can exploit or whose use it can induce (Section 3.1). For T13 (see Table 21 for full category name), we have a special rule R19-T13-ROGUE-ORIGIN that needs no entry-point taint: it starts from any semi-trusted agent and reports the assets that agent can reach through a peer or a shared store. Two labels depend on already founded instances rather than on a rule alone. T14 is added to a threat whose attack path starts at a direct user input and passes through two or more agents before reaching a capability the first agent lacks, and the T2 label of R12-T6-MANIPULATION-POINT is added only when the steered agent owns a side-effect tool. T8 has no rule: the implementation and possibilities for logging are vast and best discovered by an LLM agent, that can also decide whether the existing

logging is adequate and aligns with the system's needs based on semantic understanding of it. Table 21 counts, per ASI label, the rules that emit it and names the task of the Agentic ASI identifier that completes it: T7 and T10 are reached by candidates only, T5 by a candidate and one confirmed rule, and T8 by no rule.

Table 21: The ASI labels. # Rules is the number of rules that emit the label. C/T says whether those rules emit potential threat candidates (C) or confirmed threats (T). Completed by names the task of the Agentic ASI identifier that reaches the label beyond the rulebook. The questions of the MAS-taxonomy identifier map onto the same labels (Appendix C.1.2).

ASI label	Category	# Rules	C/T	Completed by
T1	Memory Poisoning	4	T	-
T2	Tool Misuse	3	T	-
T3	Privilege Compromise	2	T	-
T4	Resource Overload	3	T	-
T5	Cascading Hallucination and Misinformation Attacks	2	C, T	Agentic ASI identifier: candidate verification
T6	Intent Breaking and Goal Manipulation	3	T	-
T7	Misaligned and Deceptive Behaviors	1	C	Agentic ASI identifier: candidate verification
T8	Repudiation and Untraceability	0	-	Agentic ASI identifier: coverage completion
T9	Identity Spoofing and Impersonation	3	T	-
T10	Overwhelming Human in the Loop	1	C	Agentic ASI identifier: candidate verification
T11	Unexpected RCE and Code Attacks	1	T	-
T12	Agent Communication Poisoning	4	T	Agentic ASI identifier: coverage completion
T13	Rogue Agents in Multi-Agent Systems	1	T	Agentic ASI identifier: coverage completion
T14	Human Attacks on Multi-Agent Systems	1	T	-
T15	Human Manipulation	1	T	-
T16	Insecure Inter-Agent Protocol Abuse	1	T	-
T17	Supply Chain Compromise	1	T	-

Examples. Two outputs of the rulebook on Email Assistant, one of each kind. Confirmed: the content of a mail thread, fetched by a tool, enters the drafter, an agent that owns the draft-creation tool. `R12-T6-MANIPULATION-POINT` fires on any agent reached by taint, so it emits a confirmed threat labeled T6, and adds T2 because this agent owns a side-effect tool. Candidate: mailbox content enters the fetcher agent, whose output becomes the categorizer's input with no validating step the system model can show between them. `R11-T5-HALLUCINATION-CHAIN` cannot see whether the code checks that output before it is consumed, so it emits a candidate carrying that condition, and the Agentic ASI identifier confirms or rejects it based on the source code.

C.1.1 The ASI context

The system turn of the Agentic ASI identifier, compacted to the decision-relevant instructions. Field names of the threat schema are kept as the agent sees them. The inputs it receives in the user turn are listed at the end.

Role
You are a threat-enumeration agent in a threat modeling pipeline for agentic systems. You apply one taxonomy context to one system model. Taxonomy: OWASP Agentic Security

Initiative (ASI), T1–T17. Your role is to **verify**: resolve each potential threat candidate handed to you (confirm, drop or revise, via `submit_resolution`) and cover what the deterministic rules cannot decide (T8 by reading the code for an absence; T5, T12 and T13 across the whole system model). Resolve a candidate rather than emitting a duplicate threat. A drop verdict does not veto a threat emitted independently on the same surface by the MAS-taxonomy identifier; it is your judgement, surfaced for review.

Grounding contract (checked deterministically after you submit; a violation is a rejection, not a style note; you get one retry, then the emission is dropped)

- Every id you name (a DFD node, an asset, an entry point, an exit point) must already exist in this run's system model. Nothing invented resolves.
- An attack path is a node sequence checked edge by edge against the DFD. Emergent routes (cycles, novel hops) are allowed as long as each consecutive pair is a real edge.
- A claimed action authority must equal the authority of the exit point you reference.
- The edge the threat is realised over, when you give it, must be a real DFD edge. Give it whenever the threat is about one handoff, read, write or entry. Leave it off when the threat is about a component rather than a connection.
- Code evidence is a file, a line and a quote, the quote verbatim from that source line; it is re-checked literally. Read the source before you cite it.
- A threat is one adverse outcome anchored to one asset. Do not order threats into a campaign timeline: that is an attack, and it is downstream. Mechanism is orthogonal to identity; put the "how" in the attack vectors, never in the threat's identity.
- Confidence is `low`, `medium` or `high`; there is no numeric confidence.
- Cite an exit point only when the harm leaves the system or changes external state through it. When the harm completes inside the system (a goal rewritten, a peer's context poisoned, task state shaped), give one sentence saying why in `attack_path.internal_termination` instead of citing an exit point.

Questions to examine

- **T8 Repudiation and Untraceability** *(precondition: the anchor reaches a reversible or irreversible exit point).* For each consequential exit point in the T8 obligations, read the code and evaluate: (1) is the action logged; (2) is inter-agent communication logged; (3) is decision reasoning captured, not just the action; (4) are logs immutable or tamper-evident; (5) is there cross-agent log correlation. Group the obligations by the absence you found, the single missing control at the place in the code that omits it: obligations that one code change closes are one absence. Within one absence, emit one threat per action site (a component that performs an action the absence leaves unattributable), not one per exit point. Each threat walks its own attack path to its site, cites the highest-authority exit point at that site, names the site and its action in the title and description, cites the file and line of the omission as code evidence, states the remedy in full, and adds one sentence that the same change also closes the other covered sites. T8 has no rule; this is the only module that covers it.
- **T7 Misaligned and Deceptive Behaviors.** For each candidate (an LLM-driven agent with a reachable reversible or irreversible exit point and no governing gate), judge whether its capability set and system prompt create conditions for misaligned or deceptive action: (1) explicit constraints in the system prompt; (2) whether external-write, code-execution or irreversible exit points could circumvent them; (3) self-modification or self-preservation incentives; (4) a high-impact domain (finance, safety). Resolve the candidate (confirm, drop or revise) rather than emitting a duplicate.
- **T10 Overwhelming Human in the Loop.** For each candidate, assess the human reviewer's cognitive load: (1) how many agents funnel decisions to the same human; (2) whether the design allows batching or prioritisation; (3) whether high- and low-risk decisions are distinguished; (4) whether approval fatigue could lead to rubber-stamping. Resolve the candidate; do not emit a duplicate.
- **T5 Cascading Hallucination and Misinformation Attacks.** For each candidate (a chain of two or more LLM-driven agents with no validating step), assess whether a

hallucination would cascade: (1) output-validation steps; (2) multi-source verification of critical facts; (3) memory feedback loops that self-reinforce; (4) the same model used for generation and verification (correlated failure). Resolve the candidate, or, if the cascade spans several paths and is not a single candidate, emit a new T5 threat.

- **T12 Agent Communication Poisoning** *(precondition: the anchor is an agent).* Examine agent-to-agent communication: can one agent's output poison a peer's context across a taint-propagation crossing, and does any receiver validate inbound peer content? Emit one T12 threat anchored to the poisoned inter-agent channel, citing the crossing.
- **T13 Rogue Agents in Multi-Agent Systems** *(precondition: the anchor is an agent).* Per agent, a blast-radius analysis: (1) if compromised, which capabilities and which peers can it influence; (2) taint-propagation paths to other agents (peer infection); (3) write access to shared stores (shared-state poisoning); (4) external write or code execution (autonomous harmful action); (5) third-party plugins or remote MCP servers (compromise vector). Resolve the T13 candidate, or emit a new T13 threat where the cross-path blast radius is not captured by the compromised-agent seeding.

Termination. When you have examined every question and emitted every grounded finding, end your turn without calling a tool.

Inputs (user turn). The system model: DFD nodes with element subtype, parent, trust level, trust zone and capability flags; the anchored asset inventory; entry points with their receiving nodes; exit points with their action authority; boundary crossings. The observations: tainted-reachable nodes and shared stores. The threats already emitted by the Rule-based ASI identifier (extend, do not duplicate). The scoping payloads: the T8 obligations (consequential exit points with their components), the candidates to resolve, and the T5/T12/T13 hand-offs. **Tools.** `read_file`, `grep`, `list_dir` over the application source (read-only); `submit_threat_record`; `submit_resolution`.

C.1.2 The OWASP MAS context

Role
You are a threat-enumeration agent in a threat modeling pipeline for agentic systems. You apply one taxonomy context to one system model. Taxonomy: OWASP Multi-Agent System Threat Modeling Guide. Your role is to **discover**: sweep the system model against this taxonomy and emit a threat for each real instance you can ground. You do not resolve candidates; you have no resolution tool. Where your finding overlaps a threat of another module on the same surface (asset, outcome, entry point, realised edge, exit point), emit it anyway with your native taxonomy label; the Deduplicator merges the taxonomy views into one threat.

Grounding contract. As for the Agentic ASI identifier.

Questions to examine

- **Tool misuse via delegation or chain of command.** Sweep for an agent's authorised tools being triggered through delegation or orchestration rather than a direct user call, to run unauthorised operations. Anchor to the misused tool or capability; name the delegating and the executing agents.
- **Cross-agent privilege escalation.** Sweep for an agent gaining, via delegation or a shared credential, capabilities beyond its own: a confused deputy where a downstream agent executes a privileged action under its own authority without re-checking the original principal.
- **Shared-memory or vector-store poisoning.** Sweep for a data store written by one agent and read by others, where poisoned content would steer downstream reasoning. Anchor to the shared store; name the writer and the reading agents.
- **Third-party corpus or RAG poisoning.** Sweep for a document or RAG corpus, any store whose content is authored outside the system, read into an agent's

reasoning. Ask who can author or modify it in deployment (customer-supplied or operator-curated), whether retrieved content carries instruction-shaped text, and what validation the system model shows on that read path. Report what is modelled, never infer an absence. Anchor to the store; name the reader and the agent. Shared-memory poisoning is the writer case; both can hold in one application, on different stores.

- **Cascading hallucination across agents.** Sweep for a chain where one agent's unvalidated output feeds another's input with no verification step, so a hallucination propagates and amplifies across the chain.
- **Intent breaking and goal manipulation.** Sweep for a point where tainted content reaches an LLM-driven agent's context and could rewrite its goal or plan (a manipulation point). Anchor to that agent.
- **Inter-agent identity spoofing or impersonation** *(precondition: the edge is an inter-process channel).* Sweep for identity claims carried over a channel a third party can occupy (an agent-to-agent protocol edge, an MCP request or response, a message bus, any network transport) where the receiver accepts the sender's identity without authenticating it, or where one service account is shared across agents so one can act as another or as the user. An in-process handoff is not such a channel: the sender is the same interpreter, so report it out of scope rather than as an unauthenticated peer.
- **Agent communication poisoning.** Sweep for an agent-to-agent channel where one agent can inject manipulated content into a peer's context and the peer does not validate inbound peer content.
- **Malicious agent diffusion (worm-like propagation).** Sweep for an agent whose taint-propagation reach lets a compromise spread agent to agent, system-wide. Anchor to the highest-blast-radius agent.
- **Resource exhaustion or flooding.** Sweep for a looping or fanning topology the system model shows (a cycle through an agent, or one agent delegating to many) where per-iteration tool or LLM calls let flooding degrade service or enable a follow-on threat. Name the cycle or fan-out and what an iteration costs. Report what is modelled, never infer an absence: iteration caps, rate limits and timeouts are in-code guards you cannot read.
- **Overwhelming human in the loop.** Sweep for many agents funnelling approval decisions to one human with no batching or risk tiering, such that approval fatigue leads to rubber-stamping.
- **Insecure inter-agent protocol abuse.** Sweep only where the system model shows an MCP node (server or client), a `tool_poisoning` crossing, or an inter-agent edge over a protocol-handler entry point: protocol-level abuse of the framing or tool metadata a peer supplies. Name the node, the crossing and the capabilities exposed. Report what is modelled, never infer an absence: TLS, authentication and transport are code facts you cannot read. A boundary to an LLM provider is not this question; provider compromise is supply chain.
- **Supply-chain compromise (third-party tool, model or plugin).** Sweep for a third-party dependency (a remote MCP endpoint, a loaded plugin, a pinned model) whose compromise before the run gives standing attacker control of a capability. A standing threat.
- **Human attacks on or via the multi-agent system.** Sweep for a delegation chain that does not re-validate the original user's authorisation at each hop, or a path where a human can be manipulated by the system (or manipulate it): the human in the loop as an attack surface, not only a control.

Termination. When you have examined every question and emitted every grounded finding, end your turn without calling a tool.

Inputs (user turn). The system model and the observations, as for the Agentic ASI identifier. The threats already emitted by the Rule-based ASI identifier (extend, do not duplicate). No source code and no scoping payloads.
Tools. `submit_threat_record` only.

C.2 Deduplicator

The Deduplicator deterministically consolidates records produced by the three threat-identification modules. Deduplication proceeds in two step: the first removes records with the same threat identity, and the second reconciles records that describe the same exposure under different ASI labels.

First-pass identity. For a threat record t, the Deduplicator constructs the identity

$$I(t) = (\textit{asset},\ \textit{ASI labels},\ \textit{entry point},\ \textit{edge},\ \textit{exit point}).$$

The asset is resolved to its official inventory entry, and the ASI labels are compared as a set instead of order. The edge is represented by its source and target and its flow type is excluded because different producers may omit or describe this attribute differently. Titles, descriptions, attack mechanisms, and complete attack paths are not used in the identity because they are expressed at different levels of detail by the producing modules.

An omitted identity field is handled conservatively. Two omitted values do not, by themselves, establish a match. An internal termination is treated as an explicit exit value rather than as a missing exit point. Consequently, a threat whose effect remains within the system is kept separate from a threat that reaches an external exit point, even when the remaining identity fields agree.

Records with the same identity are grouped regardless of which module produced them. The merged record retains a representative title, description, and attack path. When the inputs assign different confidence values, the merged record retains the highest value.

Cross-label reconciliation. The first step cannot combine records that describe the same exposure using different ASI labels because the label set is part of the identity. The second step therefore compares the remaining records using the reduced surface

$$S(t) = (\textit{asset},\ \textit{entry point},\ \textit{edge},\ \textit{exit point}).$$

Records that agree on this surface, explicitly identify the edge, and carry different ASI label sets are merged when their labels belong to the same outcome class. The resulting record retains every ASI and MAS-taxonomy label assigned by the input modules. This permits, for example, a record labelled Tool Misuse (`T2`) and one labelled Intent Breaking (`T6`) to be combined when both refer to the same agent, authority, and interaction.

Table 22 gives the outcome mapping used in this pass. The mapping separates availability and accountability threats from the remaining labels. All other labels form a common default class and may therefore be reconciled when the four surface fields agree.

Table 22: Outcome classes used during cross-label reconciliation.

ASI label	Outcome class	Merge condition
`T4`	Availability	Only with another availability record
`T8`	Accountability	Only with another accountability record
All other labels	Default	With records in the default class

Two safeguards prevent the second pass from joining records that happen to share a coarse surface. First, an agentic record cannot act as a bridge between two distinct records produced by the Rule-based ASI Identifier. Second, rule-based records that omit the edge are kept separate unless they are repeated emissions with the same non-empty attack path. Records without an ASI label are not merged. The Deduplicator also reports groups withheld by these safeguards and groups merged across taxonomy views, allowing each decision to be inspected.

D LLM Usage: Configuration and Cost

We use LLMs at several points in the pipeline for structured analysis, threat generation and validation, risk assessment, and evaluation. In all pipeline stages, LLM outputs are treated as *proposals*: they are required to follow a predefined schema and are subsequently checked by deterministic or independent validation procedures before being accepted. Table 23 summarizes where LLMs are used and the corresponding acceptance procedure. All evaluated pipeline runs used `gpt-5.5` through the OpenAI API, with no temperature setting and a maximum output length of 32,768 tokens per call.

Table 23: LLM-assisted components in MASTRIA. Each LLM produces a structured output that is checked before being accepted into the pipeline.

Step	**LLM role**	**Structured output**	**Acceptance procedure**
Agentic analyzer	Resolves handoff items and anchors inventory assets using the system model, inventory, source, and file/search/listing tools.	IR update containing the answer, source locations, and operation.	Validated once; a failed submission is returned with the failing check and may be revised once.
Agentic ASI module	Verifies candidate threats and completes coverage using the ASI context, observations, candidates, system model, and source.	Threat-schema entry or candidate verdict.	Checked by the validation module; one retry is allowed.
MAS-taxonomy module	Applies the fourteen MAS questions to the system model and observations and maps MAS labels to ASI categories.	Threat-schema entries with MAS/ASI labels.	De-duplicated and checked by the validation module; one retry is allowed.
Validation module	Reviews each de-duplicated threat against the system model and source and, when necessary, revises its attributes.	Validation verdict or revised threat attributes.	Five structural checks are applied; after revision, all checks are rerun once.

Budgets and cost. Agentic components were subject to a 600,000-token budget, a 30-turn cap, and a run-wide ceiling of 20 million tokens. Per-threat risk-assessment calls do not use an agentic loop and allow at most two attempts. None of these limits was reached in the six evaluated runs, and no call was truncated. Table 24 reports aggregate LLM usage and API cost across the six evaluated runs.

Table 24: Aggregate LLM usage and API cost across the six evaluated runs.

Step	**Calls**	**Input tokens**	**Output tokens**	**Cost (USD)**
Agentic analyzer	378	7.45M	248k	20.20
Agentic ASI and MAS Taxonomy Identifiers + Threat Validator	250	2.18M	315k	13.47
Risk assessment	604	1.04M	167k	10.24
Total	**1,232**	**10.67M**	**730k**	**43.91**

E LLM-Assisted Evaluation

This appendix specifies the judging panel referenced in Section 4: how the panel was queried, how panel verdicts were aligned one-to-one with the human ground truth and what agreement they achieved, and portions of the judging prompt we released below.

Query protocol. Each judge evaluates one threat record per API call in a fresh context. A call contains the same four inputs: the application source code, the single threat record under review, the system model (data-flow diagram, entry- and exit-point catalogs, asset

inventory), and the ASI taxonomy reference. Judges are queried independently. Records are presented in a fixed order, and no judge receives more than one record per context, so a verdict cannot be conditioned on the base rate of verdicts already issued in the same run.

Three properties of the protocol bound what the panel can influence. First, judges cannot edit the threat records, the system model, or the source code. Second, judges are excluded from the construction of the human ground truth, which was fixed before any panel query was issued. Third, the generation pipeline and the judging panel are separated a judge never sees the generator's intermediate reasoning.

Human–panel agreement. We compare the majority-vote panel verdict with the corresponding human verdict using a one-to-one mapping over the 325 generated threat records (Table 25).

The two verdicts agree exactly on 318 of 325 records (97.85%): both assign *Correct* to 266 records, *Correct–Incomplete* to 42 records, and *FP* to 10 records. All seven disagreements are records that human reviewers labeled *FP* and the panel labeled *Correct–Incomplete*; the panel never disagrees with a human *Correct* or *Correct–Incomplete* verdict. Disagreements are therefore confined to the boundary between partially correct and incorrect records, reflecting a slightly more permissive panel rather than disagreement spread across label pairs.

Because the label distribution is highly skewed toward *Correct*, raw agreement alone overstates reliability. We therefore also report unweighted Cohen's κ, which corrects for the agreement expected by chance given each rater's marginal label distribution. Across the three verdict categories, the human and panel judgments achieve $\kappa = 0.930$ (expected chance agreement $p_e = 0.691$).

Table 25: Human (rows) versus majority-vote panel (columns) verdicts over the 325 generated threat records. Diagonal cells are exact agreements.

	Panel verdict			
Human verdict	Correct	Correct–I	FP	Total
Correct	266	0	0	266
Correct–Incomplete	0	42	0	42
FP	0	7	10	17
Total	266	49	10	325

Panel instructions. To make the evaluation procedure reproducible without reproducing deployment-specific prompt content, we provide below an compact version of the decision-relevant instructions supplied to the LLM judges. We omit file-handling instructions, CSV serialization requirements, and other operational details that do not affect the semantic judgment.

Task
Evaluate each supplied threat record for **precision**. Evaluate only the supplied threats; do not search for additional threats or measure recall.

Evidence standard
Treat the **application source code as the authoritative evidence** when deciding whether the threat claim is true.

- Do not assume that a generated threat is correct merely because it was produced by the threat-modeling pipeline.
- Treat an execution constraint or security control as present only when it is established by the supplied evidence.

Q1 — Threat claim
Using the threat description and source code, determine whether the claim is **true of the code as written**.

Do not repair an inaccurate claim during evaluation.

Q2 — Taxonomy mapping
Independently determine whether the supplied ASI taxonomy mapping correctly represents the threat claim.
Classify the mapping as:

Correct | **Incomplete** | **Over-mapped** | **Wrong**

according to the supplied taxonomy definitions.

Final verdict

Condition	Verdict
Core threat claim is unsupported	**FP**
Threat is supported and taxonomy mapping is fully correct	**Correct**
Threat is supported but taxonomy assignment is incomplete or otherwise requires correction	**Correct–I**

Provide a **concise, evidence-based justification** for the judgment.

F Evaluation Details

Evaluated applications We evaluated MASTRIA across six applications mentioned in Section 4. They span over different domains, architectures, and possible impact, and together cover a wide threat landscape. The following paragraphs will provide a short description of each application. The paragraphs below describe each application: its task, its agents and how they are organized. Table 26 places the applications along the selection dimensions of Section 4 and gives the size of the system model derived for each.

Email Assistant (Gandhi et al., 2026). Handles the user's mailbox on request: it fetches the matching emails, categorizes and prioritizes them, and creates reply drafts. Five agents run under a hierarchical process: an orchestrator delegates to a fetcher, a categorizer, a prioritizer and a drafter. The orchestrator hands the user's request to the fetcher, each worker's result feeds the next, and the drafter's output is a draft in the same mailbox.

Trip Planner (CrewAI, Inc., 2026). Creates a personalized seven-day travel guide with an itinerary for each day. Three agents run in a sequential chain: a city selection expert, a local expert and a travel concierge. The first picks the destination from the user's answers, the second builds the city guide, and the third expands it into the itinerary with a budget, each reading the previous agent's output.

HalluCheck (Solomon et al., 2026). A hallucination-tailored application: it is built so that a user can determine, with high confidence, whether a hallucination occurs in any of its agents. Three agents run in a fixed chain: the first answers a question from a given passage, the second verifies whether the passage supports the answer, and the third infers the original question from the answer and the passage, in an open-question and a multiple-choice variant selected per dataset row. Each run reads one row of a public dataset, passes the passage and question down the chain, and writes the verdict to a results file.

Finance (Levi et al., 2026). Answers portfolio questions and executes buy and sell orders against a portfolio from one command-line query. Six agents run in an LLM-driven routing loop of at most ten iterations: a router, four workers (market data, portfolio retrieval, order execution, valuation) and a final agent. In each iteration the router dispatches the query to one worker, the worker's result goes to the final agent, and the final agent either answers the user or sends the router a new instruction.

Instagram (CrewAI, Inc., 2026). Generates an Instagram post for a product from an e-commerce platform. Five agents run in two sequential pipelines chained by code: a market

analyst, a strategy planner and a content creator produce the campaign and the ad copy, then a photographer and a creative director produce and review the image brief. The user's two prompts seed the first agent, and each agent's output feeds the next across both pipelines.

RealEstate (Hancock, 2024). Analyzes a home-inspection report, identifies the issues, estimates repair costs and produces a homeowner summary. Three agents run in a sequential chain: a property analyst, a repair estimator and a homeowner advisor. The analyst retrieves the relevant passages from the user's PDF through a local vector store, the estimator prices the findings, and the advisor writes the plain-language summary to a report file.

Table 26: The six evaluated applications along the selection dimensions of Section 4. N, E, Z and C give the size of the derived system model: DFD nodes and edges, trust zones and boundary crossings.

Application	Topology	Entry points	Tools	N	E	Z	C
Email Assistant	hierarchical	user chat input; mailbox content	mailbox read; draft creation	18	52	8	30
Trip Planner	sequential	user prompts (4); search results; scraped pages	web search; web scraping; arithmetic by code evaluation	15	36	6	18
HalluCheck	sequential	public datasets (2)	–	15	28	9	16
Finance	routing loop	command-line query; remote MCP responses; portfolio state file	portfolio file read; buy and sell orders (local MCP); market data (remote MCP)	27	76	13	48
Instagram	sequential	user prompts (2); scraped pages; search results	web scraping; web search	33	96	9	58
RealEstate	sequential	user prompt or command line (PDF path, query); PDF content	PDF retrieval (RAG over a local vector store)	12	27	8	19

Model selection To evaluate the effect of the backbone model, we reran MASTRIA with Kimi-2.7 in place of GPT-5.5, while keeping the prompts, decomposition, threat schema, and evaluation procedure unchanged. Because Kimi-2.7 did not complete RealEstate, we compare the two models on the remaining five applications, containing 277 ground-truth threats.

On this common subset, GPT-5.5 achieved a recall of 0.978 and a precision of 0.944, compared with 0.693 recall and 0.831 precision for Kimi-2.7. The larger difference is in recall: Kimi-2.7 missed 85 ground-truth threats, whereas GPT-5.5 missed only 6. Precision also decreased, with Kimi-2.7 producing 39 false positives compared with 16 for GPT-5.5. These results indicate that the choice of LLM affects both the coverage and correctness of the generated threat records, with the strongest effect observed in threat coverage. Full per-application results are reported in Table 27.

Ablation study We conduct an ablation study to assess the contributions of agentic ASI module, agentic MAS module, and deduplication. Removing ASI correction while retaining independently generated ASI threats yields a average precision of 0.848 and recall of 0.885, indicating that ASI correction primarily acts as a quality-control stage that confirms, revises, and filters Tier-1 predictions. Removing threats generated by the agentic MAS module produces the highest precision (0.950) but sharply reduces recall to 0.478, with 165 false negatives, showing that this component is essential for expanding coverage beyond the conservative Tier-1 rules. In contrast, bypassing deduplication has bettet recall of 0.961 but reduces precision to 0.775 and produces 90 false positives, demonstrating that

Table 27: Model-backend ablation comparing GPT-5.5 and Kimi-2.7. Kimi-2.7 did not complete RealEstate because of a token-related error; the comparable subset therefore excludes RealEstate for both models.

		Correct		**Correct-I**		**FP**		**Missed**		**Recall**		**Precision**	
Application	$\lvert T_{\mathrm{GT}}\rvert$	**GPT**	**Kimi**	**GPT**	**Kimi**	**GPT**	**Kimi**	**GPT**	**Kimi**	**GPT**	**Kimi**	**GPT**	**Kimi**
Email Assistant	62	58	27	4	4	5	0	0	31	1.000	0.500	0.925	1.000
HalluCheck	36	33	21	2	2	0	15	1	13	0.972	0.639	1.000	0.605
Trip Planner	39	30	30	8	7	3	8	1	2	0.974	0.949	0.927	0.822
RealEstate	39	30	–	7	–	1	–	2	–	0.949	–	0.974	–
Finance	85	73	47	10	10	4	14	2	28	0.976	0.671	0.954	0.803
Instagram	55	42	38	11	6	4	2	2	11	0.964	0.800	0.930	0.957
Comparable subset (5 apps)	**277**	**236**	**163**	**35**	**29**	**16**	**39**	**6**	**85**	**0.978**	**0.693**	**0.944**	**0.831**

deduplication is important for consolidating overlapping or redundant threat reports. Overall, the three components play complementary roles: MAS module broadens threat coverage, ASI correction refines the quality of Tier-1 outputs, and deduplication controls redundancy and improves output precision.

Table 28: Ablation study results across applications.

Application	$\lvert T_{\mathrm{GT}}\rvert$	**Without ASI Module**						**Without MAS Module**						**Without Deduplicator**					
		Correct	Correct (I)	FP	FN	Precision	Recall	Correct	Correct (I)	FP	FN	Precision	Recall	Correct	Correct (I)	FP	FN	Precision	Recall
EmailAssistant	62	50	7	13	5	0.814	0.919	27	1	0	34	1.000	0.452	52	10	18	0	0.775	1.000
Hallucheck	36	31	2	4	3	0.892	0.917	13	0	0	23	1.000	0.361	31	4	13	1	0.729	0.972
Trip Planner	39	28	5	7	6	0.825	0.846	19	1	1	19	0.952	0.513	30	7	10	2	0.787	0.949
RealEstate	39	18	15	6	6	0.846	0.846	16	0	1	23	0.941	0.410	21	15	7	3	0.837	0.923
Finance	85	58	16	12	11	0.860	0.871	51	2	2	32	0.964	0.624	64	19	24	2	0.776	0.976
Instagram	55	38	12	9	5	0.847	0.909	20	1	4	34	0.840	0.382	40	12	18	3	0.743	0.945
Overall	**316**	**223**	**57**	**51**	**36**	**0.848**	**0.885**	**146**	**5**	**8**	**165**	**0.950**	**0.478**	**238**	**67**	**90**	**11**	**0.775**	**0.961**