[63] Deepseek, "deepseek-r1," 2025. [Online]. Available: https://api-docs.deepseek.com/news/news250528

[64] Microsoft, "playwright-mcp," 2025. [Online]. Available: https://github.com/microsoft/playwright-mcp

[65] mark3labs, "filesystem-mcp," 2025. [Online]. Available: https://github.com/mark3labs/mcp-filesystem-server

[66] T. Gao, X. Yao, and D. Chen, "SimCSE: Simple contrastive learning of sentence embeddings," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[67] T. Cao, B. Lim, Y. Liu, Y. Sui, Y. Li, S. Deng, L. Lu, N. Oo, S. Yan, and B. Hooi, "Vpi-bench: Visual prompt injection attacks for computer-use agents," *arXiv preprint arXiv:2506.02456*, 2025.

## Appendix A. Ethics Considerations

**Research Scope and Ethical Boundaries**. This research strictly adheres to ethical standards for security and AI system evaluation. The proposed attack scheme, TOMA, is designed and presented solely for the purpose of analyzing and improving the robustness of MASs. Our work aims to reveal the inherent and topology-driven security risks that are broadly applicable to MAS architectures, rather than exposing or exploiting any specific vulnerabilities of existing platforms. All experiments were conducted in controlled, locally hosted environments built upon open-source MAS development frameworks. No online, commercial, or third-party systems were accessed, tested, or influenced during any stage of this research.

**Experimental Control and Research Intent**. The attack implementations used in this study are purely experimental demonstrations to evaluate system-level resilience under realistic yet ethically constrained conditions. They do not contain or distribute functional exploit code targeting any real-world system. All results were obtained for academic and defensive research purposes, with the intent to inform the design of more secure and trustworthy MAS infrastructures. For ethical reasons, only the defense implementation and a video demonstration of the attack effects are included in the released artifacts.

## Appendix B. Topology Implementation

To evaluate the effectiveness of TOMA, we implemented five representative network topologies: tree, chain, star, ring, and mesh, across the MAGENTIC-ONE, LANGMANUS, and OWL frameworks. Red nodes denote attack entry points located at the network edge, while green nodes represent attack targets implemented using MCP. Each node is assigned a specific role based on its structural position and connectivity. An overview of the topologies is shown in Figure 9.

- **Tree Topology:** A hierarchical structure where V1 is the root coordinating top-down communication. Nodes V2 and V4 operate as intermediate relays, while V3 serves as the attack entry point. V5 is a leaf node designated as the execution target via MCP. This topology reflects a master-to-worker reasoning pattern, suitable for tasks requiring hierarchical instruction flow, such as structured multi-step planning or top-down information decomposition in LLM-based agents.
- **Chain Topology:** A linear structure from V1 to V7, where V1 is the attacker entry and V7 is the final execution node. Nodes V2–V6 act as sequential relays. This topology suits scenarios involving stepwise reasoning or progressive refinement, such as multi-turn dialogue pipelines or chained reasoning tasks distributed across LLM agents.
- **Star Topology:** A centralized pattern with V4 as the hub node managing communication with all other nodes. V1 is the entry point, and V7 is the execution target. The central node V4 aggregates and redistributes all data, making this topology ideal for centralized knowledge fusion, query distribution, or response ranking in language-agent orchestration.
- **Ring Topology:** A closed-loop communication structure enabling bidirectional message flow. V1 is the attack entry, and V7 is the execution target. Nodes V2–V6 serve as relays with alternate routing paths. This topology supports decentralized dialogue coordination and collaborative reasoning, where agents iteratively refine outputs or negotiate through language-based interactions.
- **Mesh Topology:** A densely connected directed graph with multiple redundant paths. V4 is the attack entry, linked only to V5, and V7 is the execution target. Nodes such as V2, V6, and V8 provide diverse routing paths for message propagation. This topology supports high-bandwidth, fault-tolerant coordination among LLM agents, and is well-suited for complex, multi-agent language reasoning tasks such as distributed question answering, multi-perspective summarization, or robust consensus generation in adversarial settings.

## Appendix C. Implementation of T-Guard

We detail the core implementation mechanisms underlying the topology-trust defense.

**Cross-modal validator**. The verification process of the cross-modal validator is defined in Algorithm 2. The implementation combines web automation, optical character recognition (OCR), and natural language processing (NLP) techniques to perform multimodal consistency verification. The *TakeScreenshot* function uses the *Playwright* library to launch a headless Chromium browser, navigate to the target URL, and capture a full-page screenshot. For text extraction, the *ExtractTextFromImage* function employs the *EasyOCR* library,

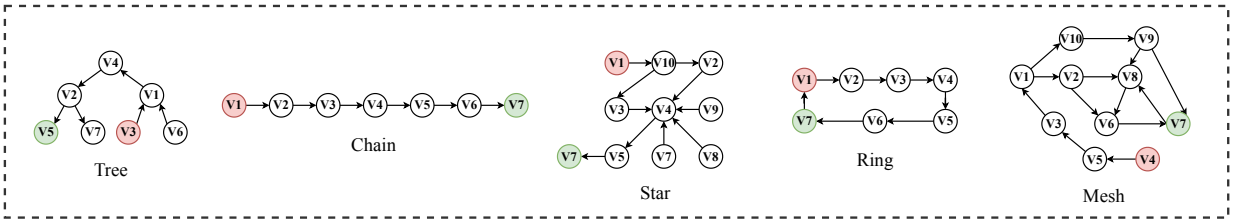


Figure 9: Illustration of five network topologies. Red nodes indicate attack entry points, and greens mark targets.

**Algorithm 2:** Cross-Modal Verification Process

**Input:** URL of the target page $u$, LLM-generated summary $S_{llm}$
**Output:** Semantic similarity score $sim$, Alert level $A$

```
// Capture visual content from the webpage
screenshot ← TakeScreenshot(u);
// Extract textual information via OCR
text_ocr ← ExtractTextFromImage(screenshot);
// Generate summary from extracted text
S_vis ← SummarizeText(text_ocr);
// Compute semantic similarity
sim ← CalculateSemanticSimilarity(S_llm, S_vis);
// Determine alert level based on similarity
if sim < 0.5 then
    A ← "High";
else
    if sim ≤ 0.8 then
        A ← "Medium";
    else
        A ← "Low";
return sim, A;
```

**Algorithm 3:** Taint Propagation and Trust Calculation

**Input:** Graph structure $G$, set of initial attacker nodes $A_0$
**Output:** Taint values $T$, Trust values $R$

```
// Initialization
Initialize all node taint values T[v] ← 0.0, ∀v ∈ G;
foreach v ∈ A_0 do
    T[v] ← 1.0;
// Iterative Propagation
for iter ← 1 to max_steps do
    T_prev ← T;
    foreach v ∈ G do
        N(v) ← GetNeighbors(v);
        mean_{N(v)} ← mean(T_prev[u] for u ∈ N(v));
        update ←
          (1 − T_prev[v]) × mean_{N(v)} × decay_factor;
        T[v] ← min(1.0, T_prev[v] + update);
    if |T − T_prev| < ε then
        break;
// Trust Calculation
foreach v ∈ G do
    R[v] ← 1.0 − T[v];
return T, R;
```

supporting both English and Simplified Chinese. OCR outputs with confidence scores below 0.8 are discarded, and the remaining text segments are sorted by their spatial coordinates to reconstruct the reading order. The *SummarizeText* function applies the *t5-small* model from *Hugging Face Transformers*, prefixing the input with "`summarize:`" to generate a concise visual summary. Finally, the *CalculateSemanticSimilarity* function leverages the *SentenceTransformer* framework with the *all-MiniLM-L6-v2* model to encode both summaries into embeddings and compute their cosine similarity. Overall, the cross-modal validator operates in a automated manner, enabling reproducible and reliable verification of visual–textual consistency in web content.

**Topological trust evaluator**. Upon detection of a medium- or high-level alert, the topological trust evaluator is activated to assess the reliability of system components. It operates based on the taint propagation model described in Algorithm 3, which simulates the diffusion of potential compromise across the agent task-flow graph. The agent system is represented as a bidirectional graph in adjacency-list form, generated by the *GraphExtractor* utility. The iterative taint propagation process is implemented in the *TaintPropagationModel*, configured with a decay factor of 0.05 and executed for a maximum of 100 iterations or until the change between successive updates falls below $\epsilon = 1\times10^{-4}$. Each node's trust value is computed as $R[v] = 1.0 - T[v]$, where $T[v]$ denotes the final taint value obtained after convergence. To enhance system resilience, a "guardian node" is selected following the evaluation. This node corresponds to the neighbor of the most vulnerable (i.e., lowest-trust) node that exhibits the highest taint value, thereby identifying the optimal candidate for enhanced monitoring or defense deployment.

**Dynamic policy updater and access control manager**. Following the trust assessment, the Dynamic Policy Updater generates a defense rule. This rule is then published for enforcement. A rule is generated based on the final taint value of the recommended *guard_node*. The rule is a structured JSON object containing a unique hash-based *rule_id*, a machine-readable *action* (e.g., *IMMEDIATE_QUARANTINE*), a *severity* level (e.g., *CRITICAL*), the *target_node*, a human-readable *reason*, and the full analysis *details* from the taint model. The generated rule is published to a simulated message queue (topic: *defense_rules*) and persisted to a shared file, *defense_rule.json*, which serves as the source of truth for the active policy. The Access Control Manager is a distributed responsibility in this proto-

TABLE 10: Adaptive Access Control Policies Derived from Taint Levels

| **Condition (Taint of Guard Node)** | **Action** | **Enforcement Logic** |
|---|---|---|
| $T > 0.8$ | *IMMEDIATE_QUARANTINE* | Block all operations originating from the target node to prevent potential compromise propagation. |
| $0.5 < T \leq 0.8$ | *RESTRICTED_OPERATION* | Permit only low-risk or read-only operations while isolating critical system interactions. |
| $T \leq 0.5$ | *LOG_SUSPICIOUS_ACTIVITY* | Allow normal operations but continuously log and report activity for post-analysis. |

type. Each agent queries the *defense_rule.json* file before operations to check for applicable policies. This self-enforcement mechanism is detailed in Table 10. This mechanism transforms abstract trust scores into concrete, system-wide access control policies. Following the trust evaluation, the *Dynamic Policy Updater* component translates the computed taint values into enforceable defense policies. Specifically, a policy rule is generated for the recommended *guard_node* based on its final taint value. Each rule is represented as a structured JSON object containing a unique hash-based *rule_id*, a machine-readable *action* (e.g., *IMMEDIATE_QUARANTINE*), a *severity* level (e.g., *CRITICAL*), the *target_node*, a human-readable *reason*, and the full taint analysis *details*. The generated rule is published to a simulated message queue (topic: *defense_rules*) and stored persistently in a shared file, *defense_rule.json*, which serves as the authoritative source for active policies. The *Access Control Manager* operates in a distributed manner across agents. Before performing any operation, each agent consults the *defense_rule.json* file to determine applicable restrictions. This decentralized, self-enforcing mechanism ensures that high-taint nodes are dynamically constrained according to predefined conditions summarized in Table 10. Through this process, abstract trust scores are effectively converted into actionable, system-wide access control rules, supporting real-time adaptive defense.