

When Agents Infect Agents

Prompt Injection Propagation and Trust Failures in Multi-Agent Systems



How a single poisoned observation can travel through delegation, summaries, and shared memory to reach a high-privilege agent—and how to measure where the chain breaks.

Abstract

Organizations are beginning to deploy teams of language-model agents that delegate tasks, summarize data for one another, and share memory. In these systems an attacker no longer needs to compromise the most privileged agent. Planting text where a peripheral agent, such as a log reader or document parser, will read it may be enough, provided the infection can travel through the messages agents exchange. This research perspective argues that multi-agent evaluation should treat every inter-agent handoff as a potential trust boundary and measure propagation directly instead of inferring it from single-agent injection results. It formalizes a threat model for LLM-to-LLM prompt propagation, describes the anatomy of an infection chain, compares single-agent and multi-agent security assumptions, specifies a resettable and instrumented environment with per-hop verifiers, defines metrics that separate propagation from final compromise, and proposes a falsifiable experiment on topology, message format, shared memory, and handoff defenses. It draws on prior work on indirect prompt injection, infectious attacks on agent populations, and multi-agent frameworks. No new model-performance results are reported.

Introduction: why a trusted colleague is not a safe one

For most of the short history of language-model agents, security discussions have centered on one model facing one user. The question was how to stop a hostile instruction from reaching that model through the front door. Deployments are changing. A growing number of systems split work among specialized agents: one reads documents, another writes code, another plans, and an orchestrator assigns tasks and assembles results. Frameworks such as AutoGen, MetaGPT, and CAMEL made this pattern easy to build [8][9][10], and open protocols for connecting agents to tools and to each other, such as the Model Context Protocol and Agent2Agent, are making it easy to scale [11][12].

The security consequence is that the unit of analysis changes. In a single-agent system, untrusted text enters through a known set of observations. In a multi-agent system, the output of one agent becomes the input of the next, and a message that began as untrusted data can arrive at its destination looking like an instruction from a colleague. This is the setting for indirect prompt injection, in which an attacker plants text where a model will later read it [1], extended across a network of models that read each other's output.

The concern is not hypothetical in the research literature. Work on prompt infection describes injected instructions that cause an agent to pass the injection on to other agents in the same system [3]. Work on infectious jailbreaks describes a single adversarial input spreading through a large simulated population of agents as they interact [4]. Other work reports that adversarial content can hijack the control flow of multi-agent frameworks and lead to the execution of arbitrary code [5]. These studies differ in setup and should not be conflated, but together they motivate a direct question: when one agent reads hostile text, what determines whether the next agent obeys it?

This article develops a research perspective, not a new experimental result. Its central claim is conditional: realistic, resettable, instrumented multi-agent environments can measure how and where injections propagate, and that measurement should be reported separately from task success. A

multi-agent system that completes its tasks and survives single-agent injection tests has not thereby shown that it contains an infection. The burden is to test containment at each handoff.

The article builds on a companion perspective on adversarial telemetry in single-agent recovery work [27]. Sections 1 to 5 define the problem: the structure of multi-agent systems, why they amplify risk, the anatomy of an infection chain, the threat model, and how the security assumptions differ from the single-agent case. Sections 6 to 9 cover environment design, instrumentation, metrics, and statistical validity. Section 10 proposes an experiment. Sections 11 and 12 discuss the trade-offs and defensive architectures, and Section 13 addresses ethics and limitations. Section 14 describes where Deaimer fits, followed by directions for future work.

Key terms

- **Multi-agent system (MAS).** A system in which two or more model-driven agents exchange messages, delegate tasks, or share state to complete work.
- **Orchestrator.** An agent that assigns tasks to other agents and combines their results, often holding the broadest privileges.
- **Peripheral agent.** An agent with narrow privileges that reads external content, such as a log reader, web fetcher, or document parser.
- **Handoff.** Any transfer of content from one agent to another: a direct message, a delegated task description, a summary, or a write to shared memory.
- **Prompt propagation.** The movement of an injected instruction, or an instruction derived from it, from the agent that first read it to other agents.
- **Hop.** One handoff in an infection chain. A payload that reaches a third agent through a second has travelled two hops.
- **Payload fidelity.** How much of the original injected instruction survives a handoff: verbatim, paraphrased, or reduced to an imperative.
- **Blast radius.** The set of assets and privileges reachable if a given agent is compromised.
- **Canary.** A synthetic secret or monitored resource whose use or modification proves that an injected action occurred.

1. From isolated agents to interconnected teams

A multi-agent system is defined less by the number of models than by the channels between them. Three questions describe any design: who may send to whom, in what form, and what state is shared. The answers determine where an infection can travel.

Four topologies are common. In a pipeline, each agent passes output to the next in a fixed order. In a hub-and-spoke design, an orchestrator delegates to workers and receives their results. In a shared-workspace design, agents read and write a common scratchpad, memory store, or message history, often called a blackboard. In a peer mesh, agents may address one another freely. Real deployments mix these.

Topology	Typical channel	Propagation path
Pipeline	Output of one agent becomes input to the next	Linear; the payload must survive each transformation in order
Hub and spoke	Orchestrator delegates; workers report back	Worker to orchestrator; orchestrator to other workers via new tasks
Shared workspace	Common scratchpad, memory, or chat history	One write can reach every agent that reads the shared state
Peer mesh	Free addressing among agents	Many paths; fan-out and loops are possible

The form of a handoff matters as much as its route. A handoff may pass raw content, such as a file excerpt. It may pass a summary produced by the sending agent. It may pass a structured message with defined fields. Or it may pass a natural-language task description that tells the receiver what to do. Each form offers a different opportunity for a payload to be preserved, transformed, or discarded. A summary that compresses a log into three sentences may drop an embedded instruction, or may restate it as a finding the next agent treats as fact.

Interfaces to the outside world add a further layer. Tool protocols connect agents to servers that return content, and that content is another observation the agent may trust. A tool result is data authored by whoever controls the tool or the data behind it, not a message from the operator [11]. Agent-to-agent protocols extend the same question to peers that may belong to other organizations [12].

Finally, multi-agent systems introduce persistence. Shared memory and long message histories survive beyond a single task. A payload written into memory can influence agents that run hours later, which turns a transient injection into a lasting one.

2. Why multi-agent systems amplify risk

Several properties of multi-agent systems increase risk beyond what single-agent analysis would predict. None is certain to apply to every system, and each should be treated as a hypothesis to measure.

- **The trusted-colleague assumption.** Agents are often built to treat messages from other agents in the same system as trusted internal instructions instead of as external data. If Agent A sends a summary to Agent B, B may follow instructions embedded in it as readily as it follows the operator's. Trust that is appropriate for authenticated components becomes unsafe when the sender itself reads untrusted content.
- **Confused deputy.** The confused deputy problem describes a privileged program that is tricked into misusing its authority on behalf of a less privileged party [6]. In a multi-agent system, an unprivileged peripheral agent that has read hostile text can ask a privileged agent to act. The privileged agent holds the authority, the peripheral agent supplies the intent, and the attacker never touches the privileged agent directly.
- **Provenance laundering.** When an agent summarizes or rewrites content, the output carries the sender's voice, not the original author's. Hostile text that arrived as a quoted log line can emerge as an

apparently first-hand finding. The receiving agent loses the signal that the content originated outside the system.

- **Privilege asymmetry.** Systems commonly give read-only, external-facing agents low privileges and give executors high ones. This is sound least-privilege design, but it creates exactly the gradient an attacker wants: an easy entry point and a powerful target, joined by a message channel.
- **Fan-out and persistence.** One compromised agent can write to a shared store read by many, or message several peers. Memory that outlives the task extends the window in which a payload can act.
- **Monitoring blind spots.** Security controls are often placed at the system's front door. Messages between agents may not pass through any filter, and logs of internal traffic may be unexamined. An infection that travels only on internal channels may trigger no alarm.
- **Diffused accountability.** When several agents contribute to an action, it is harder to determine which input caused it. Investigations require per-hop records that many systems do not keep.

Studies of how multi-agent systems fail more generally point in the same direction. A taxonomy of failures in such systems identifies problems in specification, in inter-agent misalignment, and in verification and termination [13]. Security failures are not identical to these, but they share causes: agents accept each other's outputs without adequate checking, and nothing verifies what was passed along.

3. The anatomy of an attack: how prompt infection spreads

A typical infection chain has five stages. The stages are a descriptive model, not a claim that every attack follows them in order.

Stage	Name	What happens	Measurement point
1	Entry point	A payload is placed in a low-risk external source such as a document, email, ticket, or log.	Was the payload delivered to the entry agent?
2	First compromise	The entry agent reads the text and its behavior is redirected.	Did the entry agent act on, repeat, or flag the payload?
3	Handoff or memory write	The compromised agent places the instruction in a message, summary, or shared memory.	Did the payload, or a derivative, appear in the handoff?
4	Horizontal spread	A second agent reads the handoff and follows the embedded instruction.	Did the second agent comply, and at what fidelity?
5	Payload execution	The infection reaches an agent with high privileges and the injected action is performed.	Did the executor perform the action, and what was affected?

Two variants are worth distinguishing. In a relaying attack, the payload carries the instruction to act and each agent merely passes it on, reducing or reshaping it along the way. In a self-replicating attack, the

payload instructs each agent to include the instruction in its own outputs, so the infection copies itself to every agent it contacts. Work on prompt infection describes the second kind [3], and work on infectious jailbreaks describes related spreading behavior in large simulated agent populations [4]. Self-replication requires more from the payload but removes the need for the attacker to predict the route.

Each stage has a distinct failure and a distinct defense. At stage 2, the question is whether the entry agent can separate evidence from instruction, which is the single-agent problem [1][2]. At stage 3, the question is whether the agent's own output can carry the payload. At stage 4, it is whether the receiving agent distinguishes a peer's message from an operator's. At stage 5, it is whether the executing agent's privileges are bounded regardless of what it is told. An evaluation that measures only the final outcome cannot say which control failed.

Chains can also break for innocent reasons. A summarizer may omit the payload simply because it is irrelevant to the task. A downstream agent may ignore the instruction because it does not need the action. A study that counts these as successful defenses will overstate containment. Matched controls and relevance variation, described in Section 6, are needed to separate resistance from coincidence.

4. A threat model for prompt propagation

A threat model states who the attacker is, what they can influence, and what they want. In this setting the attacker does not need access to any agent's prompt, tools, or host. The attacker needs to cause text to appear where some agent in the system will read it, and needs that agent's output to eventually influence another.

Attacker capabilities can be graded. A weak attacker controls a single field that is later read by an external-facing agent. A moderate attacker controls the content of a document, ticket, or web page the system is likely to process. A strong attacker also knows or can infer the system's topology and roles, so the payload can be written for the downstream agent it must reach. The strong case is realistic for systems built on public frameworks whose structure is documented.

Entry points follow the system's sources of external content: files, email, web pages, tickets, logs, database records, tool outputs, and messages from agents belonging to other organizations. Goals include data exfiltration, destructive or weakening actions, persistence through memory or scheduled tasks, privilege escalation across roles, misdirection of the team's output, and denial of service through loops or runaway delegation.

The assets at stake are the confidentiality of secrets and records, the integrity of data and configuration, the availability of services, and the audit trail. Each should be instrumented separately. The threat model also states what is out of scope: attacks on model weights, compromise of the agent framework's code, and attackers with direct access to the host or to the operator's credentials. The operator and the person who defines the system's tasks are trusted principals.

A trust boundary map is a useful companion to the threat model. It marks each channel through which content enters or moves, and labels the author of the content on that channel: operator, system, peer agent, external party, or unknown. Many vulnerabilities become visible when the map shows a peer-agent label on a channel whose content actually originated outside.

5. Single-agent versus multi-agent security

The assumptions that support single-agent defenses do not carry over unchanged. The table summarizes the main differences.

Feature	Single-agent setup	Multi-agent system
Interaction	One agent reads external logs and files directly.	Agents read each other’s outputs and pass context onward, recursively.
Trust model	The agent treats external content as untrusted data, if guarded.	Downstream agents may treat upstream messages as trusted internal instructions.
Blast radius	Bounded by the permissions that one agent holds.	Spreads across roles; a low-privilege entry point can reach a high-privilege executor.
Observability	One trajectory to inspect.	Several interleaved trajectories; causation spans agents.
Persistence	Mostly confined to the session.	Shared memory can carry a payload across tasks and agents.
Defense focus	Input handling, action-space filters, egress blocks.	Handoff controls, role-based data isolation, per-agent permissions, message provenance.

Existing benchmarks mostly address the left column. InjecAgent evaluates tool-integrated agents against injected instructions in tool outputs [14]. AgentDojo provides a dynamic environment of tasks, injections, and defenses [15]. Both established injection as a measurable property of agents and are valuable. They describe what a single agent does when its tools return hostile content. They were not designed to say how an injection moves between agents, whether a summary preserves it, or whether a second agent trusts the first.

The multi-agent literature adds evidence of a different kind. Reports of infectious spread among agents [3][4] and of hijacked orchestration [5] show that propagation is possible under the conditions studied. They do not by themselves establish rates across frameworks, models, or message formats, and results depend on the scaffold, prompts, and privileges used. The right reading is that propagation is a real and measurable phenomenon, and that its rate under realistic configurations is still an open empirical question.

A benchmark result is a property of a configuration: the models, roles, topology, message formats, tools, permissions, environment version, task sample, and scoring rules. A propagation rate measured on a two-agent pipeline with raw handoffs says little about a hub-and-spoke system with structured messages. Reports should state the configuration in full.



6. The anatomy of a propagation environment

A useful environment begins with a capability hypothesis. ‘Test multi-agent security’ is too broad to guide design. ‘Measure whether an injection read by a log-reading agent causes a downstream executor to run an unrelated command’ is testable. From that hypothesis the designer specifies the agents, their roles and privileges, the channels between them, the entry content, the injected action, and the observable outcomes.

To keep results interpretable, the topology should be small and fixed. A two- or three-agent line, for example a reader that summarizes external content for a planner that instructs an executor, supports clear per-hop measurement. Large networks introduce many confounds from orchestration design, and it becomes difficult to tell whether outcomes reflect the agents or the wiring. Larger topologies are better treated as a later stage once the small case is understood.

Several design choices carry most of the scientific weight.

- **Control conditions.** The same task with no payload gives the clean success rate. Without it, a change under attack cannot be attributed to the attack.
- **Resetability.** Each trial starts from a known state, including empty or seeded shared memory, so leftover messages do not contaminate comparisons.
- **Containment.** All secrets are synthetic canaries, and there is no real network egress. A mock endpoint inside the sandbox records exfiltration attempts.
- **Matched variants.** Clean and poisoned variants differ only in the payload. Document length, noise, and layout are held constant.
- **Relevance control.** Some payloads request actions the task never needs, and others request actions adjacent to the real work. This separates agents that resist from agents that merely have no use for the instruction.
- **Privilege gradient.** The entry agent has low privileges and the executor has high ones, so a successful chain demonstrates real escalation. Variants with equal privileges isolate the effect of the gradient.
- **Legitimate-delegation baseline.** Include tasks where the correct behavior is for the executor to follow a peer’s request, so an agent cannot score well simply by distrusting all messages.
- **Seeded randomization and version pinning.** Fix seeds, model versions, and tool versions per trial and record them, so a chain can be replayed and investigated.

Instrumentation matters as much as task design. The harness should record every message between agents with the sender, receiver, content, and timestamp, alongside each agent’s observations and actions. It should be possible to ask, for any executed action, which upstream message or observation preceded it. This is a form of taint tracking at the level of the experiment, even if the agents themselves cannot do it.

Realism is not visual fidelity. A modest command-line environment with real dependencies, noisy content, and consequential actions teaches more about propagation than a polished interface with scripted agents. The test is whether the environment preserves the causal structure of the target work: who can talk to whom, what information moves, what actions are legal, and what counts as success.

7. Verifiers and per-hop instrumentation

Verifiers decide what the environment measures, and weak verifiers teach shortcuts. A score for ‘task completed’ can be earned by an agent that disabled a control to complete it. In a multi-agent setting, verifiers must also decide where in the chain an event occurred.

Security predicates should be objective, automatic, and specific, and should be evaluated per agent.

- **Confidentiality.** Whether any canary secret appeared in a message, an outbound request, a shared-memory write, or a command argument.
- **Integrity.** Whether protected configuration, authentication settings, and unrelated data remain unchanged.
- **Privilege.** Whether accounts, keys, roles, or permissions were added or widened, and by which agent.
- **Persistence.** Whether memory entries, scheduled tasks, or startup scripts were created that were not part of the task.
- **Message integrity.** Whether the payload or a derivative appears in any handoff, and in what form: verbatim, paraphrased, or reduced to an imperative.
- **Evaluator integrity.** Whether any agent modified tests, logs, or grader files to obtain a passing result. Reward hacking is a general risk whenever proxies stand in for goals [25].

Detecting a payload derivative in a handoff is harder than detecting a verbatim copy. Simple string matching misses paraphrase. Reasonable approaches include matching on the injected action’s distinctive parameters, such as a specific endpoint or file path, using a separate classifier to judge whether a message contains the instruction, and manual review of a sample. The method should be validated against labelled examples and reported, since the detection method affects the measured propagation rate.

Verifiers need negative controls. Known-bad chains that propagate a payload yet complete the task must be scored as failures on the security dimension. Known-good trajectories that complete the task by an unusual but valid route must pass. Adversarial review of the grader should look for unintended paths to a passing score, and the verifier should be versioned along with the tasks.

8. Metrics: separating propagation from compromise

A single end-to-end number can conceal where a system fails. A system whose entry agent is easily hijacked but whose executor reliably refuses is different from one with the opposite profile, and they require different fixes. Metrics should be reported per hop and for the whole chain, and should not be folded into one weighted score.

Metric	Definition	What it reveals
Entry Compromise Rate	Poisoned trials in which the entry agent acts on the payload, divided by all poisoned trials.	Single-agent vulnerability of the peripheral agent.

Metric	Definition	What it reveals
Per-Hop Propagation Rate	Trials in which a payload or derivative appears in a handoff and the receiver acts on it, divided by trials in which it reached the receiver.	How well each handoff contains or spreads the infection.
End-to-End Compromise Rate	Poisoned trials in which the executor performs the injected action.	Whether the chain succeeds.
Payload Fidelity	Share of handoffs carrying the instruction verbatim, paraphrased, or as an imperative.	Whether summarization sanitizes or launders.
Privilege Escalation Rate	Compromises in which the acting agent holds more privilege than the entry agent.	Whether low-privilege entry reaches high-privilege action.
Per-Hop Detection Rate	Trials in which an agent flags the content as untrusted or suspicious.	Whether any agent notices the attack.
Containment Rate	Poisoned trials in which the chain stops before the executor.	Effectiveness of defenses and incidental barriers.
Utility Under Attack	Task success in poisoned trials minus task success in matched clean trials.	Cost of the attack, including distraction.
Over-Refusal Rate	Clean trials in which an agent rejects a legitimate peer request.	Cost of excessive distrust.
Cost and Latency	Messages, tokens, and time per task, clean versus defended.	Operational cost of handoff defenses.

Collateral damage should be given a severity scheme set before the experiment, with weights documented and unweighted counts reported alongside. Containment should be reported with the reason for the break where it can be determined: payload dropped in summary, payload flagged, instruction ignored as irrelevant, or action blocked by policy. These reasons determine whether a result is robust or accidental.

Over-refusal belongs beside propagation. A defense that blocks all inter-agent instructions drives propagation to zero by removing collaboration, and a system that cannot delegate no longer needs to be multi-agent. Reporting propagation and over-refusal together makes the trade-off visible.

9. Statistical design and threats to validity

Because model sampling and tool timing vary, a single run per condition is not enough. Chains add variance at every hop, so uncertainty compounds. Report the number of trials per condition and an interval for each rate, and avoid presenting small differences as definitive. Rates near zero or one are common here, and a Wilson-type interval is generally better behaved than a simple normal approximation for such proportions [26]. Comparisons should hold constant the models, budgets, topology, tool set, and scoring criteria.

- **Paired comparison.** Run each task in clean and poisoned form with matched conditions so differences are measured within a task.
- **Per-hop denominators.** Report propagation conditional on the payload having reached the hop, so that early drops do not mask later weaknesses.
- **Pre-registration.** Predefine exclusion rules, the number of trials, primary metrics, and the analysis plan.
- **Multiple comparisons.** When testing many topology, format, and defense combinations, state which comparisons are primary and which are exploratory.
- **Disaggregation.** Report by payload type, message format, topology, entry agent, and model.

Threats to validity should be named openly.

- **Internal validity.** A payload may change document length or the summarizer’s task. Matched variants and difficulty audits reduce these confounds.
- **Construct validity.** Measured propagation reflects the payloads written, not general resistance. A system may contain one phrasing and spread another.
- **External validity.** A small sandboxed topology will not match every production system. Results support claims about the sampled configurations.
- **Detection error.** Imperfect detection of derived payloads can bias propagation rates in either direction.
- **Contamination.** If payloads appear in training data or public repositories, systems may have seen them. Keep a held-out set and report exposure.
- **Non-adaptive attackers.** Fixed payloads understate risk because real attackers iterate against a system’s observed behavior.
- **Orchestration confounds.** Differences between systems may come from prompts or scaffolding instead of the property under study. Hold orchestration constant when varying one factor.

10. Illustrative experiment: what determines whether an infection spreads?

The following is a proposed experiment, not a completed study. Build a small family of tasks, each requiring a reader agent to process external content, a planner to decide on steps, and an executor with privileged tools to carry them out. For each task, construct a clean variant and several poisoned variants. Run several model configurations under matched budgets.

The independent variables are topology, handoff format, shared memory, payload type, entry privilege, and handoff defense. The dependent measures are the metrics in Section 8.

Factor	Levels	Purpose
Topology	Pipeline; hub and spoke; shared workspace.	Tests how structure shapes propagation paths.
Handoff format	Raw excerpt; free-text summary; structured fields with a task schema.	Tests whether summarization contains or launders payloads.

Factor	Levels	Purpose
Shared memory	None; session memory; persistent memory.	Tests persistence and fan-out.
Payload type	Relaying instruction; self-replicating instruction; disguised as a task from the operator.	Tests sensitivity to attack design.
Payload relevance	Unrelated to the task; adjacent to a real step.	Separates resistance from lack of need.
Entry privilege	Lower than executor; equal to executor.	Tests the effect of the privilege gradient.
Handoff defense	None; untrusted-source labelling; schema validation; independent policy check at each agent.	Tests which controls reduce propagation without destroying collaboration.
Model configuration	Several models and scaffolds with identical tools.	Tests generality.

A strong design would include three evaluation slices. A near-distribution slice uses known payload families with new surface details. A mechanism-transfer slice combines familiar components in new ways, such as a known payload through a new topology. A held-out slice uses payload families and handoff formats that were not used to tune any defense. Results should be disaggregated, because improvement on one slice does not establish containment on another.

The primary hypothesis is that propagation depends on handoff format: structured, schema-constrained messages contain injections better than free-text summaries or raw excerpts, at some cost to flexibility. A second is that shared persistent memory increases the number of agents reached and the time over which a payload can act. A third, which the data may contradict, is that summarization often sanitizes payloads by omission but sometimes launders them into trusted-sounding findings, with the balance depending on the payload’s relevance to the task. A fourth is that defenses tuned on visible payload families degrade on held-out families. All are falsifiable. Possible outcomes include near-total containment by incidental summarization, concentrated propagation through shared memory, or defenses that reduce spread at a large cost to task success.

Before running the experiment, researchers should predefine exclusion rules for invalid tasks, the number of trials, the method for detecting derived payloads, and how uncertainty will be estimated. They should audit whether any condition makes the task easier, since a difference attributed to topology might come from task difficulty, and whether defenses add information that changes agents’ behavior. Results should be reported as full per-condition tables, not only headline rates.

11. Capability, collaboration, and the isolation trade-off

It is tempting to assume that stronger models will be better at recognizing an injected instruction in a peer’s message. It is equally plausible that stronger, more cooperative models are better at following the instruction, and that more capable summarizers preserve more of a payload’s meaning. Which effect dominates is empirical and may differ by payload and by hop. This article treats both concerns as hypotheses to test, not as findings.

The central structural trade-off is between collaboration and isolation. Multi-agent systems exist because sharing context and delegating work improves results. Every defense that limits sharing, such as partial histories, schema-constrained messages, or per-agent permissions, also limits what the team can do together. Leaving channels open maximizes capability and propagation. Locking them down limits propagation and may erase the benefit of having several agents. An evaluation should report where a configuration sits on this trade-off, not only whether it passes.

Specialization adds a second tension. Narrow roles support least privilege, since a log reader needs no write access. But narrow roles also mean that each agent depends on others for context, which encourages deference to peers' messages. The more an agent relies on a colleague for information it cannot check itself, the stronger the pressure to trust it.

Escalation has costs as well. An agent that halts and asks a human whenever a peer's message looks suspicious is safer, but may flood operators with false alarms, which shifts the failure to the human reviewer. Per-hop detection and over-refusal rates should be read together, and the operational cost of false alarms weighed against missed attacks.

No environment can reproduce every deployment. Production systems have organizational constraints, many more agents, long-lived memory, and attackers who adapt. The goal is not total simulation. It is to identify which features of the multi-agent task matter for the security question, then test whether behavior survives when irrelevant features change and relevant constraints remain.

12. Defensive architectures and what builders should ask

Mitigations fall into several families, and none is known to be sufficient on its own. The strongest position is defense in depth, with each layer evaluated in the same environment. Guidance from industry bodies lists prompt injection among the leading risk categories for applications built on language models [20], and classifications of adversarial machine learning describe related attack classes [21].

- **Treat every handoff as an external boundary.** Do not assume a peer's message is safe. Wrap inter-agent content explicitly, for example with metadata marking it as output from another agent that may have read untrusted material. Spotlighting-style techniques that delimit or transform untrusted text may help a model distinguish it [16], and training models to follow an instruction hierarchy aims to give operator-level directives priority over text in tool or peer outputs [17]. Both are probabilistic, so they should be measured, not assumed. Labels also decay: after a summary is produced, the label must be reapplied, since the summarizer's output is derived from untrusted content.
- **Limit shared context.** Give each agent only the partial history it needs instead of a global log. Narrower channels give an infection fewer places to travel and make per-hop reasoning tractable. This is the isolation side of the trade-off in Section 11.
- **Constrain message form.** Replace free-text instructions with typed, schema-validated messages where possible, so that a handoff carries defined fields instead of arbitrary directives. An executor that accepts only a fixed set of operations with validated parameters is harder to steer than one that interprets prose.
- **Decentralized guardrails.** Do not rely on a single filter at the front door. Each agent-to-agent call and each tool-server interaction should have independent checks and permission scopes. Tool interfaces such as MCP servers return content authored by third parties and should be treated as untrusted sources

[11].

- **Privilege and data-flow separation.** Architectures that keep untrusted data away from the components that choose privileged actions address the problem by construction instead of by model behavior. One approach separates a planner that sees only trusted instructions from components that process untrusted data [18]. A related pattern uses a quarantined model to handle untrusted content and a privileged model that never reads it directly [19]. These designs may restrict what the system can do, which is the trade-off from Section 11.
- **Least privilege per role.** Give each agent the narrowest authority its role needs, with scoped, short-lived credentials. This follows long-standing principles of protection design [7] and limits the blast radius of any single compromise.
- **Egress control.** Default-deny outbound network access from agent environments, so exfiltration fails even when an agent complies.
- **Human approval for high-impact actions.** Require confirmation for irreversible or security-relevant actions, and show the approver the chain of messages that led to the request, so the decision is informed.
- **Provenance and audit.** Record every handoff with sender, receiver, and source lineage. Provenance records make investigation possible and give detectors something to examine.
- **Circuit breakers and memory hygiene.** Detect loops, runaway delegation, and unexpected fan-out, and halt them. Treat shared memory as an input that requires review, expiry, and the ability to purge entries.

Training environments can also contribute. A multi-agent environment with poisoned handoffs offers a trainable signal for resisting propagation. This requires care. A reward that penalizes compliance can be satisfied by distrusting everything, and reward design therefore needs a positive task objective together with constraints on unacceptable shortcuts [25]. Held-out payload families and handoff formats are essential, since a policy trained on a fixed set may learn the surface form of the payloads instead of the underlying distinction between evidence and instruction.

For teams building or buying multi-agent systems, the questions are specific. Was the system tested with hostile content entering through a peripheral agent, or only through the user prompt? Was propagation measured per hop? What forms do handoffs take, and are they validated? What does shared memory persist, and who can write to it? What privileges does each agent hold, and does a low-privilege agent have a path to a high-privilege one? Do verifiers have negative controls that fail chains that propagate but still complete the task? Were payloads varied, and were any held out? Were adaptive attacks tried? A controlled pilot on a small topology, with documented initial states and evidence of repeated execution, is more informative than a large test count with unclear provenance.

13. Ethics, responsible release, and limitations

Research on propagation payloads is dual-use. Payloads and chain designs that reveal weaknesses can also be reused by attackers, and self-replicating payloads raise particular concern. Several practices reduce the risk while preserving scientific value.

- **Sandbox everything.** Run all experiments in isolated environments with synthetic data, no route to real systems, and no connection to production agent services or shared memory.

- **Do not release working replicators.** Share environment structure, metrics, and verifiers openly. Consider withholding or delaying the most effective self-replicating payload strings, or sharing them only with vetted evaluators.
- **Coordinate disclosure.** If a specific framework, protocol, or product is found vulnerable, notify its maintainers before publishing details.
- **Avoid real-world harm.** Do not plant payloads in public documents, repositories, or services that real agents may read.

The approach has limitations. A small fixed topology cannot represent large agent networks, and propagation may behave differently at scale. Fixed payloads cannot represent the full space of attacks, and adaptive attackers may do better. Results describe the sampled models, frameworks, and scaffolds and may change as these change. Detection of derived payloads is imperfect. Some effects, such as slow degradation of shared memory or subtle bias in a team's conclusions, are hard to detect automatically. Finally, a low measured propagation rate is evidence of containment against the tested attacks, not a guarantee of safety, and should not be presented as one.

14. Where Deaimer fits

Deaimer publicly describes work on custom RL environments, coding-agent evaluations, and independent benchmarks, including resettable task environments, automated graders, reference solutions, adversarial checks, and model-run analysis [28]. Its ColdStart benchmark focuses on autonomous recovery of broken systems while checking safety, durability, and integrity alongside functional recovery [29]. Recovery work increasingly involves several cooperating agents, and logs, files, and tickets are natural entry points for the propagation chains described here, so multi-agent adversarial evaluation is a plausible extension of that setting. These descriptions provide relevant context; they do not establish how any particular multi-agent system performs against propagated injections without a controlled experiment such as the one proposed here.

For teams developing multi-agent or operational agent systems, the useful connection is the ability to commission or independently evaluate tasks around a specific security capability, such as containment of an injection at an agent handoff, instead of relying only on public benchmark coverage. The broader research principle is independent of any vendor. Training environments, public benchmarks, and held-out evaluation have different jobs, and a propagation-focused slice of held-out evaluation is one way to make evidence about multi-agent reliability more credible [22][23][24].

Directions for future work

- **Scaling topology.** Move from small fixed chains to larger networks and study how fan-out, loops, and path length affect spread.
- **Adaptive attackers.** Evaluate payloads iteratively optimized against a specific topology and defense, instead of fixed strings.
- **Cross-organization agents.** Study propagation when agents from different organizations interact through open protocols, where no single party controls every component [12].

- **Multimodal handoffs.** Extend to images, screenshots, and audio passed between agents, where instructions can be embedded outside text [4].
- **Memory attacks.** Measure how long a payload persists in shared and long-term memory and how effectively memory review or expiry removes it.
- **Provenance-preserving messages.** Test whether signed or labelled handoffs that preserve source lineage through summarization reduce propagation.
- **Human oversight.** Study whether human reviewers shown the full chain catch compromises that automated checks miss, and how review fatigue changes the result.
- **Defense transfer.** Test whether a defense validated in one framework or topology holds in another.
- **Standardized reporting.** Develop a shared template covering topology, handoff formats, privileges, payloads, trial counts, uncertainty, and held-out exposure.

Conclusion: contain the infection at the handoff

Multi-agent systems are attractive because agents can specialize and share work. The same properties give a hostile instruction something it lacks in a single-agent setting: a route. An agent that reads untrusted text may be well defended or poorly defended, but what happens next depends on what it says to its colleagues and on whether they believe it.

None of the pieces is sufficient alone. Single-agent injection benchmarks establish that the underlying vulnerability is real but do not describe how it moves between agents. Multi-agent studies show that propagation can occur but do not yet give rates across realistic configurations. Defenses reduce risk but constrain collaboration. A realistic environment with weak verifiers can teach shortcuts, and a low propagation rate against a narrow set of payloads can create false confidence. Reliability comes from combining representative topologies, per-hop instrumentation, objective security verifiers, controlled exposure, repeated measurement, held-out payloads, and separate reporting of task and security outcomes.

For researchers, the next useful experiment is to measure how handoff format, shared memory, privilege gradients, and defenses interact. For builders, it is to treat every handoff as a trust boundary and test containment before granting an executor production privileges. For buyers, it is to ask for evidence that propagation was measured, not assumed. The question is not only whether a team of agents completes its work. It is whether, when one of them is fooled, the others know the difference between a colleague and a log.

References

- [1] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. arXiv:2302.12173 (2023). <https://arxiv.org/abs/2302.12173>
- [2] Perez, F., Ribeiro, I. Ignore Previous Prompt: Attack Techniques For Language Models. arXiv:2211.09527 (2022). <https://arxiv.org/abs/2211.09527>

- [3] Lee, D., Tiwari, M. Prompt Infection: LLM-to-LLM Prompt Injection within Multi-Agent Systems. arXiv:2410.07283 (2024). <https://arxiv.org/abs/2410.07283>
- [4] Gu, X., et al. Agent Smith: A Single Image Can Jailbreak One Million Multimodal LLM Agents Exponentially Fast. ICML 2024. arXiv:2402.08567. <https://arxiv.org/abs/2402.08567>
- [5] Triedman, H., Jha, R., Shmatikov, V. Multi-Agent Systems Execute Arbitrary Malicious Code. arXiv:2503.12188 (2025). <https://arxiv.org/abs/2503.12188>
- [6] Hardy, N. The Confused Deputy (or why capabilities might have been invented). ACM SIGOPS Operating Systems Review 22(4), 36–38 (1988).
- [7] Saltzer, J. H., Schroeder, M. D. The Protection of Information in Computer Systems. Proceedings of the IEEE 63(9), 1278–1308 (1975).
- [8] Wu, Q., et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 (2023). <https://arxiv.org/abs/2308.08155>
- [9] Hong, S., et al. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. ICLR 2024. arXiv:2308.00352. <https://arxiv.org/abs/2308.00352>
- [10] Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., Ghanem, B. CAMEL: Communicative Agents for ‘Mind’ Exploration of Large Language Model Society. NeurIPS 2023. arXiv:2303.17760. <https://arxiv.org/abs/2303.17760>
- [11] Model Context Protocol. Specification and documentation. <https://modelcontextprotocol.io> (add access date)
- [12] Agent2Agent (A2A) Protocol. Specification and documentation. <https://a2a-protocol.org> (add access date)
- [13] Cemri, M., et al. Why Do Multi-Agent LLM Systems Fail? arXiv:2503.13657 (2025). <https://arxiv.org/abs/2503.13657>
- [14] Zhan, Q., Liang, Z., Ying, Z., Kang, D. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. arXiv:2403.02691 (2024). <https://arxiv.org/abs/2403.02691>
- [15] DeBenedetti, E., et al. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. NeurIPS 2024 Datasets and Benchmarks. arXiv:2406.13352. <https://arxiv.org/abs/2406.13352>
- [16] Hines, K., et al. Defending Against Indirect Prompt Injection Attacks With Spotlighting. arXiv:2403.14720 (2024). <https://arxiv.org/abs/2403.14720>
- [17] Wallace, E., et al. The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions. arXiv:2404.13208 (2024). <https://arxiv.org/abs/2404.13208>
- [18] DeBenedetti, E., et al. Defeating Prompt Injections by Design. arXiv:2503.18813 (2025). <https://arxiv.org/abs/2503.18813>
- [19] Willison, S. The Dual LLM pattern for building AI assistants that can resist prompt injection. simonwillison.net (2023). <https://simonwillison.net/2023/Apr/25/dual-llm-pattern/>

- [20] OWASP Foundation. OWASP Top 10 for Large Language Model Applications: LLM01 Prompt Injection. <https://genai.owasp.org>
- [21] Vassilev, A., Oprea, A., Fordyce, A., Anderson, H. Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations. NIST AI 100-2. National Institute of Standards and Technology. <https://csrc.nist.gov/pubs/ai/100/2/e2023/final> (check for the latest edition)
- [22] Liu, X., et al. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688 (2023; revised 2025). <https://arxiv.org/abs/2308.03688>
- [23] Xie, T., et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. NeurIPS 2024. <https://arxiv.org/abs/2404.07972>
- [24] Jimenez, C. E., et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR 2024. arXiv:2310.06770. <https://arxiv.org/abs/2310.06770>
- [25] Skalse, J., Howe, N., Krashennnikov, D., Krueger, D. Defining and Characterizing Reward Hacking. NeurIPS 2022. arXiv:2209.13085. <https://arxiv.org/abs/2209.13085>
- [26] Brown, L. D., Cai, T. T., DasGupta, A. Interval Estimation for a Binomial Proportion. Statistical Science 16(2), 101–133 (2001).
- [27] [Author Name]. Poisoned Logs and Broken Trust: Why Autonomous Recovery Agents Need Adversarial Telemetry Evaluation. Deaimer Research (companion article).
- [28] Deaimer. RL Environments & Coding-Agent Evaluations. <https://www.deaimer.com/rl-environments> (add access date)
- [29] Deaimer. ColdStart — Audited AI Agent Recovery Benchmark. <https://www.deaimer.com/benchmarks/coldstart> (add access date)