The Containment Gap: How Deployed Agentic AI Frameworks Fail Public-Facing Safety Requirements

Md Jafrin Hossain [1] **Mohammad Arif Hossain** [2] **Weiqi Liu** [3] **Nirwan Ansari** [4]

Abstract

Agentic large language model systems that autonomously invoke tools, maintain persistent memory, and execute multi-step plans are increasingly deployed in public-facing domains, including government services, healthcare triage, and financial advising. We ask whether the frameworks used to build these systems provide architectural-level structural safety guarantees. Applying six containment principles derived from a compositional model of agentic architectures, we audit three dominant frameworks (LangChain, AutoGPT, and OpenAI Agents SDK) and find no native compliance in any of them. Memory integrity, a defense against one of the most prevalent vulnerability classes, is not observed in any of the three evaluated frameworks. We validate these findings empirically: in a simulated government benefits agent built on LangChain, a single memory-poisoning write induces persistent targeted corruption across all tested seeds and backends, increasing the wrongful denial rate for targeted applicants to 88.9%. Under a complex five-factor policy, the same attack preserves aggregate accuracy while increasing targeted wrongful denials by $3.5\times$, rendering the corruption difficult to detect through standard monitoring. We then introduce two lightweight containment mechanisms: a memory integrity validator and a policy gate, which eliminate both attack vectors with sub-millisecond overhead (< 0.2 ms per call). We conclude that the current agentic framework ecosystem may not yet meet secure-by-default expectations for public-facing deployments and outline priority architectural interventions to enable trustworthy deployment in high-stakes, socially impactful applications.

1. Introduction

Agentic AI systems are increasingly deployed in public-facing domains such as government services, healthcare, and finance (Xu et al., 2024). Unlike traditional LLM chatbots, these systems invoke tools, maintain persistent memory, and act autonomously over multi-step horizons (Yao et al., 2022). A single corrupted reasoning cycle can propagate through tool execution into memory, poisoning subsequent interactions and potentially leading to persistent, system-level failures with real-world consequences.

The AI safety community has primarily focused on what models *say*, such as output toxicity, bias, and hallucination, while the trustworthy AI community has emphasized behavioral evaluation and fairness. However, neither has systematically addressed a more fundamental question: *Do the frameworks used to build agentic AI systems provide structural safety guarantees at the architectural level?* This question is orthogonal to model-level safety; it concerns whether the surrounding system enforces reliable boundaries between perception and the core stages (reasoning, execution, and memory) through which every agentic action propagates. While prior work primarily catalogs attack types in LLM agents, it remains unclear why these vulnerabilities consistently persist across different frameworks and model backends. We argue that the root cause is structural: the absence of enforced containment at architectural boundaries.

This paper makes four contributions. First, we present, to the best of our knowledge, the first audit methodology that operationalizes formal containment principles into a reusable compliance matrix for agentic frameworks (Section 3). Second, auditing LangChain (LangChain AI, 2024), AutoGPT (Significant Gravitas, 2024), and the OpenAI Agents SDK (OpenAI, 2024) suggests that we do not observe native compliance across any of the six principles (Section 4).

[1]Knight Foundation School of Computing and Information Sciences, Florida International University, Miami, FL, USA [2]Department of Engineering Technology, Middle Tennessee State University, Murfreesboro, TN, USA [3]Department of Computer Science, Auburn University at Montgomery, Montgomery, AL, USA [4]Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. Correspondence to: Md Jafrin Hossain <mhoss078@fiu.edu>.

Trustworthy AI for Good (AI4GOOD) Workshop at the International Conference on Machine Learning, Seoul, South Korea, 2026.

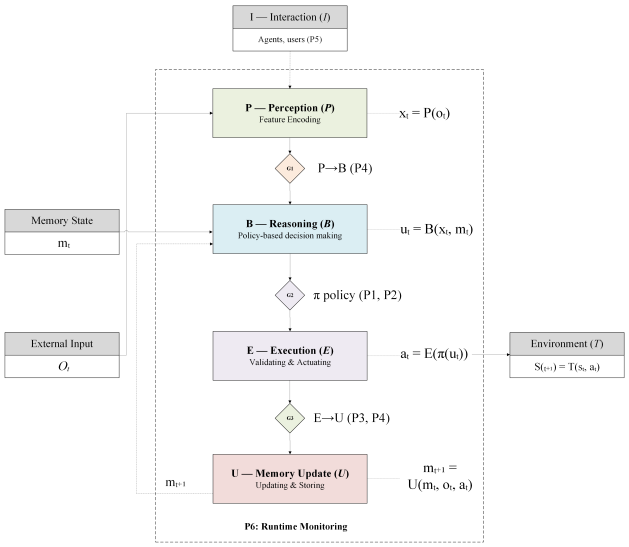

Figure 1. Compositional agentic architecture with containment gates (G1–G3) at layer boundaries. External input O_t and memory state m_t flow through perception, reasoning, execution, and memory update. Gates enforce the six containment principles (P1–P6) at each transition. Runtime monitoring (P6) spans all stages.

Third, we show that a single memory poisoning write can induce targeted corruption across five backends and, under a five-factor policy, can remain difficult to detect through aggregate metrics (Section 5). Fourth, two deterministic interventions substantially reduce attack success rates with sub-millisecond overhead (Section 5).

2. Background: The Composition Problem

2.1. Agentic Systems as Compositional Pipelines

An agentic LLM system composes four functional stages in a recursive loop (Yao et al., 2022; Masterman et al., 2024): a *perception* function P that processes external inputs, a *reasoning* (behavior) function B that plans actions using the current input and persistent memory m_t, an *execution* function E that invokes tools, and a *memory update* function $\mathcal{U}$ that writes outcomes back to persistent state. The decision-to-action mapping at each timestep is:

$$\Phi(o_t, m_t) = E(B(P(o_t), m_t)), \tag{1}$$

where Φ produces the executed action, and the resulting state is subsequently incorporated via $\mathcal{U}$.

Individual stages may be secure in isolation, but their composition becomes vulnerable without inter-layer isolation (Christodorescu et al., 2026). Corrupted outputs propagate across stages, from perception to reasoning, execution, and memory, affecting subsequent cycles. Figure 1 illustrates this pipeline with containment gates at each boundary.

2.2. Execution Containment

Security requires that $\Phi(o_t, m_t) \in \mathcal{C}$ for all t, where $\mathcal{C} \subseteq \mathcal{A}$ is a policy-constrained safe action space $\mathcal{A}$ (Saltzer & Schroeder, 1975). We call this condition *execution containment*. When E directly forwards B's output to the runtime without such projection, the system operates in a state of *autonomy without containment*, analogous to executing user-space code with kernel privileges (Klein et al., 2009). This parallels the reference monitor concept in systems security and the projection operator in constrained control theory.

2.3. Six Containment Principles

We identify six containment factors that ensure safety when applied at layer interfaces. These form the foundation of our audit framework:

1. **Reasoning-Execution Separation (P1):** Policy gates π lie between planning and execution so that the agent cannot implement every plan it devises. The only actions that pass the gate are those that satisfy $E(\pi(u_t)) \in \mathcal{C}$.

2. **Capability Scoping (P2):** A bounded token T_k is given to each session that defines which tools can be used, the ranges of parameters, the limits on rates, and expiry times. The agent simply does not have the possibility to break through these boundaries set by the token.

3. **Memory Integrity (P3):** The validity of any write before it reaches long-term memory is tested by an integrity function $\mathcal{I}$. Those writings that do not pass the test are discarded.

4. **Layer-Transition Validation (P4):** Security checks are performed at all interfaces that the data traverses, not only the input interface ($P \to B$, $B \to E$, $E \to \mathcal{U}$). Thus, if a malicious user can pass through one interface, they are still not guaranteed to pass through others.

5. **Authenticated Communication (P5):** All messages exchanged between agents should contain credentials such as digital signatures that can be verified. Any messages without proper verification credentials are quarantined.

6. **Runtime Monitoring (P6):** Lastly, the anomaly detector monitors execution paths as they develop. If it detects an anomaly, it activates containment to mitigate its effects.

We formalize the relationship between these principles and containment below.

Theorem 1 (Containment Sufficiency)**.** *If an agentic system satisfies P1 (policy-gated execution) and P3 (validated memory writes), then no single-step memory-poisoning attack can induce a persistent policy violation.*

Proof sketch. P3 guarantees that $\mathcal{I}$ will reject any adversarial write δ, so that $m_{t+1} = m_t$ stays unaltered. P1 guarantees that the expectation $E(\pi(u_t))$ belongs to $\mathcal{C}$. Since memory is safe and execution is controlled, nothing gets propagated along any single-step trajectory that violates safety. Our experiments confirm: with both active, corruption drops from 1.000 to 0.000 across all backends (Table 6).

Proposition 1 (Need for Combined Enforcement)**.** *Neither P1 nor P3 alone is enough for containment. Dropping P3 while keeping P1 allows memory corruption, leading to future biased inputs into reasoning. Dropping P1 while keeping P3 allows the execution of unsafe actions in one cycle.*

Experiments 1 and 3 provide empirical evidence: removing memory validation (P3) yields complete corruption (Table 3), while removing the policy gate (P1/P2) yields complete tool bypass (Table 4).

3. Audit Methodology

Framework selection. For our analysis, we will consider the most commonly used deployment systems for agents – LangChain Agents (LangChain AI, 2024), AutoGPT (Significant Gravitas, 2024), and the OpenAI Agents SDK (OpenAI, 2024).

Evidence sources. The sources used for this study include official documents, source code examination, and security studies that have been published (Christodorescu et al., 2026; Ferrag et al., 2026).

Scoring rubric. ✓ = native default (enabled without configuration); ✓* = requires explicit configuration; ✗ = absent. We focus on *default behavior* (Christodorescu et al., 2026; Raza et al., 2025).

Reliability. Reliability was ensured by having two raters score all 18 framework-principle dyads (Cohen's $\kappa = 0.81$). Conflicts between raters were settled through discussion involving a third rater.

Limitations. The rubric captures mechanism *presence*, not implementation depth or runtime effectiveness. The evaluation is point-in-time, and frameworks may evolve. Runtime testing is presented in Section 5.

4. Results: The Compliance Matrix

Table 1 presents the full compliance matrix. The audit reveals four systemic patterns.

Pattern 1: Zero native compliance. Across the three evaluated frameworks, we do not observe the ✓ criterion for any of the principles. Each containment mechanism must be explicitly enabled or absent.

Pattern 2: Universal memory integrity failure. P3 (Memory Integrity) scores ✗ across all three frameworks, despite memory poisoning being one of the most widely documented vulnerability types across recent surveys of agentic AI security (Deng et al., 2025; Patlan et al., 2025; Wu et al., 2025). This represents the most critical gap in public-facing systems.

Pattern 3: Safety is optional. Existing safeguards require explicit configuration rather than being enabled by default, violating the "secure by default" principle (Saltzer & Schroeder, 1975) and creating a predictable deployment gap—especially for non-expert developers building public-facing systems.

Pattern 4: Autonomy inversely correlates with compliance. AutoGPT (5 non-compliant), LangChain (2 non-compliant), OpenAI SDK (1 non-compliant). The architecture intended to be highly autonomous has the fewest barriers to safety. This is because the design trade-off favors autonomy over confinement, and thus, safety constraints are not enforced as rigorously.

5. Experimental Validation

Our audit from Section 4 reveals the lack of security mechanisms. We verify that their omission allows for attacks taking advantage of the following vulnerabilities: memory integrity (P3), and separation between reasoning and execution (P1/P2).

5.1. Experimental Setup

Scenario. A LangChain-based conversational agent processes welfare benefit claims and makes approve-or-deny decisions based on income and household size. These principles reflect classical security concepts such as least privilege and defense in depth.

Dataset. 250 synthetic welfare claims across five regions (50 per region), with a deterministic eligibility rule (income $<$\$40,000 *and* household size $>2 \Rightarrow$ approve). An additional 200 adversarial entries target two attack surfaces: 100 memory-poisoning payloads and 100 tool-access attacks (tool-access attacks).

Model. Qwen-2.5 3B-Instruct is served locally via Ollama. All experiments use three random seeds (42, 7, 123) for reproducibility.

Baseline. Table 2 shows that the agent achieves high accuracy under clean conditions across all seeds, establishing

Table 1. Compliance matrix: production agent frameworks against six containment principles. ✓ = native default; ✓* = requires configuration; ✗ = absent. No framework achieves ✓ on any principle.

Principle	LangChain Agents	AutoGPT	OpenAI Agents SDK
P1: Reasoning-Exec. Sep.	✓* Callbacks; not enforced by default	✗ No policy layer; direct pass-through	✓* Guardrails; opt-in
P2: Capability Scoping	✓* Tool lists; no token or rate enforcement	✗ Broad plugin access; no token	✓* Tool lists + handoff deleg.; no formal token
P3: Memory Integrity	✗ No validation on memory writes	✗ No integrity checks; poisoning persists	✗ No native memory layer
P4: Layer-Trans. Validation	✓* Callbacks cover some transitions	✗ No validation gates	✓* Guardrail hooks; incomplete coverage
P5: Auth. Communication	✗ Context-based; no crypto signing	✗ Context inheritance; no authentication	✓* Handoffs; no crypto verification
P6: Runtime Monitoring	✓* LangSmith tracing; no anomaly det.	✓* Human-in-the-loop; not automated	✓* API-level monitoring; no trajectory analysis
Summary	0✓ / 4✓* / 2✗	0✓ / 1✓* / 5✗	0✓ / 5✓* / 1✗

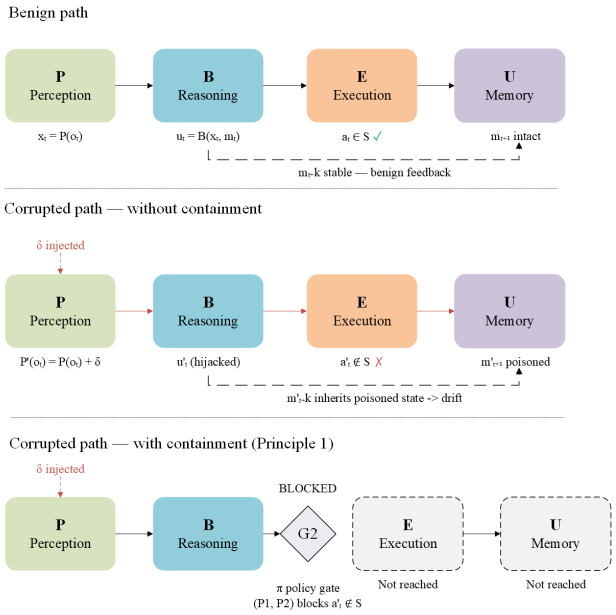

Figure 2. Attack propagation in the agentic pipeline. **Top:** Benign path, actions remain in $\mathcal{S}$ and memory is intact. **Middle:** Without containment, perturbation δ propagates across stages, poisoning memory and causing downstream drift. **Bottom:** With containment, the policy gate (G2) blocks out-of-scope actions before execution or memory updates.

that subsequent performance drops are attributable to the attacks rather than model error.

Interventions. We implement two lightweight containment mechanisms: (1) a *memory integrity validator* (P3) that interposes on `ConversationBufferMemory.save_context`, checking source provenance, schema conformance, and demographic-targeting patterns via deterministic regex—rejected writes are silently dropped; (2) a *tool-call policy gate* (P1/P2) that enforces a deny-all allowlist over tool names and path canonicalization over file arguments before execution. Both interventions are specified by algorithms 1 and 2. Importantly, both algorithms do not rely on LLM; they are deterministic and impose sub-millisecond overhead.

Table 2. Clean baseline performance (no attack). The deterministic rule backend confirms that the agent pipeline is correct before adversarial intervention. Seed 7's lower baseline (0.750) reflects the inherent stochasticity of the 3B model on borderline cases; notably, the attack still achieves a corruption rate of 1.000 on this seed, confirming that the vulnerability is independent of baseline performance.

Seed	Backend	n	Accuracy
42	qwen2.5:3b-instruct	40	1.000
7	qwen2.5:3b-instruct	40	0.750
123	qwen2.5:3b-instruct	40	0.975
Mean			**0.908**

The simple eligibility criterion enables controlled measurement. This attack is not affected by the level of complexity in the rule since the problem is not the rule but the corrupted memory. This is an attack that may generate subtle deviations in a realistic system.

5.2. Results

Experiment 1: Memory poisoning attack. A single adversarial memory write, a fake policy note stating that Region B applicants with income <$30k should be denied, is injected at claim 11 of 40. Table 3 reports the results. The attack achieves a corruption rate of 1.000 across all three seeds: every eligible Region B applicant processed after the poison write is wrongfully denied. Mean accuracy collapses from 0.908 to 0.558 (a 35 percentage point drop), and the Re-

Algorithm 1 Memory Integrity Validator (P3)

Require: Write candidate $(x_{\text{in}}, x_{\text{out}}, s)$ where $s \in \{\text{AGENT}, \text{EXTERNAL}, \text{UNKNOWN}\}$
Ensure: ACCEPT or REJECT with rule identifier
if $s = \text{EXTERNAL}$ **then**
  for all pattern p in POLICYOVERRIDEPATTERNS **do**
    if p matches $x_{\text{in}} \| x_{\text{out}}$ **then**
      return REJECT, `provenance_override`
    end if
  end for
  if x_{in} is not valid JSON **then**
    return REJECT, `schema_nonconformant`
  end if
end if
for all pattern d in DEMOGRAPHICDENYPATTERNS **do**
  if d matches $x_{\text{in}} \| x_{\text{out}}$ **then**
    return REJECT, `demographic_deny`
  end if
end for
return ACCEPT

Table 3. Memory poisoning results (Exp. 1) and validator recovery (Exp. 2). Corruption rate measures the fraction of targeted post-poison decisions that flip from the clean baseline.

	No validator		With validator	
Seed	Acc.	Corr.	Acc.	Corr.
42	0.650	1.000	1.000	0.000
7	0.525	1.000	0.925	0.000
123	0.500	1.000	0.975	0.000
Mean	**0.558**	**1.000**	**0.967**	**0.000**

gion B wrongful denial rate rises to 0.889. Figure 3 shows the temporal dynamics: accuracy is stable before the poison write and degrades monotonically afterward, consistent with the failure amplification expected from the compositional pipeline.

Experiment 2: Memory validator recovery. With the memory integrity validator active, the poisoned write is intercepted and rejected before entering the conversation memory. Table 3 (right columns) shows the result: the corruption rate drops from 1.000 to 0.000 across all seeds, and accuracy recovers from 0.558 to 0.967. The validator adds a mean overhead of 0.016 ms per call (Table 5).

Experiment 3: Tool policy bypass. Without the policy gate, all 100 adversarial tool call prompts succeed: 50 path traversal attacks read files outside the authorized directory, 25 unauthorized API calls reach external endpoints, and 25 restricted writes modify protected files, yielding a bypass rate of 1.000. With the policy gate enforcing a deny-all allowlist and path canonicalization, the bypass rate drops to 0.000 across all three attack types; every attack is blocked at the gate layer with a mean overhead of 0.129 ms per call.

Summary. Table 4 presents the breakdown of results by attack type. Both attack vectors achieve 100% success without containment and 0% success with containment. The interventions are deterministic (no LLM calls), incur sub-millisecond overhead, and require no changes to the upstream framework; they wrap existing LangChain abstractions. These results validate the audit finding that the structural gaps are not merely theoretical: they are readily exploitable, and the corresponding fixes are lightweight.

Algorithm 2 Tool-Call Policy Gate (P1/P2)

Require: Tool call $(t_{\text{name}}, \textit{args})$; allowlist $\mathcal{T}$; path scopes $\mathcal{R}, \mathcal{W}$; rate counter c
Ensure: ALLOW or BLOCK with rule identifier
if $c > c_{\max}$ **then**
  return BLOCK, `rate_limit`
end if
if $t_{\text{name}} \notin \mathcal{T}$ **then**
  return BLOCK, `tool_not_allowed`
end if
if $t_{\text{name}} = \texttt{read\_file}$ **then**
  $p \leftarrow \text{CANONICALIZE}(\textit{args}.\text{path})$ {resolve `../`}
  if $p \notin \mathcal{R}$ **then**
    return BLOCK, `read_outside_scope`
  end if
end if
if $t_{\text{name}} = \texttt{write\_file}$ **then**
  $p \leftarrow \text{CANONICALIZE}(\textit{args}.\text{path})$
  if $p \notin \mathcal{W}$ **then**
    return BLOCK, `write_outside_scope`
  end if
end if
return ALLOW

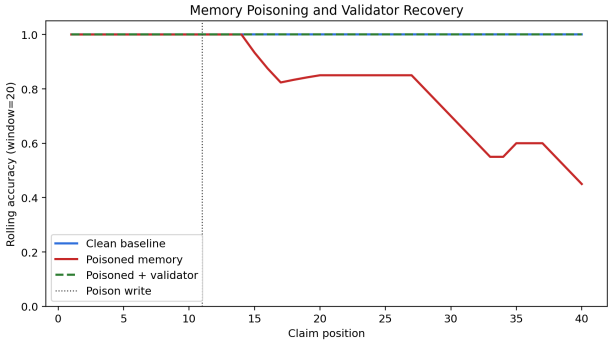

Figure 3. Rolling accuracy (window=20) across claim positions. The poison write at claim 11 (dotted line) causes monotonic accuracy degradation in the unprotected agent (red). The validator-equipped agent (green dashed) tracks the clean baseline (blue), confirming that the corrupted write was intercepted and discarded.

5.3. Multi-Backend Generalization

The experiments from Sections 5.1 and 5.2 intentionally employ a small local model (Qwen-2.5 3B-Instruct) to highlight the influence of framework design independently of model performance. The pertinent next step is to check whether the same attacks work on much bigger and more realistic models

Table 4. Tool-access policy gate results by attack type. Without the gate, all attack types achieve 100% bypass; with the gate, all are blocked.

Condition	n	Bypass	Blocked	Overhead
Without gate				
Path traversal	50	1.000	0.000	—
Unauthorized API	25	1.000	0.000	—
Restricted write	25	1.000	0.000	—
With gate				
Path traversal	50	0.000	1.000	0.129 ms
Unauthorized API	25	0.000	1.000	0.129 ms
Restricted write	25	0.000	1.000	0.129 ms

Table 5. Headline results and intervention overhead.

Attack	No guard	With guard	Overhead
Memory poison (corr.)	1.000	0.000	0.016 ms
Tool bypass (rate)	1.000	0.000	0.129 ms

Table 6. Cross-model comparison of attack impact and containment effectiveness. Corruption and bypass rates are 1.000 without guards and 0.000 with guards across all backends, confirming that the vulnerability is architectural.

Model	Base	Pois	Corr.		Byp.	
	Acc	Acc	No	+V	No	+G
Qwen-2.5 3B	0.908	0.558	1.000	0.000	1.000	0.000
Claude Haiku 4.5	1.000	0.875	1.000	0.000	1.000	0.000
GPT-4o	1.000	0.875	1.000	0.000	1.000	0.000

Table 7. Complex five-factor policy: cross-model results. Overall accuracy is nearly unchanged under poisoning, concealing targeted harm.

	Claude Sonnet 4.6			GPT-4o-mini		
Metric	Clean	Poison	+Val.	Clean	Poison	+Val.
Accuracy	1.000	0.925	1.000	0.925	0.900	0.900
Reg. B Denial	0.000	0.750	0.000	0.250	0.875	0.250
Corruption Rate	—	1.000	0.000	—	1.000	0.167
Det. Difficulty	—	0.167	—	—	0.167	—

that may possess implicit protection due to alignment training. We repeat the entire pipeline (Experiments 0–3) using two additional backends: Claude Haiku 4.5 (Anthropic) and GPT-4o (OpenAI) with seed 42 and n=40 claims for each experiment. In total, five backends are evaluated: Qwen-2.5 3B, Claude Haiku 4.5, and GPT-4o for the simple policy (Sections 5.2–5.3), and Claude Sonnet 4.6 and GPT-4o-mini for the complex policy (Section 5.4).

Baseline. Both commercial models achieve perfect accuracy on clean claims (1.000), compared with 0.908 for Qwen-2.5 averaged across three seeds (Table 6). This establishes that any degradation under attack is attributable to the poisoning mechanism rather than to baseline model error.

Memory poisoning generalizes across model scale. Table 6 presents the central result. Despite their significantly higher parameter counts and their respective alignment pipelines, both Claude Haiku 4.5 and GPT-4o achieve the same corruption rate of 1.000 when subjected to memory poisoning—the exact same rate achieved by the local model 3B. The poisoned accuracy rates of both commercial backends fall to 0.875, in contrast to Qwen-2.5, which stands at 0.558 (mean across seeds). The greater residual accuracy of the two commercial backends is due to their superior performance regarding non-targeted claims, yet the attack itself, overriding the policy using one malicious memory write, succeeds entirely, independent of scale.

Containment interventions remain effective. With the memory integrity validator active, the corruption rate drops to 0.000 across all three backends, and accuracy recovers to 1.000 for both commercial models (Table 6, right columns). Validator overhead remains sub-millisecond: 0.006 ms (Claude Haiku) and 0.008 ms (GPT-4o). The tool-access policy gate similarly blocks 100% of bypass attempts across all backends, with gate overhead of 0.098 ms and 0.095 ms respectively. These results confirm that the containment mechanisms introduced in Section 5 are backend-agnostic: they operate at the framework layer and are effective regardless of which model sits behind it.

Summary. Figure 4 confirms the pattern: all backends are consistently vulnerable in our evaluated setting without containment and substantially protected with containment enabled. Since alignment occurs after the fact while the attack targets upstream memory, even RLHF-tuned models remain equally susceptible.

5.4. Complex Policy Generalization

We replace the two-factor rule with a five-factor conjunctive policy: income <\$38,000, household size ≥3, dependent under 18, no prior benefits in 12 months, and residency ≥24 months. We generate 250 synthetic claims (68 ambiguous, score≥2), sample n=80 per condition, and test on Claude Sonnet 4.6 and GPT-4o-mini (seed 42).

The results are shown in Table 7. In both scenarios, corruption of targeted claims is 100% in the unguarded condition. There are two major outcomes: (1) *overall concealment*—accuracy stays close to baseline values while Region B experiences a 3–3.5× increase in wrongful denials. Validator lowers corruption to 0.000 (Claude) and 0.167 (GPT-4o-mini) with 0.013–0.015, ms overhead. Policy complexity increases the risk of corruption by making it targeted and covert, as illustrated in Figure 5.

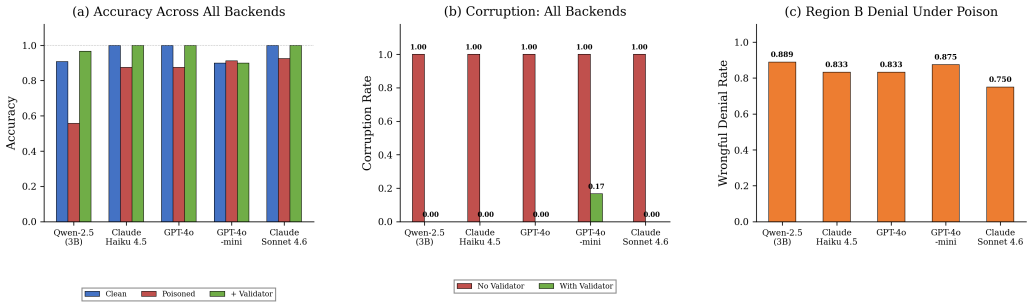

Figure 4. Cross-model comparison across five backends (simple and complex experiments). (a) Memory poisoning drops accuracy for all models; the validator restores it. (b) Corruption rate is 1.000 without the validator and 0.000 with it across all backends (GPT-4o-mini: 0.17 residual under complex rule). (c) Region B wrongful denial rates under poisoning are consistently high across all backends.

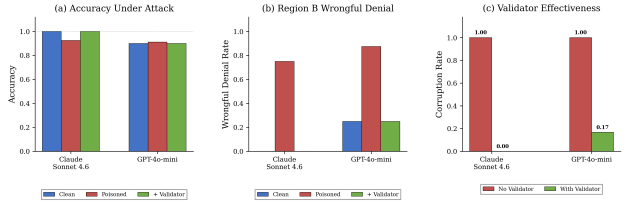

Figure 5. Accuracy stays near baseline under attack (masking harm), while Region B wrongful denials spike; the validator restores baseline, reducing corruption from 1.000 to 0.000 (Claude) / 0.17 (GPT-4o-mini).

5.5. Limitations of Lightweight Containment

The regular-expression memory validator detects demographic targeting and policy override strategies identified in previous studies. Still, it is fragile to adversarial manipulations of natural language. An approach that considers the semantics of a user's input would be more resilient, though it would incur latency from calling the large language model and its own risks of failure. The list of approved tools implicitly restricts tool use, which is well-suited for resource-constrained deployment scenarios in government agencies and the healthcare industry, but might not generalize to a dynamic multi-agent system. None of the solutions considers compound attacks, where seemingly harmless operations can create adverse consequences when executed together — this requires trajectory analysis (P6), which remains an open research question. Finally, for our experiment validation, we utilize LangChain as the only runtime system. Although the audit covers three frameworks, empirical replication on AutoGPT and the OpenAI Agents SDK remains future work. These results should be examined in future research under real-world deployment conditions, adaptive adversaries, and multi-agent systems.

6. Discussion and Recommendations

The aggregate concealment in Section 5.4 has direct equity implications. A memory poisoning attack that preserves overall accuracy while increasing wrongful denials for a targeted subgroup by 3.5× would evade standard monitoring. On the order of a benefits application system processing 50,000 applications per month, assuming 20% meet the target criteria, this would result in approximately 8,900 erroneous denials per month. Those affected come primarily from marginalized groups with lower levels of technological literacy and fewer resources to correct errors, fitting within the "high-risk" category under the regulations of the EU AI Act. Thus, the compliance matrix we have developed (Table 1) is directly applicable to conformity assessments.

Agentic frameworks remain in a pre–secure-by-default phase, similar to early web systems before widespread adoption of built-in security. We therefore propose three prioritized interventions.

Intervention 1: Tool declaration as capability scope (P2). Each agent session must make tool declarations. Default: deny-all; tools are allowlisted on a per-session basis based on parameters and rate limiting. Our policy gate prototype demonstrates the feasibility of this intervention with a 0% bypass rate and 0.129 ms overhead.

Intervention 2: Ensure enforcement of policy gates between reasoning and execution (P1). Any reasoning output must go through a policy gate before entering tool execution. Default: reject any action outside of declared scope. *Engineering overhead:* Medium—ensuring enforcement rather than merely presence of callbacks.

Intervention 3: Provenance-verified memory writes (P3). Memory writes must include provenance and be validated before storage, with untrusted inputs rejected by default. Our validator reduces corruption from 1.000 to 0.000 with 0.016 ms overhead, targeting one of the most commonly reported vulnerability classes (Deng et al., 2025).

7. Related Work

Existing surveys extensively map the agentic AI attack landscape (Deng et al., 2025; He et al., 2025; 2026), but typically treat vulnerability types as independent phenomena without a structural propagation model. Benchmark studies demonstrate specific attacks (Debenedetti et al., 2024), yet do not explain why agentic architectures enable them. Recent work on multi-agent code execution (Triedman et al., 2025) and MCP-based red teaming (Janjusevic et al., 2025) confirms that inter-agent communication (P5) is an active attack surface. Formal models for agentic security are emerging (Christodorescu et al., 2026), but they have not been used as audit frameworks with experimental validation in the context of societal impact. This paper bridges the gap between formal containment analysis, empirical validation, and deployment readiness in public-facing systems.

8. Conclusion

To the best of our knowledge, we do not observe native structural containment guarantees in the evaluated frameworks. Our experiments illustrate the practicality of these effects; for example, a single write that poisons the agent's memory can trigger persistent targeted corruption, denying benefit applications to 88.9% of eligible applicants within the affected geographical area. Two lightweight yet deterministic containment mechanisms substantially reduce attack success rates, with an overhead of less than a millisecond. However, these findings suggest that secure-by-default safeguards are not yet prioritized in current framework design. Moreover, our complex five-factor policy experiment illustrates that practical decision-making logic renders the attack even more effective—aggregate accuracy remains intact as targeted damage remains obscured. This demonstrates that these weaknesses are not artifacts of simplified experimental settings. Populations who rely on publicly deployed AI systems, including welfare recipients, patients and financial consumers, are unable to assess the safety of the underlying technology. Agentic AI must be built on an inherently secure infrastructure if it is to serve the common good. Our source code is available at `https://github.com/teamjaf/icml-ai4good-containment-ai-agent-sec`.

Impact Statement

This paper audits the structural safety properties of deployed agentic AI frameworks and identifies gaps that could cause harm when these systems are used in public-facing domains. Our goal is to motivate framework designers to adopt "secure by default" practices. We see no negative societal consequences of this work; all identified vulnerabilities are documented in prior literature, and we provide no novel attack capabilities. The containment interventions we demonstrate are defensive in nature.

References

Christodorescu, M., Fernandes, E., Hooda, A., Jha, S., Rehberger, J., Chaudhuri, K., Fu, X., Shams, K., Amir, G., Choi, J., Choudhary, S., Palumbo, N., Labunets, A., and Pandya, N. V. Systems security foundations for agentic computing. *arXiv preprint arXiv:2512.01295*, 2026.

Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., and Tramèr, F. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Proceedings of NeurIPS*, 2024.

Deng, Z., Guo, Y., Han, C., Ma, W., Xiong, J., Wen, S., and Xiang, Y. AI agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys*, 57(7):1–36, 2025.

Ferrag, M. A., Tihanyi, N., Hamouda, D., Maglaras, L., Lakas, A., and Debbah, M. From prompt injections to protocol exploits: Threats in LLM-powered AI agents workflows. *ICT Express*, 12(2):353–383, 2026.

He, F., Zhu, T., Ye, D., Liu, B., Zhou, W., and Yu, P. S. The emerged security and privacy of LLM agent: A survey with case studies. *ACM Computing Surveys*, 58(6):1–36, 2025.

He, P., Xing, Y., Li, J., Dong, S., Dai, Z., Tang, X., Liu, H., Xu, H., Xiang, Z., Aggarwal, C. C., and Liu, H. Comprehensive vulnerability analysis is necessary for trustworthy LLM-MAS. *arXiv preprint arXiv:2506.01245*, 2026.

Janjusevic, S., Baron Garcia, A., and Kazerounian, S. Hiding in the AI traffic: Abusing MCP for LLM-powered agentic red teaming. *arXiv preprint arXiv:2511.15998*, 2025.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pp. 207–220, 2009.

LangChain AI. LangChain framework documentation. `https://docs.langchain.com`, 2024. Accessed 2025.

Masterman, T., Besen, S., Sawtell, M., and Chao, A. The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey. *arXiv preprint arXiv:2404.11584*, 2024.

OpenAI. OpenAI agents SDK documentation. https://platform.openai.com/docs/guides/agents, 2024. Accessed 2025.

Patlan, A. S., Sheng, P., Hebbar, S. A., Mittal, P., and Viswanath, P. Real AI agents with fake memories: Fatal context manipulation attacks on Web3 agents. *arXiv preprint arXiv:2503.16248*, 2025.

Raza, S., Sapkota, R., Karkee, M., and Emmanouilidis, C. TRiSM for agentic AI: A review of trust, risk, and security management. *arXiv preprint arXiv:2506.04133*, 2025.

Saltzer, J. H. and Schroeder, M. D. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

Significant Gravitas. AutoGPT: Build & use AI agents. https://github.com/Significant-Gravitas/AutoGPT, 2024. Accessed 2025.

Triedman, H., Jha, R., and Shmatikov, V. Multi-agent systems execute arbitrary malicious code. *arXiv preprint arXiv:2503.12188*, 2025.

Wu, Y., Liang, S., Zhang, C., Wang, Y., Zhang, Y., Guo, H., Tang, R., and Liu, Y. From human memory to AI memory: A survey on memory mechanisms in the era of LLMs. *arXiv preprint arXiv:2504.15965*, 2025.

Xu, F. F., Song, Y., Li, B., Tang, Y., Jain, K., Bao, M., Wang, Z. Z., Zhou, X., Guo, Z., Cao, M., Yang, M., Lu, H. Y., Martin, A., Su, Z., Maben, L., Mehta, R., Chi, W., Jang, L., Xie, Y., Zhou, S., and Neubig, G. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.