This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record will appear in the SHIELD-AI 2026 workshop proceedings (Springer CCIS) and is available online at: http://dx.doi.org/[DOI to be added on publication]. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use
https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms

When the Analyst Scans for the Attacker: Subverting Agentic LLM Security Operations in OT Networks

Jaafer Rahmani[1,2*], Eren Çil[1], and Axel Sikora[1]

[1] ivESK, Offenburg University of Applied Sciences, Offenburg, Germany
jaafer.rahmani@hs-offenburg.de, cil.eren@outlook.com,
axel.sikora@hs-offenburg.de
[2] Faculty of Engineering, University of Freiburg, Freiburg, Germany
jaafer.rahmani@email.uni-freiburg.de

Abstract. Security Operations Centers increasingly deploy large language model (LLM) assistants to triage alerts. Such an assistant receives tools that act on the network and a memory of monitored data. This creates an attack surface because the assistant reads data partly written by the adversary it is meant to detect. We study this surface on CoAnalyst, an agentic LLM analyst for operational technology (OT) alert triage that we introduced in prior work. That earlier paper described what CoAnalyst does for the analyst; here we ask the opposite question: what can an adversary make it do? Unlike prior indirect-injection studies on general-purpose agents, the tools here act on real OT assets through a real alert schema. We find three vulnerabilities in how the agent selects a scan target, retrieves alerts, and summarizes them. Three narrow, typed interfaces already gate two attacks, although none was added for security. Widening one interface for analyst utility re-opens the surface and lets a crafted alert steer the agent into scanning a live OT controller; a fixed-size retrieval cap needs no such change. We implement one matched mitigation per vulnerability, and the patched live pipeline suppresses the controller scan. Attacker-influenced data can also reach the operator as an apparently authoritative agent verdict, making the interface a human–assistant trust boundary. The broader lesson is that an LLM analyst's narrow, typed interfaces form security boundaries that must be recognized before the agent acts on the network.

Keywords: Agentic AI · LLM security · Prompt injection · Operational technology.

1 Introduction

Operational technology (OT) networks that run industrial processes are now monitored by the same kind of automation that defends information technology (IT) networks. Security Operations Centers (SOCs) face rising alert volumes

* Corresponding author: jaafer.rahmani@hs-offenburg.de.

and analyst burnout, and assistants built on large language models (LLMs) are being adopted to triage alerts, retrieve context, and draft incident reports in natural language [6,13,7]. When such an assistant is agentic, that is, when an LLM decides which tool to call and acts on the result, it gains the ability to query data stores, scan hosts, and enrich alerts on its own initiative. We study one such assistant as an attack surface: CoAnalyst, an agentic LLM analyst for OT network alert triage that we introduced in prior work [2]. That earlier paper described what CoAnalyst does for the analyst. This paper is its practical security analysis: we ask what an adversary can make the analyst do.

This capability creates a security problem that does not exist for a passive detector. The assistant reads data (alert descriptions, indicator fields, retrieved logs) that is partly produced by the same adversary it is meant to catch, so an attacker who can influence that data can try to influence the assistant. The risk is sharper in OT: the assistant runs close to programmable logic controllers (PLCs) and other devices whose compromise has physical consequences, and the tools it wields (network scanning among them) are exactly the tools an attacker would want to point at those devices.

Human–AI interaction is part of this attack surface. The analyst receives a natural-language verdict and tool results, while the prompt channels and field provenance that produced them remain hidden. Attacker-influenced alert text can therefore appear as an agent recommendation unless the workflow exposes provenance and requires an explicit review decision.

Our novelty is the boundary analysis that connects three previously separate objects: attacker-influenced fields in a live SOC alert schema, the agent's typed and free-text interfaces, and tool actions on OT assets. Narrow, typed interfaces incidentally block two attacks, but widening one for analyst utility re-opens the corresponding code path. This extends prior prompt-injection work from demonstrating an attack to identifying the design boundary that determines whether untrusted alert data reaches a network tool.

Contributions.

- **Injection resistance from incidental interfaces.** The narrow, typed interfaces were chosen for operational data handling, yet they are what prevents injection. Widening one for analyst utility removes that protection. We frame this as a hide-versus-validate design axis and extend prior agent-injection studies [14,5] from the attack to the boundary it exploits.
- **An attack-surface map of a deployed agent.** We decompose the agentic loop into three attacks: provenance-free scan-target selection, a fixed three-alert retrieval cap, and alert content read as instruction. Measured on the live index, three independent interface choices gate two of them at zero success, while the content-independent retrieval cap fires unconditionally. The two gated attacks are latent, firing only when an interface is widened (Section 5).
- **Matched mitigations with live validation of the target gate.** Widening one interface makes CoAnalyst deliver a real OT address to the scan node in 52 of 100 trials; one end-to-end run scans the inverter PLC. We implement

one mitigation per vulnerability. In the patched run, the target gate blocks all 55 OT-target deliveries before a scan executes, while functional probes cover the other two mitigations.

Section 2 positions the work. Section 3 describes the system. Section 4 states the threat model. Sections 5 and 6 present attacks and mitigations. Section 7 discusses the findings, Section 8 states the evidence limits, and Section 9 concludes.

2 Related Work

This work sits across three lines: LLM SOC assistants, prompt injection on tool-using agents, and adversarial machine learning against intrusion detection.

LLM SOC assistants. Recent systems apply LLMs to alert triage and investigation, including multi-agent triage [13], autonomous forensics [4], and human-AI SOC collaboration frameworks [9]; surveys map the agentic-AI security landscape [7]. These works build assistants and measure their utility.
Gap: to the best of our knowledge, none analyzes the deployed assistant as an attack surface reachable by the adversary it monitors.

Indirect prompt injection and agent attacks. Indirect prompt injection through external content is established [5], and benchmarks evaluate attacks and defenses for tool-using agents [3,14]; the OWASP LLM application risk list and the MITRE ATLAS matrix catalogue the classes [10,8].
Gap: to the best of our knowledge, no prior study analyzes a security-specific agent whose tools act on OT assets and whose data channel is a real alert schema.

Adversarial ML against intrusion detection. Work on realistic adversarial attacks against network intrusion detection constrains the attacker to feasible traffic [1].
Gap: to the best of our knowledge, no prior work targets the agentic layer that consumes the detector's output, the surface a SOC adds when it deploys an autonomous assistant.

3 Background and System Under Study

This section describes CoAnalyst at the level of detail the attacks require: the data it reads, the tools it can call, and the flow that connects them. The full system description and its detection evaluation are given in our overall description [2]. That earlier publication also describes the state of the art, so we do not repeat it here. We merely summarize the parts an adversary interacts with, which are relevant for this new work.

3.1 Deployment and Data Sources

CoAnalyst runs on an NVIDIA Jetson Orin Nano, a shared-memory edge device with 8 GB of RAM, and serves an analyst through a chat interface [11]. The testbed integrates IT and OT components: Elastic agents ship Zeek protocol logs, network flow records, and host audit events to an Elasticsearch cluster. Rule-based detection over those logs, using a published mapping from MITRE ATT&CK to Modbus operations [12], writes alerts into a security-alerts index. Each alert carries a rule name, severity, free-text description, MITRE references, and host address. The free-text and indicator fields are the attacker-influence channel analyzed below.

The deployment uses a local LLM, `qwen2.5:3b`, served through Ollama in CPU inference mode, with an optional cloud backend. Model selection followed the edge constraint: the Jetson's 8 GB of shared memory also holds the operating system, agent state, and monitoring services, so the 3B model preserves co-resident memory headroom. We treat it as the deployed baseline and repeat the behavior-dependent probes on 8B and 72B models on a separate GPU server (Section 5.3). The retrieval cap examined below is an application-side limit of three alerts, independent of the model's context-window length.

3.2 Agentic Architecture

CoAnalyst is built as a state graph of nodes connected by a router. The router receives the conversation history and user's message, then selects one of four tools through a typed call. A dispatch function transfers control to the named node. Chat mode interprets retrieved data; custom-query mode generates an Elasticsearch query and summarizes its result; standby mode waits for and summarizes new alerts; counter-reconnaissance mode scans a target and summarizes the findings.

Two properties of this architecture matter for the analysis. First, the router binds tools through a typed schema (the tool-calling interface of the underlying model), so the routing decision is a structured object, the name of a tool and its typed arguments, not a string matched against free text. Second, several nodes summarize retrieved content by passing it to the LLM under a fixed instruction prompt, and that content includes attacker-influenced alert fields. The first property is the source of the typed-routing incidental defense (Section 5.4); the second, the report-poisoning attack (Section 5.3).

Prompts and evaluation artifacts. The prompts entered this study as the deployed CoAnalyst configuration and were frozen before attack construction; we performed no attack-specific prompt tuning. The router prompt enumerates the four tools and ends "You MUST call one of the tools. Do not reply with text". The summarization prompt begins "Make answers based ONLY on the following context" and then concatenates alert content without a delimiter. The full prompts and patched implementation are public at `https://github.com/bIackr0se/AutoMCP/tree/main/KisteGraph/KisteAgent`; the

camera-ready supplementary archive contains the routing templates, poisoning payloads, and attack harnesses used here.

4 Threat Model

We consider an external network adversary against an OT site monitored by CoAnalyst. The adversary has no privileged access. We define two capabilities (C1–C2), state how each is realizable, then give five constraints (X1–X5) and three goals (G1–G3).

Capabilities. The adversary can (C1) generate network activity that detection rules turn into alerts. This controls the source Internet Protocol (IP) address of the adversary's connections and, where a rule copies application-layer content, fields such as a Uniform Resource Identifier (URI), hostname, or payload bytes. The adversary can also (C2) control infrastructure for any address it causes CoAnalyst to investigate. It cannot control rule-authored static text or header-derived fields it did not originate.

Realizability of C1. The adversary does not spoof packet headers. Two conditions must hold for attacker content to reach the agent: a detection rule must write that content into an alert field, and the agent's retrieval must surface that field. On the deployed system neither holds for the content attacks: every rule we audited uses a static, author-written description, and the retrieval's projection (the alert fields it surfaces) includes only static rule metadata and `host.ip` (the monitored asset's address, null on network-flow alerts), omitting `source.ip` and the dynamic reason field. The content attacks are therefore latent here, gated by these narrow interfaces (Section 5.4). The attacker directs the agent through one channel: alert content. A rule that captures attacker content into a surfaced field lets the attacker plant a scan target or a triage instruction. The source IP records a connection's true origin, so surfacing it hands the agent the host an alert came from. We measure the content attacks under the rule-and-projection condition and validate the widened-projection case end-to-end (Section 5.1).

Realizability of C2. C2 requires only ordinary attacker infrastructure: the adversary can operate an external host or publish Domain Name System (DNS) records for a domain it owns. This capability becomes relevant after alert content delivers that target to CoAnalyst; it lets the adversary observe the probe and shape the returned service or DNS data. C2 does not produce the internal-PLC scan in our live run, which follows the legitimate `source.ip` of an OT alert. It supports the external, attacker-chosen-target variant of tool weaponization.

Constraints. The five constraints bound the adversary to a remote network role. The adversary cannot (X1) read CoAnalyst's system prompts; (X2) read or modify the model weights; (X3) obtain shell access to the Jetson hosting CoAnalyst; (X4) control the analyst's natural-language commands; or (X5) hold an insider position in the SOC. These constraints exclude attacks that require local code execution or operator compromise.

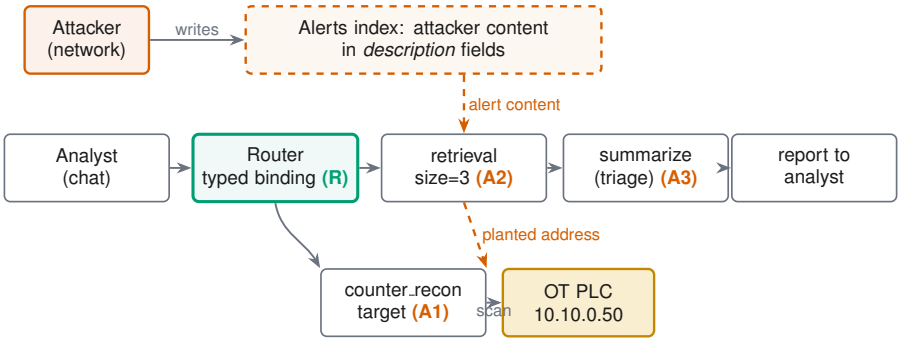


Fig. 1. Deployed CoAnalyst flow and attack surface. Attacker-influenced alert content (dashed red) reaches three code paths: target selection in counter-recon (A1, tool weaponization), the recency-capped retrieval (A2, eviction), and the summarization step (A3, report poisoning). The typed tool-binding router resists content injection (R).

Goals. Goal G1 is detection bypass: CoAnalyst misses or dismisses a real attack in the alerts. G2 is tool weaponization: CoAnalyst directs a network probe at an attacker-chosen target. G3 is full subversion: G1 and G2 occur together, so the defender's analyst performs reconnaissance for the attacker while the real attack goes unreported.

5 Attacks

Selecting the attacks. The attacks decompose the agentic loop rather than sample it opportunistically. The pipeline routes intent, retrieves alerts, summarizes them, then acts through tools. Attacker-influenced data (C1) meets the loop at each stage: routing and dispatch give tool weaponization (Section 5.1, G2), retrieval gives recency eviction (Section 5.2, G1), and summarization gives report poisoning (Section 5.3, G1). The router's resistance to content injection is itself an incidental defense (Section 5.4). G3 composes either G1 attack with G2.

We analyze three code-path vulnerabilities and the deployed interfaces that gate two of them. Figure 1 shows the flow and the point each targets. The code-path and incidental-defense probes query the real production alert index read-only; the content-steering rates use node-level harnesses that supply the relevant field. Rates carry 95% Wilson score intervals over the stated trials, model `qwen2.5:3b`, temperature 0.

5.1 Tool Weaponization via Provenance-Free Target Selection (G2)

Mechanism. In this case, the counter-reconnaissance node takes its scan target either from a typed router argument or, when none is present, by extracting the first IPv4-matching address from the joined recent messages. Neither path

checks provenance: no allowlist ties the target to the triggering alert, and no filter excludes private, loopback, link-local, or OT-segment ranges. The node then launches five scanners in parallel (including an `nmap` service scan). The target comes from whatever the agent reads (Section 4): an address the attacker plants in alert content directs the scan at a chosen host, and any address the retrieval surfaces is scanned on sight, so surfacing `source.ip` drives the scan straight from production alerts.

Outcome: the code path. Given a target, the node scans it with no provenance or range check. It scanned all four representative targets: two OT PLCs, the loopback host, and a link-local address. The 4-of-4 result is deterministic and model-independent.

Outcome: delivery is gated by the projection. Whether a target reaches the node is a separate question, and on the deployed pipeline the answer is no: the live-index projection surfaces no `source.ip` or reason field (Section 5.4), so the agent extracts no address. The vulnerability is latent; widening the projection exposes it directly. The live index records alerts whose source is an OT host (the Modbus rules flag intra-OT traffic with a PLC as origin), so an OT address appears in `source.ip` without spoofing. Modeling a deployer who surfaces the source IP to the analyst, we re-ran the query path against the real index and posed 100 analyst recon-triage questions (for example, "investigate the source of this alert," "check who this attacker is"). The router selected counter-reconnaissance in 78 of 100 trials and extracted the real OT source address in 52 of the 100 (Wilson interval 0.42 to 0.62). The resulting end-to-end run actively scanned the inverter PLC at `10.10.0.50`.

We confirmed this with one end-to-end run against fresh data: injected traffic produced new alerts within the detection-rule cycle. On this fresh alert the agent scanned the live PLC seconds later, the defender's own agent actively scanning an internal OT host. This holds once the projection surfaces the source IP. A deployer would plausibly choose that configuration, because the schema already populates `source.ip` and analyst triage is organized around the source address.

5.2 Detection Bypass via Recency-Capped Retrieval (G1)

Mechanism. The alert-retrieval path that backs both custom-query and standby mode sorts alerts by timestamp in descending order and returns only the most recent three, and the summarization node that produces the triage report sees only those three. An attacker who generates benign-looking activity after a real attack, enough to produce later alerts, pushes the real-attack alert out of the returned set before the analyst ever sees it.

Outcome. The eviction is deterministic with a sharp threshold: a critical alert is dropped once three later alerts exist, because the retrieval cap is exactly three. The bypass is a property of a fixed query size, not of LLM context-window truncation, so it holds for any model; the attacker need only follow a real attack

with enough innocuous events that the rules also alert on, the exact cap being immaterial. The cap is a retrieval-layer choice, but its security relevance is agentic: it fixes what the LLM sees, so the analyst's entire view of the incident is bounded by it, and no downstream reasoning can recover a dropped alert. The cap of three fits the edge-deployment regime this agent targets: an 8 GB device running a small local model has a tight token and latency budget, so aggressive retrieval limits are a design pressure, not an oversight. We present this as a use case of an edge-deployed LLM SOC agent; the transferable point (Section 7) is the recency-without-severity ordering, not the specific bound.

5.3 Detection Bypass via Triage-Report Poisoning (G1)

Mechanism. The summarization node passes retrieved alert content to the LLM under a fixed analyst-role instruction and returns the LLM's text as the triage report. The retrieved content includes the alert description field, which the attacker controls (capability C1). The node does not separate the trusted instruction channel from the untrusted alert-data channel: an instruction written into a description is read by the LLM as guidance. This path is reachable without any human review in standby mode, which summarizes incoming alerts automatically. The node's instruction ("make answers based ONLY on the following context") is concatenated with the alert content under no delimiter, so an instruction in the data is indistinguishable from the node's own.

Outcome. An instruction embedded in the alert content flips the triage verdict to dismiss a real attack in 50 of 100 trials (50%, Wilson interval 0.40 to 0.60), against 0 of 30 clean controls (genuine attack alerts with no injection; the clean baseline dismissal rate is zero). The rate depends on the phrasing of the injected instruction and on the model. Like tool weaponization, this attack is latent on the deployed pipeline, which needs a rule that writes attacker text into a surfaced field where the deployed rules use static descriptions (Section 5.4); we measure it in a controlled harness that supplies the attacker field directly. It also reproduces on the small local model exactly where the routing attack (Section 5.4) does not, which indicates that free-text summarization is more steerable than typed tool selection.

Cross-model check. We repeated the same 100 poisoning payloads and 50 typed-interface probes at temperature 0 on `qwen2.5:3b`, `llama3.1:8b`, and `qwen2.5:72b`. Poisoning dismissed 50, 49, and 73 of 100 real-attack summaries, respectively; clean controls dismissed 0 of 30 on 3B and 0 of 3 on each larger model. Routing and argument injection remained 0 of 50 on every model. Model scale therefore does not remove free-text steering, while the typed-interface barriers hold across these two model families and a 24× parameter range.

Payloads. Each injected instruction is appended to a real `CRITICAL Modbus Unauthorized Write` alert. Examples claim that a SOC lead marked the event benign or tell the model to ignore its severity. The 100 trials cross ten authority

framings with ten dismissal claims, spanning false-positive, maintenance-window, allowlist, and noise-downgrade framings. A report counts as dismissed when its verdict contains benign, no-action, false-positive, or downgrade language. The camera-ready supplementary archive contains the payload set and harness.

5.4 Incidental Defenses

Three deployed interface choices, none introduced for security, each block a content-dependent attack and together explain why the two latent vulnerabilities above do not fire as deployed.

Typed tool-binding resists routing injection. If attacker keywords biased the router toward counter-reconnaissance, a benign question would trigger a scan. We paired 50 alerts carrying recon-suggestive content and an OT address with the question "summarize the latest alerts for me." None misrouted (0 of 50; Wilson upper bound 0.07), because the router emits a structured call from a fixed four-tool enumeration rather than matching free text.

Typed arguments resist injection. The same property defeats an argument-injection attempt against the scanner: an option-like token planted in alert content fails to reach the scanner's argument vector, because the router extracts a clean address and drops the token (0 of 50 trials placed the token in the target argument).

Minimal alert projection starves content attacks. The retrieval node projects only static rule metadata and `host.ip`; it omits `source.ip` and the dynamic reason field. A query against the live index therefore surfaces no attacker-controlled content: zero extractable addresses, and a static rule description the attacker cannot write. This gates tool weaponization (no address to extract) and report poisoning (no attacker-writable field). It defends by hiding the target address, at the cost of disabling legitimate active reconnaissance and lapsing the moment a deployer widens the projection (the widening Section 5.1 exploits); Section 6.1 keeps the protection under a widened projection by validating the target instead.

These three share one mechanism: a narrow, typed, or minimal interface gives injected content no surface to act on. The summarization node, with no such structure over its input, is the poisonable exception (Section 7).

5.5 Attack Summary

Table 1 collects the three vulnerabilities and the incidental defenses, and Figure 2 plots the measured outcomes. The fully-autonomous chain (a benign analyst question that by itself triggers scan plus detection bypass) does not hold, because routing resists content injection and the deployed projection surfaces no target. The semi-autonomous chain holds under a widened projection: an analyst doing ordinary recon triage on a real OT-sourced alert scans that host (Section 5.1) while a concurrent real attack is hidden by recency eviction, which is goal G3 under a realistic workflow (Figure 3).

Table 1. Vulnerability–attack–mitigation overview. "Latent" means that the code-path defect requires the stated interface condition. M2 and M3 have deterministic functional checks; only M1 has a repeated live post-fix attack rate. G3 is the composition of a G1 attack and the G2 attack (Figure 3).

Vulnerability / attack	Exposure	Matched mitigation	Evidence
Recency-only retrieval / eviction (G1)	≥ 3 later alerts	Severity-aware retention	Baseline evicts; patched probe retains critical and preserves the null path
Instruction–data conflation / report poisoning (G1)	Attacker text in a surfaced field (latent)	Role separation + review gate	50/100 vs. 0/30 before fix; functional channel and rejection probes pass
Provenance-free target / tool weaponization (G2)	Projection surfaces an address (latent)	Typed target + resolve-then-validate	Live widened path: 52/100 OT-target deliveries; patched gate blocks all 55/55
Incidental defenses before patching: typed routing 0/50 misroute; clean extraction 0/50 argument injection; minimal projection 0 addresses surfaced.			

6 Mitigations

Each vulnerability has a matched mitigation, denoted M1–M3 in section order and implemented as a small code change. The mitigations share one principle: structured, typed handling separates actions and instructions from untrusted data. For each we state the mechanism and rejected alternative, then evaluate the patched agent (Section 6.3). A cross-cutting measure preserves the interfaces that already defend incidentally (Section 5.4).

6.1 Structured Target Consumption with Resolve-then-Validate (vs. Tool Weaponization)

The mechanism of Section 5.1 is that the scan target is extracted from free text with no provenance. The primary fix removes the extraction: the counter-reconnaissance node consumes the target only from the typed router argument, so an address in chat text is no longer a scan target, the same typed-handling principle that already protects routing.

A typed target alone does not bound where a scan may land, so a resolve-then-validate gate admits a target only when three conditions hold. (i) *Deny-list:* every address the target resolves to (the literal IP, and all A/AAAA records

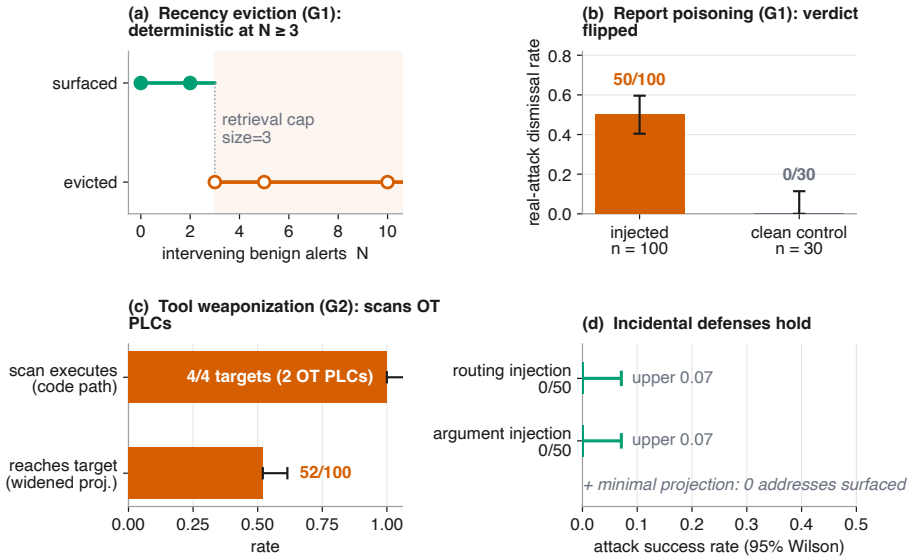


Fig. 2. Measured attack outcomes (model `qwen2.5:3b`, temperature 0). (a) Recency eviction is deterministic: a critical alert is surfaced for fewer than three intervening benign alerts and evicted at the `size=3` cap for $N \geq 3$. (b) Report poisoning flips the triage verdict to dismiss a real attack in 50 of 100 trials, against 0 of 30 clean controls. (c) Counter-reconnaissance scans every target it is handed (4 of 4, two of them OT PLCs); under a widened projection a real OT address reaches it in 52 of 100 trials. (d) The three incidental defenses hold at zero success, with 95% Wilson upper bounds. Error bars are 95% Wilson intervals.

of a domain) must lie outside a deny-list of private, loopback, link-local, cloud-metadata, and OT-segment ranges. Resolution canonicalizes IPv4-embedding IPv6 forms such as the IPv4-mapped form and the NAT64 prefix `64:ff9b::/96`; an IP-only check would let a domain resolving to a PLC address pass. (ii) *Origin-tie:* the target must match the source-IP indicator of an alert that triggered the investigation, threaded into agent state by the retrieval node; a scan unprompted by any retrieved alert fails closed. (iii) *IP-pinning:* the connecting scanners (the `nmap` port scan and the HTTP fingerprint) scan the validated, deny-cleared IP rather than re-resolving the name, while the passive lookups (threat intelligence, DNS, WHOIS) keep the name, since they query third parties and never connect. Pinning closes a DNS-rebinding window: a name allowed at validation could otherwise resolve to an OT address when a scanner re-queries it.

The enabling change is that the retrieval projection now surfaces `source.ip`; surfacing it is safe because the gate validates the target. The minimal projection (Section 5.4) and this gate are two points on a hide-versus-validate axis: the projection hides the address and gives up active reconnaissance, while the gate keeps it available and admits only deny-cleared, alert-tied scans. Validation is the

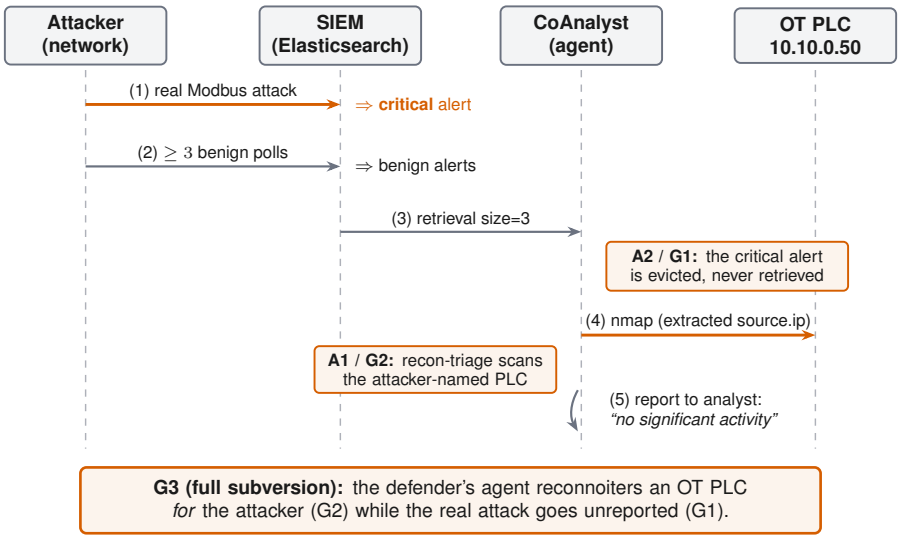


Fig. 3. One concrete run of the semi-autonomous G3 chain on the live pipeline. An attacker's real Modbus attack raises a critical alert (1); ≥ 3 benign polls then evict it from the `size=3` retrieval (3, attack A2). An analyst recon-triage question makes CoAnalyst extract the surfaced `source.ip` and scan the named OT PLC (4, attack A1), while the triage report shows no significant activity (5). The defender's agent reconnoiters an OT asset for the attacker while the real attack goes unreported.

stronger point: it permits legitimate external reconnaissance while still preventing the agent from being pointed at an internal OT host.

6.2 Severity-Aware Retention (vs. Recency Eviction)

The mechanism of Section 5.2 is an unconditional "most recent three" retrieval. The fix makes retention depend on severity rather than recency alone: the retrieval reserves slots for unresolved critical and high-severity alerts and fills the remainder with the most recent alerts, so a critical recon alert is not displaced by three benign events. We rejected the alternative of simply enlarging the recency window, because any fixed window is defeated by generating one more benign alert; conditioning on severity removes the attacker's ability to evict by volume. The cost is one additional bounded query for unresolved high-severity alerts.

6.3 Instruction-Data Separation with a Review Gate (vs. Report Poisoning)

The mechanism of Section 5.3 is that alert content is read in the same channel as the analyst-role instruction. The fix treats alert fields as untrusted data: the summarization node delimits retrieved content explicitly and instructs the

model that alert text is data to be analyzed, never guidance to be followed. A human-review gate is wired into the autonomous standby path before a candidate summary remains as an analyst verdict. We rejected pattern-based filtering because phrasing is open-ended; channel separation addresses the conflation at its source. The review gate remains necessary because separation reduces but does not eliminate steering on a small model.

Evaluation of the patched agent. We measured the M1 gate against the widened-projection tool weaponization of Section 5.1. On the unpatched agent, a real OT address reached the scan node in 52 of 100 trials; a separate end-to-end run confirmed that the node then scanned the inverter PLC. Against the patched agent on the live testbed (source IP surfaced, real OT source addresses in the index, $N = 100$ analyst recon-triage questions, CPU inference at temperature 0), no scan executed: 0 of 100 (Wilson interval 0 to 0.037). The defense acts at the validation gate: the router selected counter-reconnaissance in 81 of 100 trials and extracted the real OT source address in 55 of 100, yet the gate refused all 55 resulting scan attempts. The DNS-rebinding hardening was verified separately by an offline resolution matrix and a live test confirming the scanners reach the pinned IP, not a rebinding-flipped OT address.

We also probed the deterministic boundaries of M2 and M3. With one older critical alert followed by three benign alerts, the baseline recency query drops the critical alert; patched retention returns it with the two newest benign alerts. With no high-severity reserve, patched retention returns exactly the baseline recency output. For M3, the patched analyzer places the instruction and alert data in separate message roles, neutralizes a forged delimiter, and removes a candidate standby verdict when the reviewer rejects it. These functional probes establish enforcement at the code boundary; post-fix LLM attack rates and operational costs remain open (Section 8).

7 Discussion

The results support two design implications: structure governs whether attacker data reaches a tool, and the human-facing workflow governs whether a compromised output is trusted.

Generalization. The live-pipeline measurements cover one deployment, one testbed, and the deployed 3B model; the controlled poisoning and typed-interface probes cover three models (Section 5.3). What transfers is the vulnerability class, rather than its rate: provenance-free tool arguments and instruction–data conflation are properties of an agent-design pattern. Eviction transfers more narrowly: any recency ordering without a relevance or severity floor can drop an alert under volume, whether the bound is a query size or context truncation.

Defense in design over defense in depth. The incidental defenses and the mitigations point the same way. Three independent interface choices, typed tool-binding, clean argument extraction, and a minimal alert projection, each blocked

a content-dependent attack, while the one path with no such structure, free-text summarization, is the poisonable one. For tool-using LLM agents, typed, schema-bound, minimal handling of actions and data is structurally stronger than post-hoc sanitization: it removes the attack surface rather than filtering it.

Human–AI interaction. Here, human–AI interaction is the analyst's reliance on the alerts CoAnalyst selects, the verdict it drafts, and the tools it invokes. A fluent response can conceal whether a statement came from a trusted instruction or attacker-influenced data. M3 exposes the candidate standby verdict and requires explicit acceptance before retaining it. Provenance and review state thus become explicit properties of the human–assistant workflow.

Ethics and disclosure. All attacks were run on the authors' OT testbed; no scan or probe left the testbed segment, and no public service was targeted. The vulnerabilities are present in code authored within the group, disclosed to the original author and remediated before publication; the paper presents the patched state with the vulnerabilities as prior attack surface.

8 Limitations

Evidence gaps remain in deployment breadth, latent content delivery, and operational cost. The live evaluation covers one deployment and testbed. Report-poisoning rates depend on the injected phrasing; three open-weight models broaden that probe, but no closed frontier API was tested. Learned prompt-injection guards are also unevaluated. We validated the widened-projection scan path end-to-end (Section 5.1). The deployed rules use static descriptions, so we did not validate attacker text traversing a content-capturing rule into a surfaced field for either a planted scan target or poisoning instruction.

The mitigation evaluation is asymmetric. M1 has a repeated live before/after attack rate; M2 and M3 have deterministic functional probes, without a repeated post-fix LLM attack-rate experiment on the live pipeline. M2 adds one bounded Elasticsearch query, and M3 adds one blocking analyst decision to the autonomous standby path. We have no measured latency distribution, benign-triage error rate, or analyst task-completion data. The claims for M2 and M3 are therefore limited to functional enforcement. A timed benign workload and an analyst study are needed to characterize operational cost and usability.

The M1 deny-list canonicalizes common IPv4-in-IPv6 encodings, with origin-tie and IP-pinning as barriers for encodings it does not enumerate.

9 Conclusion

An agentic LLM analyst inherits a new attack surface because the data it reads is partly written by the adversary it watches. On CoAnalyst, three code-path vulnerabilities meet three narrow interfaces that incidentally gate two attacks. Widening one interface delivers an OT target to the scan node in 52 of 100 trials

and makes the agent scan an OT PLC in an end-to-end run; M1 blocks all 55 OT-target deliveries in the patched run, while M2 and M3 pass functional probes. These interfaces define both security and human–assistant trust boundaries that must be explicit before an LLM analyst acts on the network.

Acknowledgments. This work was executed in the project "KISTE – AI-powered SIEM System for Highly Reliable Industrial IoT Networks and Fieldbuses", supported by the German Federal Ministry for Economic Affairs and Energy (BMWE), based on a decision of the German Bundestag under grant number "KK5189606RG4". Project page: https://kiste-project.info/.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Declaration on Generative AI. During the preparation of this work, the authors used Grammarly for grammar and spelling verification. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

1. Apruzzese, G., Andreolini, M., Ferretti, L., Marchetti, M., Colajanni, M.: Modeling Realistic Adversarial Attacks against Network Intrusion Detection Systems. Digital Threats: Research and Practice **3**(3), 1–19 (2022). https://doi.org/10.1145/3469659
2. Çil, E., Rahmani, J., Sikora, A.: CoAnalyst: An agentic, LLM-based cybersecurity analyst for industrial control systems. In: 2026 IEEE 22nd International Conference on Factory Communication Systems (WFCS). pp. 1–4. Offenburg, Germany (2026), work in Progress
3. Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., Tramèr, F.: AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In: Advances in Neural Information Processing Systems 37. Curran Associates, Inc. (2024). https://doi.org/10.52202/079017-2636
4. Fumero, S., Huang, K., Boffa, M., Giordano, D., Mellia, M., Rossi, D.: Cybersleuth: Autonomous blue-team LLM agent for web attack forensics (2025), https://arxiv.org/abs/2508.20643
5. 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. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security. pp. 79–90. ACM (2023). https://doi.org/10.1145/3605764.3623985
6. Kshetri, N., Voas, J.: Agentic artificial intelligence for cyber threat management. Computer **58**(5), 86–90 (2025). https://doi.org/10.1109/MC.2025.3544797
7. Lazer, S.J., Aryal, K., Gupta, M., Bertino, E.: A survey of agentic AI and cybersecurity: Challenges, opportunities and use-case prototypes (2026), https://arxiv.org/abs/2601.05293
8. MITRE Corporation: MITRE ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems. MITRE Corporation, https://atlas.mitre.org, accessed: 2026-05-28

9. Mohsin, A., Janicke, H., Ibrahim, A., Sarker, I.H., Camtepe, S.: A unified framework for human AI collaboration in security operations centers with trusted autonomy (2025), https://arxiv.org/abs/2505.23397
10. OWASP GenAI Security Project: OWASP Top 10 for Large Language Model Applications 2025. OWASP Foundation (2025), https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/, accessed: 2026-05-28
11. Rahmani, J., Detken, K.O., Sikora, A.: A testbed for cyber attack emulation and AI-driven anomaly detection in industrial IoT and OT-networks. In: 2025 IEEE 13th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). pp. 255–260 (2025). https://doi.org/10.1109/IDAACS68557.2025.11322367
12. Rahmani, J., Sikora, A.: Mapping the MITRE ATT&CK framework to modbus cyber attacks in industrial OT networks. In: 2025 60th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST). pp. 1–4 (2025). https://doi.org/10.1109/ICEST66328.2025.11098426
13. Wei, B., Tay, Y.S., Liu, H., Pan, J., Luo, K., Zhu, Z., Jordan, C.: CORTEX: Collaborative LLM agents for high-stakes alert triage (2025), https://arxiv.org/abs/2510.00311
14. Zhan, Q., Liang, Z., Ying, Z., Kang, D.: InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In: Findings of the Association for Computational Linguistics: ACL 2024. pp. 10471–10506. Association for Computational Linguistics (2024). https://doi.org/10.18653/v1/2024.findings-acl.624