Submitted to
International Conference on Trends in Artificial Intelligence, Data Analytics and Cyber Security (ICTAIDACS 2025)
ESP Journal of Engineering & Technology Advancements (ESP-JETA)
ISSN: 2583-2646 / Volume 5 Issue 3 July-September 2025
Paper Id: JETA-V5I3P127/ Page No: 212--221

Original Article

AgenticSecOps for Microservice APIs: Policy-Constrained Multi-Agent Investigation and Response to Application-Layer Incidents

Rajesh Balaji
Independent Researcher, Seattle, WA, USA
Anna University, Chennai, India

Received Date: 10 September 2025 Revised Date: 20 September 2025 Accepted Date: 30 September 2025

Abstract: *AI agents can investigate API security alerts and propose responses far faster than people, but in an application they must read fields that attackers write: profile names, order notes, support tickets. An agent that reads such text and can suspend accounts gives anyone who can fill in a form a way to steer it. This paper presents AgenticSecOps for microservice APIs. Five cooperating agents investigate and respond to application-layer incidents, and each is treated as an untrusted workload. Only one agent reads user-written content, and it can neither act nor close an alert alone. A second agent works only from the authorization engine's decision logs, which users cannot write into. A response agent chooses from a fixed catalogue of application-level actions (step-up, throttle, revoke, suspend, disable a feature) using the application's threat model. A policy gate binds every action to evidence, prices it in legitimate requests and users locked out, and requires a human signature for anything that locks someone out. The design specifies an inverse or compensating procedure for each state-changing action; successful rollback has not been demonstrated against live systems.*

We compare rule-based automation, a single AI agent and AgenticSecOps on the generated alert episodes and request streams of five held-out weeks from an earlier detection study: about a million requests and 200 incidents across eight abuse scenarios. Agent error and prompt-injection susceptibility are explicit parameters shared by the agent-based designs. No language model is run. With agents that are wrong 10% of the time and follow injected instructions 30% of the time, AgenticSecOps reduced wrongful lockouts from 0.68 to 0.02 a day and injection success from 30% to 3% compared with the single agent. It contained less attack activity, 35% against 42%, and against 49% for rule-based automation that acts in seconds. Ablations show that the Zero Trust policy checks provide the safety, agent coordination the injection resistance, and threat-model context most of the containment. The manuscript describes a simulation and a protocol for subsequent validation with real agents; public availability of the executable artifacts must be confirmed before submission.

Keywords: *AI Agents, LLM Security, Prompt Injection, Incident Response, API Security, Microservices, Zero Trust, Human-In-The-Loop, Open Policy Agent*

I. INTRODUCTION

Every API alert eventually lands with a person who has to answer three questions. Is this real? Who or what is behind it? What should we do about it? For an alert such as "support agent s3 read 400 customer profiles in half an hour", answering means pulling the agent's recent requests, checking whether those customers had open tickets, looking at the session's device and network, and deciding whether to throttle, revoke or call the agent's manager. It takes time, and it is repetitive enough that large language model agents look like an obvious way to speed it up.

The trouble is what an agent investigating an application incident has to read. Account names, delivery addresses, order notes, support-ticket text and user-agent strings are all written by users, and therefore by attackers. A customer who knows an AI reads order notes can write one that says the activity was approved, or that the real culprit is another account. Indirect prompt injection of this kind is well documented [6] and is first on the OWASP list of LLM application risks [7]. An agent that reads these fields and can suspend accounts or reverse refunds hands that power to anyone who can type into a form.

This paper completes a three-part study on one microservice application. The first part derived request-level authorization from a threat model and measured what it closes [22]. The second added detection and graded responses for abuse the policy allows [23]. Its responses were deliberately small, such as step-ups and throttles, so that a person could decide the rest. Here we ask how much of that remaining work can safely go to cooperating AI agents, and what it takes to keep them safe.

This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/2.0/)

Our answer, AgenticSecOps, treats every agent as an untrusted workload, in the Zero Trust sense [1]. The agent that reads user-controlled text cannot act. A second agent works only from structured decision logs, which users cannot write into. Every proposed action must cite evidence about the principal it targets, is priced in legitimate requests and users it would lock out, needs a human signature if it would lock anyone out, and is rolled back automatically if it proves wrong.

The contributions are as follows.

- A threat model for AI agents responding to application-layer incidents, in which attackers reach the agent through ordinary user-controlled fields.
- A multi-agent design that separates reading user content from acting, adds an independent correlation agent that works on decision logs only, and constrains every action through a policy gate with evidence checks, measured cost, human approval and rollback.
- An application-level response catalogue (step-up, throttle, revoke, suspend, disable a feature) with each action's cost measured on generated request streams rather than assumed.
- A comparison of rule-based security operations, a single AI agent and AgenticSecOps on the incidents and traffic of [23], under a common, parameterised model of agent error and prompt injection. It includes ablations that remove the threat-model context, the agent coordination and the Zero Trust policy checks one at a time.

No language model is run in the experiment. The agents' mistakes are modelled explicitly, so the results describe how each architecture handles a fallible agent rather than how good any particular model is. Section VIII sets out how to repeat the experiment with real agents. Section II reviews related work and Section III sets out the threat model. Sections IV and V describe the architecture and the action catalogue. Sections VI and VII cover the experiment and results, and Section VIII discusses limitations and concludes.

II. BACKGROUND AND RELATED WORK

A. Agents for security operations

LLM agents interleave reasoning with tool calls, as in ReAct [2], and can divide work between cooperating roles, as in AutoGen [3]. The Model Context Protocol [4] standardised how tools are exposed to them. On the offensive side, PentestGPT [5] and Fang et al. [9] showed that agents can carry out multi-step attacks with limited guidance, which raises the bar for defenders and makes defensive agents attractive. NIST's updated incident-response guidance [25] frames detection, analysis, containment and recovery as one continuous risk-management activity, the loop such agents would sit in. Alert fatigue among analysts [12] is the practical motivation, and context-based triage such as NoDoze [21] showed how much of that load is avoidable.

B. Prompt injection and agent safety

Greshake et al. [6] showed that instructions hidden in content an LLM application retrieves can take control of it. Benchmarks such as InjecAgent [16] and AgentDojo [15] have since measured how often tool-using agents follow injected instructions, and found the rates substantial and model-dependent. The OWASP Top 10 for LLM Applications 2025 [7] lists prompt injection and excessive agency among its leading risks, and NIST AI 600-1 [8] calls for human oversight of consequential generative-AI actions. Recent defences take an architectural view. CaMeL [17], for example, separates the control flow a model plans from the untrusted data it processes, so that injected text cannot change which tools are called. Our design applies the same principle to incident response, together with classic separation of privilege [24]: the role that reads untrusted text is not the role that acts.

C. Application-level response mechanisms

The responses available for API incidents are standard: step-up authentication challenges [18], token revocation [19], continuous-access-evaluation signals between services [20], throttling at the gateway, account suspension and feature flags. Our earlier work [23] measured the cost of the first three on generated request streams. What is missing is a way to decide, safely and quickly, which of them to use on a given incident.

D. Position of this work

Work on agent safety mostly evaluates agents against benchmarks of injected tasks. Work on security automation mostly assumes the automation is trustworthy. We combine the two for a concrete setting. We give AI agents the job of responding to API incidents, assume they will sometimes be wrong or manipulated, and measure how much harm each architecture lets through, using synthetic request streams generated for a policy-evaluated application model.

III. THREAT MODEL

A. Setting

The setting is the AegisCommerce application from [22] and [23]. Seven microservices behind a gateway serve 400 customers, 6 support agents and anonymous visitors. Request-level authorization runs in Open Policy Agent, and the risk-adaptive detector from [23] raises alert episodes on its decision logs. Each episode names one principal: a customer, a support agent or an anonymous client. It carries the decision-log windows that triggered it and the risk score, and in the evaluation it carries ground truth. The security operations function has to decide, for each episode, whether it is real and what to do.

B. Adversaries

- A1, the application attacker. Carries out the eight abuse scenarios of [23], drawn from the OWASP API Security Top 10 [13]: credential stuffing, account takeover, object-ID enumeration, function-level probing, scraping, insider refund abuse, insider data harvesting and SSRF attempts from a hijacked session.
- A2, the agent manipulator. Any attacker who knows that an AI agent reads incident evidence, and who writes instructions into fields the agent will read: a profile name, a delivery address, an order note, a user-agent string, the text of a support ticket. The goal is either to have the alert dismissed or to redirect the response onto someone else. A typical redirect target is a support agent, whose suspension would slow the real investigation.
- A3, the compromised agent. An attacker who controls one agent's process or credentials and tries to use its permissions directly.

The identity provider, the policy engine and the people approving actions are trusted. A malicious approver is out of scope.

C. What can go wrong

Three kinds of harm matter, and the evaluation measures each:

- Missed containment: a real incident is dismissed, or contained too late to matter.
- Wrongful action: the response hits an innocent principal. The worst case is a lockout, such as a revoked or suspended account, or a disabled feature. A milder case is an unnecessary step-up challenge or throttle.
- Collateral disruption: even a correct action stops legitimate work. Suspending a taken-over account also stops its real owner, and suspending an insider stops their normal duties.

D. Design goals

- G1, no lockout without a person. Any action that fully blocks a principal or disables a feature needs a human signature.
- G2, separation of reading and acting. The component that reads user-controlled text cannot propose or execute actions.
- G3, evidence-bound actions. A proposal may only target a principal that the episode's own evidence names.
- G4, priced proposals. Every proposal states what it would cost in legitimate requests and locked-out users before anyone approves it.
- G5, reversibility. Every executed action records its inverse and is rolled back automatically if it proves wrong.

IV. ARCHITECTURE

AgenticSecOps handles each alert episode with five agents, a policy gate and a human approval path (Fig. 1). Every agent is treated as a workload that could be wrong, manipulated or compromised, and its authority is limited accordingly.

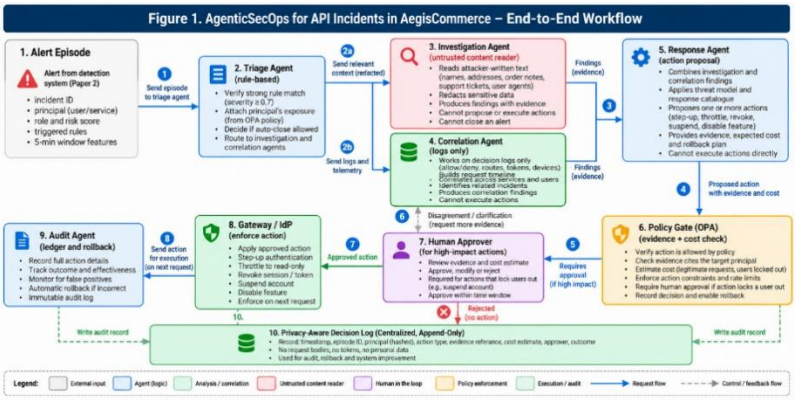


Figure 1: Agenticsecops for API Incidents · 5 Agents, Policy Gate, Human Approval, Rollback

Figure 1. Proposed AgenticSecOps architecture for policy-constrained incident investigation and response in AegisCommerce. Five specialized agents process API security alerts through separate evidence channels. A policy gate validates response proposals against evidence, estimated disruption, and approval requirements. Low-impact actions may execute immediately, while actions that lock out principals require human approval. An audit component records decisions and supports rollback.

A. Agent roles

Triage agent. Receives the episode as typed fields: principal, role, risk score, the rules that fired and the window features. It attaches the principal's exposure from the authorization policy. It enforces one hard rule: an episode with a strong rule match (severity 0.7 or more) cannot be closed by the agents alone. Investigation agent. The only agent that sees user-written content: names, addresses, order notes, support tickets, user agents. It reads them through a tool that redacts tokens, card fragments and contact details and marks the content as data. It has read-only tools for request history, account history and session details. It produces a findings record that cites evidence by identifier and names the principal it believes is responsible. It cannot propose or take any action.

Correlation agent. Works only from the policy engine's decision logs: principal, route template, whether the object belonged to the caller, allow or deny, session context. The evaluation treats these fields as trusted, but a real implementation must validate provenance and prevent user-controlled strings from crossing into the trusted schema. It reaches its own verdict without seeing the investigation agent's narrative. If the two verdicts differ, the episode goes to a person with both. Response agent. Takes agreed findings and chooses one action from a fixed catalogue (Section V), guided by the threat-model context: the principal's exposure and which kind of response fits which kind of signal. It never sees user-written text. Audit agent. Records every message, tool call, gate decision and action in an append-only ledger with each action's inverse, and applies the inverse when a rollback is triggered.

B. Identity and least privilege

As in [22], each agent runs with its own SPIFFE workload identity [11], and its tools are served by tool servers that check that identity. The investigation and correlation agents' tools are read-only. Only the gateway and identity-provider tools can change state, and they accept requests only from the gate. Under the proposed least-privilege tool configuration, a compromised investigation agent would have read-only access; this isolation was not validated in a deployed agent system. Under the proposed gate-enforced tool permissions, a compromised response agent could propose an action but not execute it directly; this remains a design property to test.

C. The gate and the person

Every proposal passes the gate, an Open Policy Agent policy that checks two things. The first is that the proposal targets the principal the evidence is about. The second is what the action would cost, in legitimate requests and in principals locked out. Actions that lock nobody out, step-ups and throttles, run immediately. Anything that locks someone out waits for a person, who sees the evidence, the proposal and its cost. The proposed tools record compensating actions for reversible state changes, and the audit agent undoes it automatically if the approver reverses it or the principal clears a support identity check.

V. ACTIONS, EVIDENCE AND GATE RULES

A. Action catalogue

Table 1: Application-level action catalogue

Action	Effect	Duration	Locks someone out?	Inverse
Step-up	Challenge the session for a second factor (CAPTCHA for anonymous clients); end it on failure	Once	No	None needed
Throttle	Session becomes read-only, own objects only, 10 requests a minute	30 min	No	Lift the throttle
Revoke	Revoke all of the principal's OAuth tokens [14] and block new ones	60 min	Yes	Re-enable
Suspend	Disable the account	24 h	Yes	Re-enable
Disable feature	Turn off an endpoint for everyone, here image import (E4), and revoke the principal	24 h	Yes	Re-enable

The response agent cannot write arbitrary API calls. It chooses from five parameterised actions, each implemented by a reviewed tool that changes gateway or identity-provider state and records its own inverse (Table I).

B. Choosing the smallest sufficient action

With threat-model context, the response agent maps what the episode shows to the smallest action that addresses it, following [23]. Evidence of server-side request forgery leads to disabling the affected feature. A very high risk score leads to revocation. A sign that the account may be in the wrong hands, meaning a new device and network or failed logins, leads to a step-up. Anything else, meaning abuse by the real account holder, leads to a throttle. The context also tells the agent each principal's exposure, so it knows that a support agent's anomaly matters more than an anonymous visitor's. Without that context, an agent has no principled basis for anything smaller than revocation.

C. Evidence handling

The raw-content tool redacts tokens, card fragments and contact details before returning user-written fields to the investigation agent, and wraps them in a marked data block. The correlation agent receives only decision-log records: principal, route template, object relation, allow or deny, and session context. None of these fields can be written by users, which reduces its exposure to direct natural-language injection; structured records and their upstream provenance must still be validated. Findings cite evidence by record identifier, never by copied text.

D. Gate rules

The gate is an Open Policy Agent [10] policy with three rules, applied in order:

- Evidence check. The proposal's target must be the principal that the episode's evidence records belong to. A proposal targeting anyone else is rejected and sent to a person, who works from the evidence and re-targets it or drops it.
- Cost check. The gate computes what the action would block, in requests and in principals locked out, from recent traffic. Step-ups and throttles, which lock nobody out, run at once.
- Approval. Revocations, suspensions and feature disables wait for a human signature. The approver sees the evidence, the proposal and its cost.

After execution, the audit agent watches for a rollback trigger: the approver reversing the decision, the principal passing a support identity check, or the evidence being retracted. When triggered, it applies the inverse and opens a review.

Evidence boundary. The reported comparisons are outputs of a parameterized simulation driven by synthetic API traffic and detector episodes. No language model, agent tool server, human approver, live identity provider, or production response action was executed. The human decision model, action delays, injection susceptibility, and rollback outcomes are assumptions rather than observed operational measurements.

VI. EXPERIMENTAL SETUP

A. What is Measured and What is Modelled

The question is architectural: given AI agents that are sometimes wrong and sometimes manipulated, how much harm does each security-operations design let through? No language model is called. Agent errors and injection susceptibility are explicit parameters, shared by both agent-based designs, so differences between designs come from their structure.

The environment is instantiated using synthetic, policy-evaluated request streams rather than observed production traffic. Every design works on the generated alert episodes and request streams of the five held-out test weeks from [23]: about 200,000 requests a week, 200 incidents across eight scenarios, plus that paper's unseen benign events. Every action a design takes is replayed against those request streams. Its benefit, in attack requests blocked after the action, and its cost, in legitimate requests blocked and principals wrongly locked out, are counted from the traffic rather than assumed.

B. Alert Stream

The risk-adaptive detector of [23] raised between 58 and 71 alert episodes per test week (333 in total), of which 32 to 39 per week were attacks. Several incidents produced no episode, because that detector missed them. No design can contain an incident it never hears about.

C. Designs Compared

B1 uses the same action catalogue and the same threat-model-driven action choice as B2, so the comparison isolates gating, coordination and separation of reading from acting.

Table 2: Security-Operations Designs

Design	Investigation	Action	Safeguards
B0 Rule-based SOAR	None: rule severity ≥ 0.7 acts automatically; 0.4 to 0.7 goes to an analyst queue; anomaly-only episodes closed	Revoke	Analyst review for medium severity
B1 Single AI agent	One agent reads everything, including user-written fields	Smallest action, or suspend	None; acts directly
B2 AgenticSecOps	Investigation agent plus correlation agent; strong rule matches cannot be closed by agents; disagreements go to a person	Smallest action from the threat model, or suspend	Evidence check, cost check, approval for lockouts, rollback
B2 – context	As B2	Always revoke	As B2
B2 – coordination	Investigation agent alone	As B2	As B2
B2 – Zero Trust checks	As B2	As B2	No evidence check, no cost check, no approval; rollback kept

D. Agent and Human Model

Table 3: Model Parameters (Defaults)

Parameter	Value
Agent verdict error	0.10
Agent names the wrong principal	0.05
Agent over-reaches (suspends when a smaller action suffices)	0.20
Attack episodes carrying an injection in a user-written field	50%; half aim to dismiss, half to redirect onto an innocent principal
Probability that an agent reading user-written fields follows it	0.30
Correlation of the two B2 agents' verdict errors	0.5
Human approves a correct / an incorrect proposal	0.95 / 0.10
Human resolves a disputed verdict correctly (also the B0 analyst)	0.90
Median latency: one agent / two agents / analyst / approval	2 / 3 / 15 / 10 min (lognormal)
Time a wrongful lockout lasts: manual restore / B2 rollback	median 90 min / 10 min
Step-up outcome	As [23]: stolen sessions and hijacked sessions fail, bots fail 90%, insiders and own-account abusers pass, benign users pass 97%

E. Metrics

- Attack activity stopped: for each incident, the share of its requests after the first action on it that were blocked, averaged over all 200 incidents. Undetected or unanswered incidents count as zero.
- Incidents contained: the share of incidents with at least 90% of their post-action requests blocked.
- Wrongful lockouts per day: revocations, suspensions or feature disables that hit a principal not involved in the incident.
- Wrongful soft actions per day: step-ups or throttles on a principal not involved in the incident.
- Legitimate requests blocked per day, in total and by wrongful actions only.
- Verdict accuracy: the share of episodes acted on if and only if they were real.
- Investigation time: minutes from an attack episode's first alert to the final decision.
- Injection success: the share of injected episodes in which the payload caused a dismissal or an action on the redirect target.
- Human touches per day.

Each design runs 20 times on each of the five weeks (100 runs). Sensitivity sweeps use 10 runs per week. Standard deviations for the main rates are below 0.09.

F. Reproducing the results

```
python3 sim.py   # needs ../app1 and ../app2; about 1 minute -> results.json
```

VII. RESULTS

AgenticSecOps almost eliminated wrongful lockouts: 0.02 a day, against 0.68 for a single AI agent and 3.6 for rule-based automation. It held prompt-injection success to 2.8%, against 30% for the single agent. It paid for this in containment. It stopped 35% of attack activity, against 42% for the single agent and 49% for rule-based automation, because actions that lock someone out wait for a person, and attacks on an API are often over within minutes.

A. Main comparison

Table 4: Outcomes By Design (Mean Of 100 Runs Over Five Test Weeks)

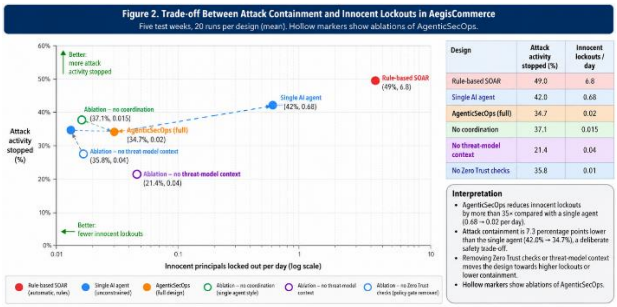
Metric	B0 Rule-based SOAR	B1 Single AI agent	B2 AgenticSecOps
Attack activity stopped	48.9%	42.0%	34.7%
Incidents contained (≥ 90% stopped)	48.9%	38.2%	29.8%
Wrongful lockouts per day	3.58	0.68	0.02
Wrongful step-ups or throttles per day	0	0.32	0.48
Legitimate requests blocked by wrongful actions, per day	3.7	11.7	0.05
Legitimate requests blocked in total, per day	32.5	243.7	287.5
Verdict accuracy	52.5%	86.6%	90.8%
Investigation time, median / p90 (min)	0.5 / 22.1	2.0 / 2.0	9.7 / 26.1
Injection success	n/a	30.2%	2.8%
Human touches per day	2.2	0	7.6

Rule-based automation contains the most because it acts in half a minute on any strong rule match. Credential stuffing, enumeration and probing all trip a strong rule, and the client is revoked before the attack is over. But it does not investigate. It acted on 84% of benign episodes, including the customers hit by the buggy client release in [23], and it locked out 3.6 innocent principals a day. The single agent investigates, so it acts on far fewer benign episodes, and it is fast. Its problem is that every mistake and every successful injection becomes an action. Seven in ten of its wrongful actions were lockouts, which cost 11.7 legitimate requests a day, and three in ten injected incidents went the attacker's way.

AgenticSecOps turns most mistakes into a question for a person rather than a change to the system. Its wrongful actions are almost all step-ups or throttles, which lock nobody out. Those that cost anything are undone within ten minutes. Its total legitimate-request cost is the highest of the three, but almost none of it comes from mistakes. It is the cost of correct actions: when the agent suspends a taken-over account or an insider for a day, the account's genuine activity stops too. That cost is real, and Section VIII returns to it.

B. The trade-off

Fig. 2 plots the two outcomes that matter most against each other: attack activity stopped and innocent principals locked out.



App3 Results.Json · Five Test Weeks × 20 Runs Per Design

Figure 2. Trade-off between attack activity stopped and wrongful lockouts per day for rule-based SOAR, a single-agent responder, and AgenticSecOps, including three architectural ablations. Values represent simulation outcomes across five synthetic test weeks, with 100 runs per design. The horizontal axis uses a logarithmic scale; hollow markers represent ablations.

The dashed line marks the move from a single agent to AgenticSecOps: a step left by more than an order of magnitude on lockouts, and a step down of about 7 points on containment. Removing the Zero Trust checks puts the design back almost exactly where the single agent sits. Removing the threat-model context keeps the safety but loses a third of the containment.

C. Ablation

Table 5: Removing one component of AgenticSecOps at a time

Variant	Attack activity stopped	Wrongful lockouts / day	Wrongful soft actions / day	Injection success	Investigation median (min)	Human touches / day
AgenticSecOps (full)	34.7%	0.02	0.48	2.8%	9.7	7.6
− threat-model context	21.4%	0.04	0.00	1.9%	15.2	10.3
− agent coordination	35.8%	0.01	0.60	9.3%	7.5	6.7
− Zero Trust policy checks	41.8%	0.66	0.39	15.9%	3.0	4.3

The Zero Trust policy checks are what keep innocent users from being locked out. Without the evidence check, cost check and approval step, wrongful lockouts rise from 0.02 to 0.66 a day and injection success from 3% to 16%. They are also where the containment cost comes from: removing them stops 7 points more attack activity and makes decisions three times faster.

Agent coordination is the main defence against injection that aims at dismissal. The correlation agent never reads user-written fields, so it cannot be talked out of an alert. Removing it triples injection success, from 2.8% to 9.3%. It has little effect on containment.

Threat-model context matters a great deal for containment. Without it, the agent has no basis for anything smaller than revocation. Every response then needs approval, human touches rise by a third and containment falls from 35% to 21%. With it, most responses are step-ups and throttles that run at once. The context also explains the higher total cost of the full design. Over-reaching agents sometimes choose a 24-hour suspension, which the context-free variant never uses.

D. Robustness

Table 6: Sensitivity (wrongful lockouts per day / attack activity stopped)

Setting	B1 Single agent	B2 AgenticSecOps
Injection susceptibility 0	0.33 / 48.3%	0.01 / 39.4%
Injection susceptibility 0.3	0.69 / 41.6%	0.01 / 36.0%
Injection susceptibility 0.8	1.27 / 29.6%	0.01 / 28.5%
Verdict error 0.05	0.61 / 44.0%	0.00 / 37.1%
Verdict error 0.30	0.87 / 32.2%	0.02 / 26.0%

Within the tested parameter ranges, the simulated reduction in wrongful lockouts persists. However gullible or error-prone the agent, AgenticSecOps keeps wrongful lockouts near 0.01 a day, while the single agent's grow almost fourfold. Containment falls for both as agents get worse. At high injection susceptibility the two designs contain about the same amount, because injected dismissals cost the single agent the advantage its speed gave it. Changing the correlation between the two B2 agents' errors from 0 to 0.9 barely changed any result. Coordination helps against injection, not against shared reasoning errors.

VIII. DISCUSSION

A. What the results support

The results support one strong claim and one honest qualification. The claim: the proposed policy and approval architecture strongly influences simulated response harm under the specified agent-error assumptions. With the same error and injection rates, separating reading from acting, adding a correlation agent that cannot be injected, and requiring a person for any

lockout cut wrongful lockouts about 35-fold. This pattern persisted across the tested agent-error and injection-susceptibility settings. The qualification: in this application, safety cost containment. API attacks are short, a ten-minute approval is long, and step-ups and throttles do little against someone abusing their own account. AgenticSecOps stopped 35% of attack activity, against 42% for an unconstrained agent and 49% for rules that act in half a minute on anything strong.

The right choice therefore depends on what an organisation fears more. Where locking out a customer or a support agent is expensive, AgenticSecOps is clearly better than either baseline. Where the main fear is a fast attack running unchecked for ten minutes, rule-based automation remains hard to beat on the incidents it can see. The two can coexist: rules act instantly on the narrow set of strong, unambiguous signals, and agents investigate everything else.

B. Closing the containment gap

Two refinements follow directly from the results, though we have not evaluated them. First, revoking an anonymous client, such as a credential-stuffing source, locks out no real user, so the gate could let it run without approval. Second, approvals that humans grant consistently, for example revoking a session that failed a step-up, could move to the immediate tier once enough approval history exists. Both narrow the gap at a known cost in safety, and the ablation without Zero Trust checks shows the far end of that trade.

C. The cost of being right

AgenticSecOps blocked the most legitimate requests in total, almost all of them through correct actions. A taken-over account suspended for a day stops its real owner too, and a suspended insider stops their legitimate work. This cost is invisible in most evaluations of automated response, which count only mistakes. It argues for preferring revocation plus a forced credential reset over day-long suspension when an account is compromised, and for measuring the cost of correct actions as routinely as the cost of wrong ones.

D. Limitations

- No language model was run. Agent errors and injection susceptibility are parameters. Real models make correlated, context-dependent mistakes that a single correlation parameter only approximates, and real injection rates vary widely by model and prompt design [15], [16].
- Synthetic application traffic. The incidents and request streams come from the traffic model of [23], with its limitations, although every cost and benefit is counted from those streams rather than assumed.
- Simple human model. Approvers have fixed accuracy and no fatigue. At 7.6 decisions a day this is plausible; at hundreds it would not be.
- No detector retraining. Missed incidents stay missed. A real system would feed confirmed incidents back into detection.
- One application. As in the two earlier papers.

E. Validating with real agents

A real-agent validation would require implementing the agent roles, tool interfaces, prompt-injection tests, and approval workflows; the present simulation does not establish that those implementations behave as assumed. Implement the five roles with a chosen model, connect them to the decision logs and to read-only and gated tools, replay the test weeks of [23] with injection payloads placed in profile names, order notes and support tickets, and record the same metrics. Running the agents in shadow mode, proposing but not executing, against live traffic would give the measured error and injection rates that this paper had to assume. Those measured rates can then be fed back into the simulation to check its predictions.

Prospective validation protocol (not performed). Measure real-agent verdict errors, successful prompt injections, token usage, decision latency, approval outcomes, and rollback failures on the same incident episodes; compare designs using paired uncertainty intervals. No values are reported because this experiment has not been performed.

IX. CONCLUSION

This paper completes a study of one microservice application at three levels: authorization derived from a threat model [22], detection and graded response for abuse the policy allows [23], and now AI agents that investigate and respond. Delegating incident response to agents requires explicit safeguards against mistaken or manipulated actions. AgenticSecOps is designed to reduce this risk by giving no single agent both the ability to read attacker-written text and the ability to act, by binding every action to evidence, by pricing every action before a person approves anything that locks someone out, and by making every action reversible. On generated request streams spanning five simulated test weeks, with agents that were wrong 10% of the time and followed injected instructions 30% of the time, it cut wrongful lockouts from 0.68 to 0.02 a day and injection success

from 30% to 3%, at a cost of 7 points of containment. The ablations show that the Zero Trust policy checks provide the safety, coordination provides the injection resistance, and the threat-model context provides most of the containment. The simulation design provides a basis for repeating these comparisons with real agents, subject to independent reproduction and release of the implementation.

X. REFERENCES

[1] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," NIST Special Publication 800-207, Aug. 2020.

[2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. International Conference on Learning Representations (ICLR), 2023.

[3] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," arXiv:2308.08155, 2023.

[4] Anthropic, "Introducing the Model Context Protocol," Nov. 2024.

[5] G. Deng et al., "PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing," in Proc. 33rd USENIX Security Symposium, 2024.

[6] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), 2023.

[7] OWASP Foundation, "OWASP Top 10 for LLM Applications 2025," Nov. 2024.

[8] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, Jul. 2024.

[9] R. Fang, R. Bindu, A. Gupta, and D. Kang, "LLM Agents Can Autonomously Exploit One-day Vulnerabilities," arXiv:2404.08144, 2024.

[10] Open Policy Agent Project, openpolicyagent.org.

[11] SPIFFE Project, "SPIFFE and SPIRE," spiffe.io.

[12] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms," in Proc. 31st USENIX Security Symposium, 2022.

[13] OWASP Foundation, "OWASP API Security Top 10 – 2023," owasp.org.

[14] D. Hardt, Ed., "The OAuth 2.0 Authorization Framework," RFC 6749, IETF, Oct. 2012.

[15] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," in Proc. NeurIPS Datasets and Benchmarks Track, 2024.

[16] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents," in Findings of the Association for Computational Linguistics (ACL), 2024.

[17] E. Debenedetti et al., "Defeating Prompt Injections by Design," arXiv:2503.18813, 2025.

[18] V. Bertocci and B. Campbell, "OAuth 2.0 Step Up Authentication Challenge Protocol," RFC 9470, IETF, Sep. 2023.

[19] T. Lodderstedt, Ed., S. Dronia, and M. Scurtescu, "OAuth 2.0 Token Revocation," RFC 7009, IETF, Aug. 2013.

[20] OpenID Foundation, "OpenID Continuous Access Evaluation Profile 1.0," Shared Signals Working Group.

[21] W. U. Hassan et al., "NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage," in Proc. Network and Distributed System Security Symposium (NDSS), 2019.

[22] R. Balaji, "Threat-Model-Driven Zero-Trust Authorization for Microservice APIs in Kubernetes-Based Cloud-Native Applications," manuscript in preparation.

[23] R. Balaji, "Request-Level Anomaly Detection and Risk-Adaptive Authorization for Microservice APIs," manuscript in preparation.

[24] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278-1308, 1975.

[25] National Institute of Standards and Technology, "Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile," NIST Special Publication 800-61 Rev. 3, Apr. 2025.