Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Closed-Loop Containment Assurance in Public Clouds: Compromise-Domain-Aware Enforcer Selection and Scoped Isolation Verification

Kwangmo Kim

kwangmo@soongsil.ac.kr

Soongsil University

Research Article

Keywords: Zero trust, Containment, Policy enforcement point, Closed-loop security management, Isolation verification, Cloud networks

Posted Date: November 17th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11754741/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Closed-Loop Containment Assurance in Public Clouds: Compromise-Domain-Aware Enforcer Selection and Scoped Isolation Verification

Kwangmo Kim

Department of Financial Technology Convergence, Graduate School, Soongsil University, Seoul, Republic of Korea

Corresponding author: Kwangmo Kim, e-mail: kwangmo@soongsil.ac.kr

Abstract

Containing a compromised workload is a management action in zero trust clouds, usually recorded as complete once the interface acknowledges the request. The acknowledgement does not show that traffic stopped, and the workload may revert the policy that isolates it. Live measurements confirmed that, as documented, tracked flows survive an isolation security group; a configuration analysis reported no path, and flow records at one interface did not separate two filtering layers. We propose a containment loop built on one compromise model computed from placement, management planes and access-policy reachability; the evaluated controller follows role assumption only. The model decides which evidence may trigger containment, which enforcers the target cannot revert, and which observations may confirm the result. An incident ends EFFECTIVE for a stated scope within the virtual network, or UNRESOLVED; disruptive actions await approval. We evaluated the loop in an emulator of the measured semantics on 10,000 triggered incidents with assumed frequencies. With every established flow tracked and no credentials revoked, containment that trusted the acknowledgement was reported falsely in 75.6% of incidents with a security group alone and in 34.0% with access control lists added, both within one standard error of closed-form expectations. Verification without enforcer selection gave 5.75%. The loop gave 0.12%, only when an injected access-analysis miss coincided with late reversion, which the emulator admits by construction. The price was a median of 11 minutes per declaration and an instance stop in 30% of incidents; in one integrated set, the trigger missed 15% of real bypasses.

Keywords Zero trust · Containment · Policy enforcement point · Closed-loop security management · Isolation verification · Cloud networks

1 Introduction

Zero trust architectures place policy enforcement points close to each workload [1, 2], and incident response guidance treats containment of a compromised workload as a core activity [3]. In public clouds the isolation is a management action: an orchestration playbook or an operator replaces the workload's security group, adds deny entries to network access control lists (network ACLs), or asks a guest agent to cut network access [3, 4]. The provider's own incident response service automates the first of these and, as a separate action, contains an access-management principal by attaching a deny-all policy [5]; this article studies the network actions and treats credential revocation as complementary (Sects. 10.5 and 11). Recent surveys of zero trust describe enforcement points and their placement [6, 7]; whether a containment action applied through them actually holds is the question this article addresses.

The management problem studied here starts after the decision to contain has been taken. The controller issues the change, the interface answers with success, and the incident record is marked contained. Three things can make that record wrong. First, the acknowledgement confirms that a request was accepted, not that packets stopped: a stateful security group keeps forwarding connections it already tracks [8], and a network ACL does not filter traffic inside a subnet. Second, the enforcement point may be under the adversary's control. A guest agent runs inside the compromised workload, and disabling or modifying security tools is a documented adversary technique [9]; less obviously, a security group can be changed back by any principal that the workload's role can reach through role assumption, or by the management server that also manages the workload's agent. Third, the evidence used to confirm containment is delayed, incomplete and ambiguous. Flow log records (flow records in the following) arrive minutes later and are delivered on a best-effort basis [10]; silence after a probe may mean a block, an outage, or a record that has not arrived; and a rejection record does not name the layer that rejected the packet.

An operator who trusts a wrong record stops watching a workload that is still exchanging traffic. We call this outcome a false-effective report and treat its rate as the primary quantity to manage, next to the exposure that remains, the time to a decision, and the number of disruptive actions and approvals that the process costs.

Our approach has two parts. We first measure, on a live system, what an operator can conclude from the provider's acknowledgement, from its configuration analysis, from active probes and from flow records. We then build the loop on one observation: whether a piece of

evidence, an enforcer, or a verifier can be believed depends on the same question, namely whether the compromise that affects the target could also manipulate this component. We answer the question once, from infrastructure metadata, and reuse the answer at each step of a monitor, analyse, plan and execute loop [11] that we extend with a verification step. The contributions are as follows.

1. **Measurements of what the provider's signals establish.** The provider documents that changing a security group does not end the connections that the group tracks [4, 5, 8]. Three live runs in a virtual private cloud, one of them with five repetitions on two processor architectures, confirmed this for TCP, UDP and ICMP flows. They also showed what a verifier has to know beyond the documented behaviour: removing stateless deny entries restored tracked TCP connections; a route change blocked traffic without leaving a rejection record; a configuration analysis reported no path while established exchanges continued; rejections by a security group and by a network ACL in the target's subnet were recorded identically at the target's interface; and changes took effect and records arrived with delays that a verifier must wait for (Sect. 4).
2. **A compromise-domain model for management decisions.** Each enforcement or observation point receives an attribute vector (execution environment, node, management plane, control principals, account) obtained only from infrastructure interfaces. A deterministic relation share(p, q) states whether one compromise of an execution environment, a node or a control principal can reach both points; unknown attributes count as shared, and management planes and accounts are handled separately (Sect. 5). Dependency data have been used before to audit whether redundant deployments are independent with respect to common failures (Sect. 2.3); the relation here asks instead what one compromise that includes the target can control. The evaluated implementation follows role assumption only; the further escalation rules and the account rule of Sect. 5 are specified but not evaluated.
3. **A containment loop that uses the model three times, with stated conditions of correctness.** An independence-aware trigger counts independent groups of evidence rather than witnesses, a principle proposed for quorums of software agents that we apply to infrastructure fault domains (Sects. 2.3 and 6); enforcer selection keeps only enforcers whose policy the target cannot revert and reports UNRESOLVED when none exists (Sect. 7); verification accepts probes and records only from independent

verifiers and observers, never reads silence as success, counts a rejection only when the configuration attributes it to a selected enforcer, and declares effectiveness for an explicit scope (Sect. 8). Disruptive actions are gated by operator approval (Sect. 9). Three propositions state what a single compromise cannot do, each with its assumptions, and the evaluation includes scenarios in which the assumptions do not hold.

4. **An evaluation under assumed conditions.** An emulator that encodes the measured semantics compares the loop with five configurations that reflect current practice and two ablations, on five paired sets of random incidents and 17 targeted scenarios, starting from a triggered target, and adds a run from trigger to declaration. It reports, next to false-effective reports, the scope that each declaration covers, the time to a declaration, approvals and instance stops (Sect. 10). For the configurations that accept an acknowledgement, the emulated rates lie within one Monte Carlo standard error of closed-form expectations under the assumed conditions; what the emulator adds is the interaction of reversion, verification, approval and record timing in the other configurations.

The loop is an assurance mechanism for a management action, not an intrusion detector. It does not compete with learning-based detection, which decides whether a workload is suspicious; it governs what happens after that decision and can take its trigger from any detector. Section 2 reviews related work, Sect. 3 formulates the problem, Sect. 4 reports the live measurements, Sects. 5–9 present the method, Sect. 10 reports the evaluation, and Sect. 11 discusses the limitations and the work that remains.

2 Related Work

2.1 Zero trust and response management

The zero trust reference architecture separates policy decision from policy enforcement and notes that both may be attacked [1]; its cloud-native refinement places enforcement in sidecars and gateways [2]. The systematic review by Gambo and Almulhem in this journal and the survey by Syed et al. classify zero trust work by component [6, 7]. Incident response guidance lists containment among the core activities [3]. Surveys of intrusion response and countermeasure selection frame response as an optimisation of cost and benefit over a set of candidate actions [12, 13], and security orchestration platforms execute the chosen playbooks

[14]; Huang et al. propose expressing response as intents over a defensive ontology [15]. Jahan et al. couple the monitor, analyse, plan and execute loop of an adaptive system with a loop over security assurance cases, so that adaptations are checked against security requirements [16]; the assurance there concerns the adapted system, not whether a containment action holds against an adversary that controls part of that system. These lines of work address the selection, orchestration and assessment of responses; Table 1 separates these objectives from verifying, from traffic, that a cloud isolation holds against an adversary able to revert it.

Several recent articles in this journal manage security functions in ways that border on ours. Robles-Enciso et al. allocate security functions to nodes under security constraints in intent-based orchestration [17]; the placement problem resembles our enforcer selection, but its placement constraints concern node resources, location, latency and anti-affinity, not independence from a compromised target. Pizzato et al. translate isolation intents into Kubernetes network policies and check the compatibility of consumer requests with provider authorisations before the intents are harmonised and the policies generated [18]. The check concerns intents before enforcement, and neither verifying from traffic that an isolation holds in a given incident nor an enforcer under adversarial control is within its scope. The same group closes the loop from detection to response: a continuously running process turns alerts from monitoring agents into security policies and reconfigures distributed firewalls in a provably correct and optimised way [19]. Its guarantee concerns the computed configuration; verifying from traffic that the reconfiguration holds against an adversary that can change the firewall back is not among its stated objectives. Khozam et al. mitigate denial-of-service attacks in software-defined networks with a module outside the controller and evaluate the quality of service of legitimate traffic [20]; Khanal et al. embed detection and mitigation in the controller and evaluate its overhead [21]; and Filho et al. adapt detection to infrastructure that changes with each deployment [22]. CUSTOS delays destructive containment until volatile evidence has been captured [23]; it governs when containment may proceed, not whether it holds.

2.2 Cross-view monitoring and attestation

Comparing an untrusted inside view with a more trusted outside view is a long-standing idea. Cross-view differencing exposes hidden files and processes [24], and virtual machine introspection moves the monitor below the guest [25, 26]. Remote attestation provides

integrity evidence rooted in hardware [27, 28]. These techniques assume that the outside view is independent of the compromise. In a public cloud the tenant cannot place monitors in the hypervisor, and observers that look independent often share a node, a credential or a management server with the target. Our model states the independence assumption explicitly and checks it.

2.3 Independence of evidence sources

Byzantine agreement protocols tolerate a bounded number of arbitrary faults [29, 30], and common-mode failures [31] can make more replicas fail together than the protocol tolerates. Work on design diversity models how separately developed versions can still fail coincidentally [32], and Garcia et al. quantify how operating-system diversity reduces common vulnerabilities among replicas [33]. INDaaS collects network, hardware and software dependency data to audit, before failures occur, whether redundant systems believed independent share hidden dependencies, and in its case study two replicas are redeployed as the audit recommends [34]. It is the closest precedent for deriving independence from infrastructure data and acting on the result. Its subject is the correlated failure of redundant deployments, and its trust assumptions concern the confidentiality of the providers' dependency data. We put a different question to similar data, namely whether one compromise that includes the target can change an enforcer's policy or forge an observer's records, and we use the answer during an incident to admit or exclude evidence, enforcers and verifiers. Collaborative intrusion detection systems aggregate alerts from many monitors, and some of the systems surveyed in [35] weigh the monitors by trust to counter malicious insiders. He and Yu argue that approval quorums of software agents should count fault-separated evidence paths rather than voters, and propose a controller that enforces a structural measure of such separation when a quorum is admitted [36]. In a second article the same authors certify AI-generated infrastructure changes before they are executed, by aggregating the judgements of a diverse panel of validators under a risk-adaptive quorum rule that enforces model diversity, together with a model of correlated validator failures [37]. In a third they extend Byzantine fault tolerance with budgets for correlated invalid endorsements [38]. The three articles address different parts of the approval problem. In all of them fault domains arise from shared models, data and inputs, the decision certified is whether a proposed change may be executed, and the component that executes an approved change is trusted. Our trigger applies the principle of counting fault-separated evidence, which is theirs and not a contribution of this article, to evidence about enforcement; treating missing information

conservatively is likewise common practice and is not claimed as new. The differences are that fault domains are derived from where a point runs, who manages it and which principals can change it, and that the same domains are used again after the decision, to choose enforcers and verifiers, so that the component that carries out containment may itself be excluded.

2.4 Verifying network policy and its enforcement

Verification methods for data planes and configurations check whether forwarding state satisfies reachability or isolation invariants [39–41], including networks with stateful middleboxes [42]. TenantGuard verifies isolation between virtual machines across a cloud at runtime from collected configuration and re-verifies incrementally when the configuration changes; its authors state that the correctness of the result depends on the input data extracted from logs and databases, and name ensuring the integrity of that input as a limitation (Sects. III-A and VIII of [43]). Automated reasoning is used in production to analyse network reachability [44] and access policies [45] of the provider we use as an example. Open-source and research tools compute privilege-escalation paths between the principals of an account, as a directed graph of principals [46] or by combining analysis of the full configuration with targeted questions when it is only partly known [47]. CloudLens models cloud access-control policies and uses a planner to generate attacks that expose misconfigurations, evaluated on 14 real configurations of the same provider [48]. The network verifiers answer whether a configuration, if enforced as written, yields isolation. They reason over configuration, and TenantGuard also over its changes, but they do not observe traffic, so connections that a stateful enforcer still tracks and a rule that was acknowledged but not applied are outside their view; Sect. 4.3 shows one such analysis next to live traffic. That a rule can be acknowledged before it is active is documented for hardware switches [49], and Monocle probes the data plane to check the rules for which it can generate a distinguishing probe [50]. P4Consist injects active probe traffic into programmable networks and compares independent reports of the control plane and the data plane to detect inconsistencies at runtime [51]; the reporting components are independent by design, and their independence is not derived from the permissions that a compromise could use. Research on the data plane of software-defined networks has also dropped the assumption that the enforcing devices are honest. SDNsec lets a trusted controller validate, from proofs that switches embed in packets, the path that traffic actually took when switches are compromised [52], and FlowCloak detects compromised switches that let traffic bypass a middlebox, by verifying packet tags in the data plane [53]. Both verify forwarding against a compromised enforcer, with a controller that is trusted and

with proofs carried in packets; neither derives from the permissions of a cloud account which enforcer or observer a compromised workload could influence. Intent-based networking draws the same line for intents: its assurance function compares the observed behaviour of the network with the intent and, when they diverge, triggers corrective action or informs the operator, and its security considerations ask, at the level of principles, that the intent-based system itself be protected [54]. eBPF-Auditor runs on each node, independently of the agent of the network plugin, and compares the state that a cluster's control plane intends with the kernel's data-plane maps to detect silent divergence caused by faults; it compares state and sends no traffic [55]. Neither specifies how to choose an enforcer or a checking component that an adversary in control of the target cannot influence. Operational practice also goes beyond the acknowledgement in places. A patented method judges a network isolation device by sending traffic towards it and observing whether the traffic appears behind the device [56], and the provider's containment runbook replaces an instance's security groups in two steps, through a temporary group that allows all inbound traffic [57]. Checking after enforcement, with or without test traffic, is therefore not new. For cloud containment we add the two questions that these approaches leave open: which enforcers, probers and observers a compromise of the target can control, and which enforcer a recorded rejection belongs to.

2.5 Gap

Table 1 summarises the comparison. The elements of our approach have precedents. Independence has been audited from infrastructure data for failures, and deployments have been changed accordingly [34]; firewalls have been reconfigured in a loop from detection [19]; evidence has been counted by fault domain [36]; and access-policy reachability has been analysed for attack paths [45, 48]. The effect of a rule or an intent has been checked after enforcement [50, 51, 54, 55], and forwarding has been verified against compromised switches [52, 53]. What we did not find in the work we reviewed is their combination for containment. In that combination one relation, computed from what a compromise of the target can control, decides which evidence is admitted, which enforcers are relied on and which probes and records are accepted, and the outcome is declared for the scope that the accepted observations support. The combination matters because it changes management decisions. A design that judged the three steps separately could give each node one vote, trust any enforcer that runs outside the target, and accept any rejection that a trusted observer records. In our evaluation each of these answers is wrong in at least one case: two alarms that trace back to one credential (case c in Sect. 10.4), a security group that the target's role can change back after

verification (scenario S4L in Sect. 10.5), and a rejection produced by a second layer that the target controls (scenario S13). Our search covered the publisher sites of this journal and of the venues cited above and public preprint servers; it was not an exhaustive query of subscription indexes, and we state the claim with that reservation.

Table 1 Related approaches and the question each leaves open for containment management

Approach	Management question addressed	Method	Open for containment
Response selection and orchestration [12–15, 17]	Which action or placement is best	Cost models, intents, optimisation	Traffic-level verification of the outcome and compromised enforcers outside the stated objectives
Intent-driven isolation [18]	Translate isolation intent to policy	Intent refinement, compatibility check of intents before policy generation	Intents checked before enforcement; incident-specific verification from traffic and compromised enforcers out of scope
Closed-loop firewall reconfiguration [19]	Mitigate a detected attack by reconfiguring firewalls	Policy extraction from alerts, provably correct configuration synthesis	Correctness of the computed configuration; traffic-level verification against an adversary that reverts the enforcer not among the objectives
SDN attack mitigation [20, 21]	Mitigate while preserving service	Learning-based control in SDN	Controller and switches trusted
Configuration and data-plane verification [39–44]	Does the configuration imply isolation	Formal models of forwarding and policy	Reason over configuration, not traffic, so flows that a stateful enforcer still tracks are not seen; integrity of the collected input left to future work in [43]
Access-policy reasoning [45]	Which requests an access policy allows	SMT encoding of access policies	Answers who may change an enforcer, not whether traffic stops
Data-plane probing [50, 51]	Is an acknowledged rule active, do control and data plane agree	Generated probe packets, for rules that admit a distinguishing probe; probe traffic with comparison of control-plane and data-plane reports	Whether a compromise of the target controls the prober or the enforcer is not modelled
Forwarding verification against compromised switches [52, 53]	Did traffic follow the intended path or middlebox chain	Proofs or tags carried in packets, checked by a trusted controller or egress switch	Controller trusted; enforcers and observers not selected from cloud permissions
Intent assurance and state audits [54, 55]	Does the deployed state still match the intent	Continuous compliance assessment; comparison of control-plane and data-plane state	No procedure for choosing an enforcer or checker outside the adversary's control; [55] audits state for faults, without traffic
Adaptive systems with assurance cases [16]	Do adaptations still meet security requirements	Feedback loop over security assurance cases	Assures the adapted system's requirements; containment against an adversary that controls the enforcer not addressed
Access-policy attack planning [48]	Which attack paths misconfigured policies admit	Relational model of policies, planning	Finds reachability; does not select enforcers or verify traffic
Independence auditing [34]	Do redundant systems share hidden dependencies	Dependency data, fault-graph analysis	Common failure causes rather than control by a compromise; used to choose deployments, not containment enforcers or verifiers
Operating-system diversity [33]	How many vulnerabilities replicas share	Analysis of vulnerability data	Not applied to enforcement points or to action selection
Independence-aware quorums [36–38]	How many fault-separated evidence paths or diverse validators support an approval	Structural fault domains, diversity-enforcing validator quorums	Fault domains from models, data and inputs; approval certified before execution; executing component trusted

Approach	Management question addressed	Method	Open for containment
This work	Is the workload contained, within which scope, and can the report be believed	One compromise model reused for trigger, selection and verification	See Sect. 11

3 Problem Formulation

3.1 Setting and roles

A tenant runs workloads in a protected cloud account A. Enforcement points act on a workload’s traffic: a guest agent inside the workload, node-level enforcers, the provider’s security groups and network ACLs, and, as a last resort, stopping the instance. We call any mechanism that applies a containment action an enforcer: the guest agent, security groups and network ACLs are policy enforcement points in the sense of [1], whereas stopping the instance is a disruptive action. Flow logs observe traffic without enforcing. A containment controller runs in a separate assurance account B and reaches account A only through narrowly scoped cross-account roles: read access to flow logs, inventory, network configuration and access policies, write access limited to isolation changes, and a separate role for disruptive actions that can be assumed only after approval. Figure 1 shows the components and the loop.

Two human roles are involved. The security operator receives the outcome of each incident and decides on requests for disruptive actions. The platform administrator owns the permission structure of account A, which determines how often an independent enforcer exists. The controller automates everything else.

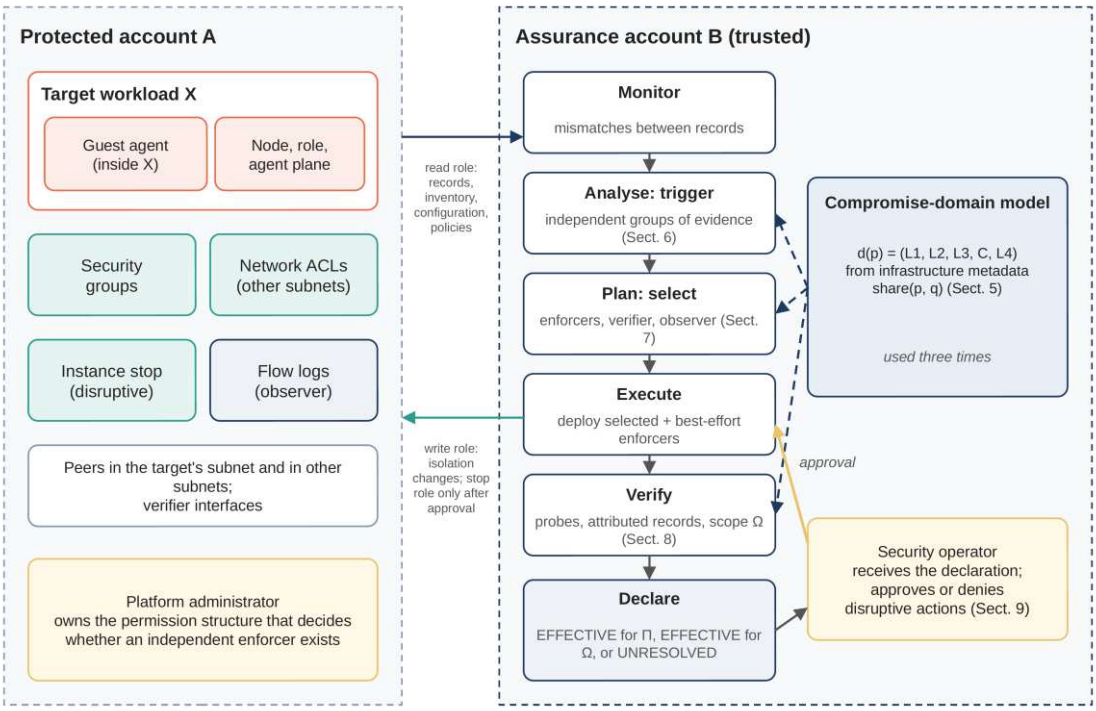


Fig. 1 Containment loop. One compromise model, computed from infrastructure metadata, constrains the evidence that may trigger containment, the enforcers that may be relied on, and the verifiers and observers that may confirm the result

3.2 Threat model and trusted base

The adversary controls the target workload X: its execution environment, its host node if the adversary has escaped the execution environment, every principal reachable from X's credentials by role assumption or by the escalation paths of Sect. 5, and possibly the credentials of the management plane that manages X's agent. It can therefore silence or forge the records of every point that shares one of these with X, change the policy of any enforcer that one of these principals may modify, do so immediately or after a delay of its choosing, and keep quiet while it believes it is being observed. Independently, an adversary may control other nodes, credentials or one management plane and use them to report mismatches that never happened, in order to have a healthy workload contained.

The provider's control plane, its flow-log service and account B are trusted. One compromise event is assumed: an adversary who separately compromises two management planes, or a plane and an unrelated node, is outside the model, and Sect. 10.4 shows that the trigger then fails. Denial of service against the controller and side channels are out of scope.

3.3 Decisions, objectives and constraints

For an incident with target X, let Π be the set of path classes over which X can communicate. A path class is a triple (side, family, kind): the peer is in X's subnet or in another subnet, the address family is IPv4 or IPv6, and the connection or flow, of any protocol, is new or already established when containment starts. Π contains only paths whose peer is inside the virtual network. Traffic that leaves it, through an internet gateway, an egress-only gateway, a network address translation gateway, a gateway endpoint, peering, a transit gateway or a virtual private network, is outside Π in this article: the enforcers of Sect. 4.1 do not cut established flows on such paths, no verifier is placed outside the network, and neither the live measurements nor the emulator include such paths. An EFFECTIVE declaration, including one for all of Π, therefore says nothing about them, and the declaration has to name them as not covered. Path classes do not distinguish ports or protocols because the enforcers the loop relies on deny all traffic of X's addresses regardless of port and protocol (Sect. 4.1), apart from an explicit allow list for management and recovery. Under such an enforcer one rejected new connection is representative of the other ports of its class, provided that every path of the class is governed by the same rules. This is an assumption about the configuration snapshot, not something a probe establishes: an EFFECTIVE class means that sampled probes of the class were rejected, and it extends to the other ports and protocols of the class only through that assumption, which the emulator satisfies by construction and which we have not tested against a configuration that violates it. The controller therefore splits a class whenever the configuration snapshot shows different rules for its paths, for example peers in two other subnets whose network ACLs differ, and treats each part as a class of its own. An enforcer with port- or protocol-specific rules would require a finer partition, which we do not evaluate. Let E be the available enforcers, each of which blocks a known subset of Π by construction (Sect. 4.1). The controller decides:

- whether the available evidence justifies containment of X (trigger);
- which enforcers $S \subseteq E$ to rely on, and which to apply only as best effort;
- which verifier V and observer O to believe;
- what to declare: EFFECTIVE for a scope $\Omega \subseteq \Pi$, or UNRESOLVED;
- whether to ask the operator to approve a disruptive action.

Let open(p, t) denote that traffic of path class p would be delivered at time t, that is, that no enforcer blocks the class then, whether or not X sends traffic; an adversary that stays silent does not thereby close a path. For a declaration made at time t_d and a horizon H, the objectives are, in order of priority:

$$\min \Pr[\,\mathrm{EFFECTIVE}(\Omega) \wedge \exists\, p \in \Omega,\ t \in [t_d + \delta,\ t_d + H] : \mathrm{open}(p, t)\,] \qquad (1)$$

$$\min \Pr[\,\exists\, p \in \Pi : \mathrm{open}(p,\ t_d + H)\,] \qquad (2)$$

$$\min\ (\,t_d,\ \text{approvals},\ \text{disruptive actions}\,) \qquad (3)$$

Objective (1) is the false-effective rate, (2) the residual exposure, and (3) the operational cost. The offset δ excludes the time that a change needs to take effect after the declaration; the evaluation uses δ = 60 s and H = 1 h (Sect. 10.3), so a declared path that is open only within δ of the declaration is not counted, and no result in this article covers that interval. The order is deliberate: a report that says UNRESOLVED or names a narrow scope costs operator attention, whereas a wrong EFFECTIVE removes it. The decisions are subject to four constraints. (C1) Disruptive actions require approval, which may be denied or time out. (C2) The controller observes traffic only through its own probes and through records that are aggregated per interval, delayed, occasionally missing, and silent about which layer rejected a packet. (C3) Attribute vectors may be incomplete, and the analysis of access policies may be imprecise. (C4) The adversary of Sect. 3.2 can modify any enforcer within its reach at any time.

3.4 Research questions

- **RQ1.** How often does current practice report containment that does not hold, and which provider semantics cause it?
- **RQ2.** Which part of the reduction comes from selecting independent enforcers and which from verification?
- **RQ3.** What does the loop cost the operator in decision time, approvals and stopped instances, and how does the cost depend on the permission structure that the platform administrator controls?
- **RQ4.** Can the trigger be made resistant to fabricated evidence without losing real detections, and what does it cost to compute at scale?

4 Provider Semantics and Observability: Live Measurements

The design of the loop depends on what the provider’s mechanisms block and on what its signals show. We therefore measured both before fixing the method. This section answers the second half of RQ1.

4.1 Mapping to a public cloud

We map the setting to Amazon Web Services. The isolation security group has neither inbound nor outbound rules and replaces the groups of X’s network interface. Deny entries for X’s address are added, with a lower rule number than existing entries, to the network ACLs of the other subnets, for IPv4 and for IPv6 where X has such an address. Flow logs are configured with a maximum aggregation interval of one minute and a custom format that includes the interface, the action, the log status and the TCP flags, and are delivered to storage owned by account B. The provider’s incident response guide and its managed containment action both state that changing security groups does not shut down tracked connections [4, 5], and the flow-log documentation gives a typical delivery time of about ten minutes to object storage on a best-effort basis [10]. The provider further documents that untracked flows are interrupted as soon as the rule that allows them is removed and that a network ACL, being stateless, breaks existing connections [8]. These statements are the documented part of Table 2, which lists the semantics that the loop depends on. The rest of this section reports how far we confirmed them on a live system and what we observed beyond them: which protocols and timeouts were affected in our setting, what happens when deny entries are removed, what a routed drop leaves in the records, how the two filtering layers appear in flow records, and how long changes and records take. The runbook behind the managed action replaces the groups in two steps, through a temporary group that allows all inbound traffic [57]; its documentation gives no reason for the intermediate step, and the description of the managed action states that only future traffic is blocked [5]. We measured a single replacement and did not measure whether the intermediate step ends connections that are already tracked (Sect. 11). The traffic generators and analysis scripts were drafted with a generative AI assistant (Claude Opus 5.5, Anthropic), which also ran them through the provider’s scripting interface; scripts, phase timestamps and outputs are available as stated under Data availability.

Table 2 Enforcement semantics used by the loop. The last column refers to the live measurements of this section

Means	Blocks	Does not block	Can be modified by	Checked live
Guest agent	All traffic of X while the agent is intact (not measured)	Anything once the agent is impaired	Code in X's environment; management plane	No
Isolation security group	New connections, both families, any subnet	Tracked TCP, UDP and ICMP flows not idle beyond the interface timeout	Principals allowed to change groups or interfaces	Yes (Sects. 4.2–4.4)
Network ACL deny entries	New and established traffic to and from other subnets, per family	Traffic inside X's subnet; removing the entries restores tracked TCP connections whose tracking entry has not expired (not observed with a 60 s TCP timeout)	Principals allowed to change the ACL's entries or to associate its subnets with another ACL [58]; the entries match X's addresses, so principals allowed to add addresses or interfaces to X can move traffic outside them (Sect. 7)	Yes (Sects. 4.2–4.4)
Route to an unattached interface	Traffic between X's subnet and the routed prefixes, per family	Traffic inside X's subnet; leaves no REJECT record; affects every workload in the subnet	Principals allowed to change route tables	Yes (Sect. 4.3)
Instance stop	Everything	—	Principals allowed to start instances	Yes (Sect. 4.2)
Flow log (observer)	—	—	Principals allowed to delete or change the flow log, account administration of A among them; records already delivered to B are out of reach	Delivery delay (Sects. 4.2–4.5); layer attribution only where no other layer filters at the recording interface (Sect. 4.4)

4.2 Initial run: acknowledgement, tracked connections and record delivery

On 10 November 2026, in one virtual private cloud in the Seoul region, we ran three t3.micro instances without internet access: the target X and a peer P2 in one subnet, and a peer P1 in a second subnet. The working security group allowed TCP ports 5000–5100 within the cloud in both directions, with specific rules so that connections are tracked [8]. Four long-lived TCP connections between X and the peers exchanged one byte in each direction every 5 s, a fifth every 240 s to represent an idle connection, and three generators opened a new connection every 10 s. Flow logs were delivered to object storage. After a baseline period we (1) replaced X's security group by the isolation security group, (2) 13.7 min later added deny entries for X's address to the network ACL of P1's subnet, and (3) 8.5 min later stopped X. Outcomes were taken from application logs on the instances and, independently, from 1,861 flow records in 18 delivered objects.

Table 3 Initial run: application-level exchanges that succeeded / failed in each phase

Traffic	Baseline	Isolation security group	plus ACL deny entries	Instance stopped
Established, other subnet (3 connections)	267 / 0	291 / 0	0 / 88	0 / 114
Established, same subnet (2 connections)	250 / 0	282 / 0	156 / 0	0 / 144
New connections (3 generators)	194 / 1	0 / 163	0 / 90	0 / 111

Table 3 and Fig. 2a give the application view; counts after the stop come only from the peers, since X no longer logs. Four observations matter for the loop.

- **The acknowledgement says nothing about established connections.** The interface acknowledged the security-group change within two seconds. From then on all 163 new connection attempts failed, and the flow log recorded 172 rejections and no acceptance of new connections at X's interface. All 5 established connections continued without a single failed exchange (573 exchanges), including the idle one, and the flow log recorded their traffic as accepted (88 records). The single failed attempt in the baseline occurred while the instances were starting.
- **Stateless entries cut established connections, but only across subnets.** After the deny entries were added, the established connections to the other subnet stopped (0 successful and 88 failed exchanges), while those inside X's subnet continued unaffected (156 successful, 0 failed) until X was stopped.
- **An acceptance at the target's interface is not delivery.** Packets from P1 to X were now recorded as rejected at P1's interface (41 records) and no longer appeared at X's interface (0 records), whereas attempts from inside X's subnet were still rejected at X's interface. In this topology the two filtering locations therefore produced records at different interfaces. In the opposite direction, retransmissions that X sent on a cut connection were still recorded as accepted at X's own interface and as rejected at P1's.
- **Records arrived faster than documented, in five-minute batches.** Objects were written every five minutes. Over 1,842 records the delay between the end of a record's interval and its delivery ranged from 27 to 341 s, with a median of 179 s and a 95th percentile of 306 s (Fig. 2b), against a documented typical value of about ten minutes [10]. No interval was marked as skipped in this run. After the stop no record was produced for X's interface.

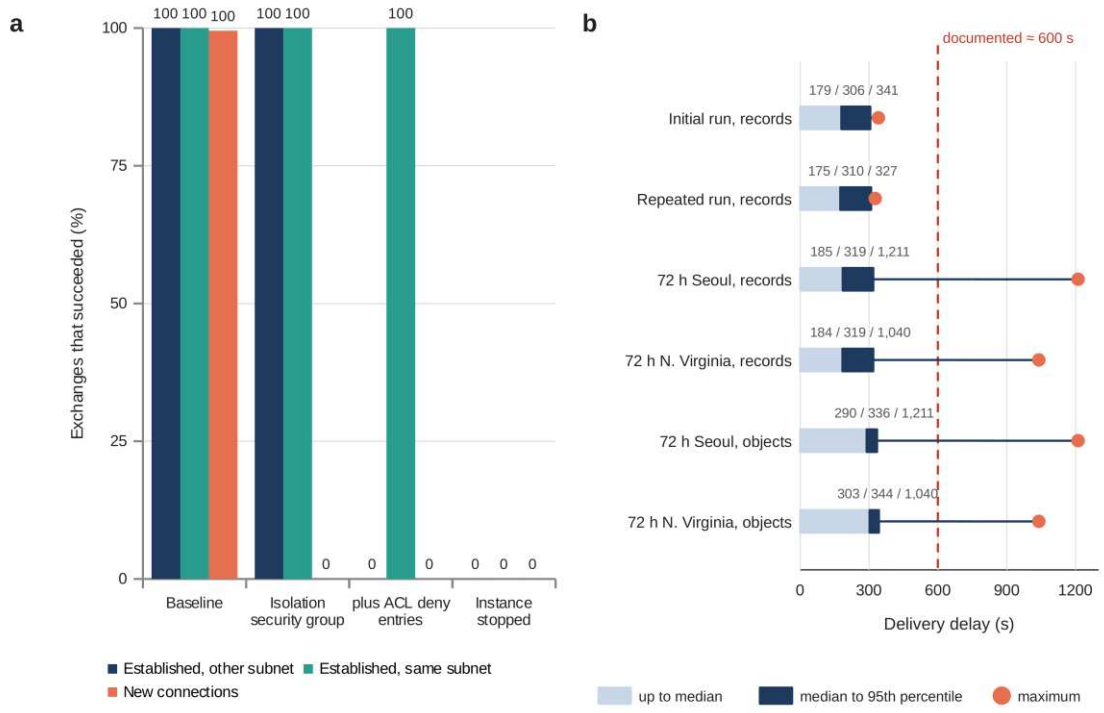


Fig. 2 Live measurements. a Share of application-level exchanges that succeeded in each phase of the initial run (Table 3). b Delay between the end of a record's aggregation interval and its delivery: median, 95th percentile and maximum in the initial run (Sect. 4.2), the repeated run (Sect. 4.3) and the 72-hour measurement in two regions (Sect. 4.5), per record and, for the 72-hour measurement, per delivered object; the dashed line marks the documented typical value of about ten minutes [10]

4.3 Repeated run: protocols, rule shapes, timeouts, removal and routes

The initial run is small: one region, one instance family, an almost idle network and less than an hour of records. On 11 November 2026 we repeated it with a wider design. Ten instances, five roles on x86 (t3.micro) and on Arm (t4g.micro), ran in a dual-stack virtual private cloud with peers in the target's subnet and in a second subnet. Three target configurations were compared: tracked rules (inbound traffic allowed only from the cloud's address ranges), untracked rules (all protocols allowed to and from 0.0.0.0/0 and ::/0 in both directions) and tracked rules with the interface's TCP established timeout set to 60 s. Each of five repetitions replaced the security group, added network ACL deny entries, removed them again after about 190 s, and finally added a more specific route that sent the second subnet's prefixes to an unattached interface. Probes covered established and new TCP, UDP and ICMP flows in both address families. A unit is one combination of configuration, probe, address family, peer side, repetition and architecture, which gives 20 units per cell of Table 4. The units of one repetition share a deployment and its phase timing, so the 20 units of a cell are not 20 independent deployments. The five repetitions are the replications, but they share one account, one region and one day on an almost idle network, so they do not sample variation

across deployments, regions or load. Outcomes were taken from 890 console dumps of the instances and, independently, from 102,826 flow records in 43 delivered objects.

Table 4 Repeated run with tracked rules unless stated: units in which every exchange succeeded / all units (5 repetitions × 2 architectures × 2 address families). 0 / 20 means that no unit had only successful exchanges; mixed units had both successful and failed exchanges; units without an open connection are noted separately

Flow	Security group, same subnet	Security group, other subnet	plus ACL entries, same subnet	plus ACL entries, other subnet	ACL entries removed, other subnet	Route, other subnet
Established TCP (5 s)	20 / 20	20 / 20	20 / 20	0 / 20	11 / 20 (9 mixed)	0 / 20
Established TCP, idle for 90 s	20 / 20	18 / 20 (a)	20 / 20	0 / 20	11 / 20 (5 mixed) (b)	0 / 20
Established UDP (same 5-tuple)	20 / 20	20 / 20	20 / 20	0 / 20	0 / 20	0 / 20
Established ICMP (same identifier)	20 / 20	20 / 20	20 / 20	0 / 20	0 / 20	0 / 20
New TCP, UDP and ICMP	0 / 20	0 / 20	0 / 20	0 / 20	0 / 20	0 / 20
Untracked rules, any established flow	0 / 20	0 / 20	0 / 20	0 / 20	0 / 20	0 / 20
60 s TCP timeout, TCP idle for 90 s	0 / 20	0 / 20	0 / 20	0 / 20	0 / 20	0 / 20

(a) In the two remaining units the idle connection had not been opened in that repetition; no exchange failed. (b) The remaining 4 units had no successful exchange; the aggregate does not separate failed exchanges from connections that were not open.

- **Tracked flows of all three protocols survive the isolation security group.** With tracked rules, established TCP, UDP and ICMP flows that exchanged data every 5 s succeeded in 20 of 20 units on both sides of the subnet boundary, an idle TCP connection survived in every unit in which it had been opened (18 of 18 across subnets), and new connections of all three protocols failed in 20 of 20 units.
- **Whether a flow survives depends on the rules and the timeouts.** With untracked rules every established flow, including ICMP, failed as soon as the group was replaced (0 of 20 in every cell), although the documentation describes untracked flows for TCP and UDP and states that ICMP traffic is always tracked [8]; we report the observation without a mechanism. The ICMP probe was one echo process on the peer per phase, with one identifier and increasing sequence numbers every 5 s, so a change of identifier does not explain it; we have not repeated this cell with packet capture. The loop does not depend on the observation, because the emulator treats every established flow as tracked. Which established flows are tracked follows from the rule shape: a TCP or UDP flow allowed by a rule for all addresses in one direction and by a rule for all addresses and ports in the other is documented as untracked [8], a combination that arises, for example, when a group admits inbound traffic from 0.0.0.0/0 and keeps the

default outbound rule; the exception is a connection that the provider tracks automatically, such as one through a network address translation gateway or a Network Load Balancer [8]. Such flows end at a single replacement of the group, so the condition on established connections in the random mix of the evaluation counts tracked flows only (Sects. 10.1 and 10.5). With a 60 s TCP established timeout, a TCP connection idle for 90 s did not survive (0 of 20).

- **Removing deny entries restores tracked TCP connections.** After the entries had blocked traffic for about 190 s and were removed, tracked TCP connections that exchanged data every 5 s resumed in every unit (11 of 20 without a failed exchange, 9 after some failures); for the idle TCP connection 11 units resumed fully, 5 partly and 4 not at all, and with a 60 s TCP timeout no TCP connection resumed (0 of 20). UDP and ICMP flows did not resume (0 of 20), which is consistent with the documented UDP stream timeout of 180 s [8].
- **A route change blocks without a record.** The route stopped every flow to the other subnet at the application level (0 of 20) and none inside the subnet, and we found no REJECT record for the dropped packets at any interface.
- **Changes take effect with a delay, and intervals are skipped.** Accepted records for flows that should have been blocked had aggregation windows that ended up to about 82 s after the network ACL change; this is an observed window end, not a bound on how long a change takes to apply. Besides 102,826 records with status OK, 122 records were marked as skipped data, all on four interfaces. The windows of the records whose length we checked were 26–32 s, below the configured maximum of one minute. Over 37,427 records the delivery delay had a median of 175 s, a 95th percentile of 310 s and a maximum of 327 s; of the 43 objects, 21 were delivered 18–22 s and 21 were delivered 173–176 s after their latest record, and one after 306 s.
- **A configuration analysis answers a different question.** After the security-group change, the provider’s Reachability Analyzer, a configuration analysis in the line of [44], reported no path for both peer paths in each of the 4 analyses, as a configuration analysis must once the group has no rules; this matches the failed new connections, while established exchanges on the same paths continued.

4.4 Attribution run: which layer does a rejection record belong to?

In the first two runs the deny entries sat in the peers' subnets, and a rejection by the network ACL appeared at the peer's interface while a rejection by the security group appeared at the target's. To test whether records separate the layers in general, a third run on 11 November 2026 placed the network ACL at the target's own subnet. Three Arm instances (t4g.micro) ran in one virtual private cloud with IPv4 only: X and a peer Ps in subnet A, a peer Po in subnet B. Each peer opened a new TCP connection to X every 5 s and exchanged one byte every 5 s on an established connection. Each of three repetitions went from a baseline to the isolation security group on X's interface for about 250 s, and then, with the working group restored, to an inbound deny entry for subnet B's prefix in the network ACL of subnet A for about 250 s. The flow-log format additionally contained the flow direction, the traffic path and the reject reason. Outcomes were taken from console output and from 2,785 flow records in 13 delivered objects.

Table 5 Attribution run, three repetitions summed. a Application-level exchanges that succeeded / failed. b REJECT records at the interface of X

a

Phase	Ps, same subnet: new	Ps, same subnet: established	Po, other subnet: new	Po, other subnet: established
Baseline	104 / 0	103 / 0	106 / 0	106 / 0
Isolation security group	0 / 138	140 / 0	0 / 138	138 / 0
Deny entry in the target's subnet	130 / 0	131 / 0	0 / 134	0 / 134

b

Phase	Flow	REJECT records	Direction	TCP flags	Reject reason, traffic path
Isolation security group	Po to X, new	136	ingress	2 (SYN)	empty
Isolation security group	Ps to X, new	133	ingress	2 (SYN)	empty
Deny entry in the target's subnet	Po to X, new	127	ingress	2 (SYN)	empty
Deny entry in the target's subnet	Po to X, port of the established connection	135	ingress	2 or 0	empty

Table 5 shows the result. The application view repeats the earlier semantics: the security group stopped new connections and no established one, and the stateless entry stopped both from the other subnet and neither inside the subnet. In the records, however, the two layers were indistinguishable. A SYN rejected by the network ACL of the target's subnet and a SYN rejected by the isolation security group were recorded at the same interface, in the same direction and with the same flags, and the reject-reason and traffic-path fields were empty in

both cases. Of the 262 rejections of traffic from Po during the deny-entry phase, 256 carried the SYN flag and 6 no flag; the SYN packets on the port of the established connection were the client's attempts to reopen it. During the same phase Po's own interface recorded its transmissions to X as accepted (110 records for new and 99 for established connections). The delivery delay in this run had a median of 160 s and a maximum of 327 s.

4.5 Record delivery over 72 hours in two regions

A fourth measurement followed record delivery alone for 72 hours from 11 November 2026, with two Arm instances (t4g.micro) at low traffic in each of two regions, Seoul and Northern Virginia, in the same account and with the same one-minute aggregation interval. The records of one delivered object arrive together, so we treat the object as the independent sample and give record-level values only for comparison with the earlier runs. Of the objects that contained records from the 72 hours, 1,703 in Seoul and 1,722 in Northern Virginia, half arrived within 290 s and 303 s of the end of their oldest record, 95% within 336 s and 344 s, and the last after 1,211 s and 1,040 s. Over the 25,575 and 29,237 records, the 95th percentile of the delay was 319 s in both regions and the maximum 1,211 s and 1,040 s. Delivery took longer than the documented typical value of about ten minutes (600 s) [10] for 2 of 1,703 objects (0.12%; 4 records) in Seoul and 1 of 1,722 (0.058%; 2 records) in Northern Virginia, after 1,031–1,211 s; apart from these three objects, no object arrived later than 369 s. No interval was marked as skipped in either region; 2,999 and 1,777 records were marked as having no data. A tail that rare is invisible in runs of an hour or two, which is one reason why the loop treats delivery time as unbounded (requirement M5 below).

4.6 Consequences for the loop

The measurements translate into five requirements, which Sects. 7–9 implement. They were used to design the loop; whether the controller meets them is a separate question that the measurements themselves do not answer and that has to be tested on the controller.

- **M1** (Tables 3, 4). An acknowledgement, and a configuration analysis, establish at most that new connections will be refused. Established flows must be covered by a stateless layer or reported as unverified.
- **M2** (Table 4). Removing a stateless entry restores tracked TCP connections, so a stateless enforcer protects only as long as the target cannot revert its policy.

- **M3** (Sect. 4.2). Accepted traffic must be read at the interface beyond the last trusted layer, because the target's own interface records as accepted what a later layer drops.
- **M4** (Table 5, Sect. 4.3). A rejection record is evidence for an enforcer only if the configuration rules out every other layer that could reject the same packet at the recording interface; a route change yields no such evidence at all.
- **M5** (Sects. 4.2–4.5). Records of traffic that should already be blocked can end more than a minute after a change, records arrive minutes later, occasionally after more than a quarter of an hour, and intervals can be skipped; the decision must wait and must never read missing records as success.

All four runs used one account on an almost idle network, the first three in one region, and none covered interfaces with a different default TCP timeout, such as the default of 350 s for Nitro v6 instance types other than P6e-GB200 [8]. The initial run calibrated the record delay of the emulator of Sect. 10; the behaviours found in the later runs are encoded in it as described in Sect. 10.1.

5 Compromise-Domain Model

Every enforcement or observation point p is described by an attribute vector d(p) = (L1, L2, L3, C, L4), listed in Table 6. Attributes are taken only from infrastructure interfaces, never from the point itself, and are stored as timestamped snapshots: evidence is judged against the snapshot valid when it was recorded, and selection uses a snapshot acquired immediately before enforcement. An attribute that cannot be obtained, or whose source fails transport authentication, is set to unknown.

Table 6 Attribute vector of a point and its sources in the example provider

Attribute	Domain	What one compromise at this level controls	Source
L1	Execution environment	Processes inside one virtual machine, container or pod	Instance identifier, pod identifier
L2	Node	All execution environments that share a kernel	Node name with cluster identifier, instance identifier
L3	Management plane	All agents managed by one management server	Agent registration; the provider's own plane for provider-managed points
C	Control principals	Every resource that these principals may change	Attached credentials, role-assumption closure, principals allowed to modify the point
L4	Account	Resources changeable with account administration rights	Account identifier

C(p) is the union of the principals reachable from p and the principals able to modify p. Reachability starts from the credentials attached to p and follows role assumption recursively with a visited set; a policy condition that cannot be evaluated is treated as satisfied, so the computed closure over-approximates the set of reachable principals with respect to conditions. Role assumption is not the only way to obtain another principal's permissions. Documented escalation paths in the example provider [59] include passing a role to a compute resource that the principal may launch; creating access keys or login credentials for another user, or adding a user to a group; changing the code or configuration of an existing resource that runs with a role; and creating, attaching or rewriting a policy or a role's trust policy. The closure therefore follows an edge from a principal q to a principal r whenever q may obtain credentials of r, join a group whose permissions r represents, or cause code to run with r's permissions, which covers the first three kinds, and treats a principal that may write the access policy or the trust policy of any principal, its own included, as reaching every principal. The edge set is defined by these conditions rather than by the list in [59], so a path that a list omits still belongs to it; whether a computed closure contains every such edge is the first assumption of Proposition 2, and tools such as [46, 47] are one way to compute or check it. Without the second rule, a principal that may rewrite its own policy is absent from C(e) when selection runs and can become a modifier of e afterwards; with it, the target's role reaches every principal and selection ends UNRESOLVED, as in scenario S10. The reference implementation used in Sect. 10 follows role assumption only; the escalation rules and the account rule below are specified here but not implemented, and the emulator represents their effect only through the condition that the target's role reaches every role (Table 7, S10). Modifiers are principals whose policies allow changing p's policy, its binding or its records. The binding of an enforcer is the set of resources to which its policy applies: the interfaces to which a security group is attached, and the subnets associated with a network ACL. A principal that may associate a subnet with another network ACL [58] removes the deny entries of Sect. 4.1 from that subnet without changing them. The request for that change names the subnet's current association and the new ACL, not the ACL being replaced [58], and every virtual network has a default ACL with which a subnet can be associated without creating a new one. Binding modifiers are therefore computed by evaluating the association action against each subnet associated with e and against every ACL of the virtual network, the default ACL included, rather than against e alone; likewise, the principals that may replace the security groups of an interface are evaluated against each interface to which the isolation group is attached, for any group. A condition that cannot be evaluated counts as satisfied, as

for reachability. Three conventions keep the relation useful. Credentials of an agent management server are represented by L3 and not by C, because otherwise all points managed by one server would depend on each other and no evidence would remain. Account administration is represented by L4: principals whose policies allow writing any access policy or trust policy in the account are not entered into C, because they can modify every point and would otherwise make all points of the account depend on each other. The exclusion is safe only while the target cannot reach them. The account rule therefore treats two points of one account as sharing compromise if the principals reachable from either of them, or the credential of the management plane of either, include an account-administration principal (Definition 1). For a target that reaches account administration, every point of the account then shares compromise with it: no evidence from the account is admitted, no enforcer, verifier, observer or disruptive action in the account counts as independent, selection returns UNRESOLVED, and the controller reports a suspected compromise of the account. Roles of account B are excluded as the trusted base. Provider-managed points such as security groups and flow logs receive unique L1 and L2 values and the provider's plane as L3.

Definition 1 (shared compromise). share(p, q) = 1 if L1(p) = L1(q), or L2(p) = L2(q), or C(p) ∩ C(q) ≠ ∅, or L4(p) = L4(q) and the account rule of this section applies to p or q, or any of L1, L2, C is unknown for p or q; otherwise share(p, q) = 0.

Lemma 1. share is a deterministic, symmetric function of the two attribute vectors and is monotone under missing information: replacing a known attribute by unknown can change share from 0 to 1 but not from 1 to 0. *Proof.* Each disjunct is symmetric and depends only on the vectors, and the unknown clause only adds cases that evaluate to 1. □

Lemma 1 makes the model safe against missing attributes under constraint C3: missing metadata removes evidence and candidates and never admits them. It does not protect against an imprecise access analysis that returns a known but incomplete closure; Proposition 2 states that assumption explicitly. L3 is not part of share, and L4 enters it only through the account rule. The management plane enters separately, as a quorum and a score cap in the trigger and as an exclusion condition in selection, which keeps independence at node level distinct from independence at management-plane level. The relation is not transitive, so groups over C are formed as connected components.

6 Independence-Aware Containment Trigger

The trigger decides whether containment of X should start. Its input is a set of mismatches between records of different points about the same flow, for example a flow that the flow log and the receiving node observed but that is absent from the records of X's agent. Mismatches may come from any detector that can attribute them to a point; how records are matched is outside the scope of this article. Each mismatch w carries a weight v(w) ∈ (0, 1] for its type and the trust t(w) ∈ [0.5, 1] of its source. Trust is updated only by later confirmation that did not use the decision itself, so that a successful containment does not reinforce the sources that asked for it.

Evidence is collected per incident key (target, flow or policy intent, observation window). A mismatch is admitted only if its attribution condition holds; in particular, a flow missing from X's records counts only if X's record channel shows no sequence or heartbeat gap in the window, since otherwise a logging outage would look like concealment. Evidence from sources with share(source, X) = 1 is discarded. The remaining sources are partitioned into node groups (by L2), management-plane groups (by L3, only planes that are known and differ from L3(X)), and control groups (connected components of the C-intersection graph). The evidence score takes one value per node group and caps the sum per management plane:

$$E(X) = \sum_{m} \min\left(c_{L3}, \sum_{j \in J(m)} \max_{w \in G_j} v(w)\, t(w) \right) \qquad (4)$$

where m ranges over the management planes of the admitted sources, including L3(X) and an unknown plane counted as one, G_j is a node group, and J(m) is the set of node-group indices whose maximising source belongs to plane m. The plane quorum below counts only planes other than L3(X), whereas the score includes evidence from sources in X's plane, capped like any other plane. Containment is triggered if the number of node groups is at least k_L2, of plane groups at least k_L3, of control groups at least k_C, and E(X) ≥ θ. Otherwise the decision is HOLD, and groups whose reports contradict independent observations are recorded for a separate check. We use k_L2 = 2, k_L3 = 1, k_C = 2, θ = 1.5 and c_L3 = 1.0; Sect. 10.4 varies them.

Proposition 1 (fabrication resistance). Let Y be a healthy workload, so that honest sources report no mismatch about Y, and let k_L2 ≥ 2, k_C ≥ 2 and c_L3 < θ. If all sources that

fabricate mismatches about Y lie (i) on one node, or (ii) in one connected component of the C-intersection graph, or (iii) in one management plane, then the trigger returns HOLD. *Proof.* In case (i) there is one node group, below k_L2. In case (ii) there is one control group, below k_C. In case (iii) all node-group maxima fall into the inner sum of one plane, so $E(Y) \le c_L3 < \theta$. □

Case (iii) is why a cap is used in addition to quorums. With quorums alone, a management plane other than Y's that controls two nodes with different credentials satisfies all three counts. Raising k_L3 to 2 would also stop it, but it would require evidence from two planes outside X's own for every real detection; Sect. 10.4 quantifies that loss. A compromise of two separate planes falls outside the threat model and is not covered by the proposition.

7 Control-Path-Independent Enforcer Selection

Once containment is triggered, the controller lists the enforcers that can act on X's paths and removes those that share a control path with X. An enforcer e shares a control path with X if share(e, X) = 1 or if the credential of X's management plane is among the principals that can modify e. The second condition covers the common arrangement in which an agent management server also holds permission to change security groups. Among the remaining enforcers at least one must be independent in both plane and control, that is $L3(e) \neq L3(X)$ and $C(e) \cap C(X) = \emptyset$. If none exists the controller does not report success; it records UNRESOLVED and asks the operator whether to approve a disruptive but independent action; when the account rule of Sect. 5 applies, no such action exists in the account, and the request says so and reports a suspected compromise of the account. The verifier V, which sends probes, and the observer O, which supplies flow records, are chosen with the same relation: share(·, X) = 0, $L3(O) \neq L3(X)$, and no control path shared with any selected enforcer, so that neither the target nor the enforcer can vouch for itself. Selection is repeated when an access-policy change event concerns a selected component.

Algorithm 1 SELECT(X, candidates, snapshot)

$S \leftarrow \{ e \in \text{candidates} : \text{share}(e, X) = 0 \wedge \text{cred}(L3(X)) \notin C(e) \}$
$B \leftarrow$ non-disruptive candidates not in S // applied as best effort, never trusted
$S_auto \leftarrow \{ e \in S : e \text{ is non-disruptive} \}$ // disruptive members of S wait for approval (Sect. 9)
if no $e \in S_auto$ has $L3(e) \neq L3(X)$ and $C(e) \cap C(X) = \emptyset$ then
    deploy B; return UNRESOLVED // ask for approval (Sect. 9)

V ← verifier with share(V, X) = 0 and no control path shared with S
O ← observer with share(O, X) = 0, L3(O) ≠ L3(X), no control path shared with S
if V or O does not exist then deploy S_auto ∪ B; return UNRESOLVED // nothing may vouch for the result
return (S_auto, B, V, O)

An attribute that is unknown for a candidate excludes it by Lemma 1, so missing metadata can only shrink S_auto and lead to UNRESOLVED. The guarantee also rests on the computed modifier set C(e) of each selected enforcer and on the effective permissions of X's management-plane credential, which Proposition 2 assumes to be complete. A member of S_auto with L3(e) = L3(X), such as an agent on another node managed by X's plane, is applied but not covered by Proposition 2, which requires L3(e) ≠ L3(X); in the deployment of Sect. 4.1 every non-disruptive candidate other than the guest agent is provider-managed. Excluded non-disruptive enforcers are still applied as a best-effort layer. They need no approval, they reduce exposure when the adversary does not or cannot act, and they are never counted as evidence. Because a best-effort layer may reject the same probe as a selected enforcer, a rejection has to be attributed before it counts; Sect. 8 gives the rule.

Proposition 2 (revert resistance). Let A(X) be the principals available to the adversary of Sect. 3.2: C(X), the credential cred(L3(X)) of X's management plane, the principals attached to the other execution environments on X's node if the adversary has escaped, and the principals attached to the workloads that L3(X) manages if the adversary controls that plane. Suppose the computed closure of X contains every principal reachable from X, by role assumption or by the escalation edges of Sect. 5, and every principal in A(X) is in C(X), is cred(L3(X)), or is an account-administration principal reachable from X. Then the adversary cannot modify the policy or the binding of an enforcer e with share(e, X) = 0, L3(e) ≠ L3(X), cred(L3(X)) ∉ C(e) and C(e) ∩ C(X) = ∅. *Proof.* By the second assumption the adversary's principals are in C(X) ∪ {cred(L3(X))} or are account-administration principals reachable from X, and its code runs in X's execution environment and node and, if it controls L3(X), in agents of that plane. Modifying the policy or the binding of e requires a principal in C(e) or an account-administration principal, code in e's environment or node, or e's own management plane. A principal in C(e) is excluded by the two conditions on C(e). An account-administration principal is excluded because share(e, X) = 0 implies, by the account rule, that neither X's closure nor cred(L3(X)) contains one, so the adversary holds none. Code in e's environment or node is excluded by share(e, X) = 0, and e's plane by L3(e) ≠ L3(X). □

The two assumptions carry the weight. The second fails when X shares a node with other credential-holding workloads and the adversary escapes, or when the adversary runs commands through a compromised management plane on the workloads it manages; our implementation does not add those principals to the adversary's set, the emulator does not model them, and Proposition 2 then gives no guarantee (Sect. 11). Treating unevaluable conditions as satisfied makes the closure conservative, but an analysis that misses an edge silently voids the guarantee. Section 10.5 includes that case.

Proposition 2 is about the policy and the binding of e, which includes the subnets associated with a network ACL (Sect. 5), not about the target's side of the traffic that the policy covers. Network ACL rules match traffic by CIDR range [60], and the deny entries of Sect. 4.1 match the addresses that X has when they are written; the isolation security group applies to the interface it replaces. A principal that may assign further private addresses to X's interface, which the provider can pick automatically from the subnet's range [61], or attach another interface to X [62], can move X's traffic outside the block without modifying e. Where such a principal is in A(X), two routes lead around the block. If the security group is not trusted, as in S4 and S5, X can revert the group and open new connections to another subnet from an additional address of the same interface. Even if the group is trusted, a principal that may create an interface and attach it to X [62] gives X an interface that the isolation security group does not cover, because the group replaced the groups of the original interface only, and an address that the deny entries do not match. In both cases the network ACL that the loop relies on is unchanged and the probes, which target the old address, keep being rejected, so a declaration of EFFECTIVE, including one for all of Π, can be false. Let M(X) be the principals that may change X's addresses, interfaces or source and destination check. Under its assumptions, Proposition 2 therefore guarantees that the policies and bindings of the selected enforcers remain unchanged. That these policies keep covering X's traffic requires the additional condition $M(X) \cap A(X) = \emptyset$. Adding that condition to selection, matching deny entries on the subnet rather than on X's addresses, or invalidating a declaration on any address or interface change of X would close the gap; our implementation does none of these, the random mix of Sect. 10.1 does not sample such a permission, and no targeted scenario of Sect. 10.5 contains it.

8 Scoped Isolation Verification

Verification answers what the acknowledgement cannot: whether traffic of X is in fact rejected, on which path classes, and by which layer. Figure 3 shows the timeline. Before deployment the verifier sends baseline probes; their answers, or the ACCEPT records that arrive later for the intervals before deployment, establish which path classes were reachable. The controller takes a snapshot of the network configuration that applies to the verifier's and the target's interfaces, deploys the selected and the best-effort enforcers, and records T_iso. Because records are aggregated per interval and a change takes time to take effect, records up to T_iso + Δt are not evaluated; Δt must exceed the latest window end of about 82 s observed in Sect. 4.3 (requirement M5); that value is an observation, not a guaranteed bound, and Δt is a parameter of the deployment.

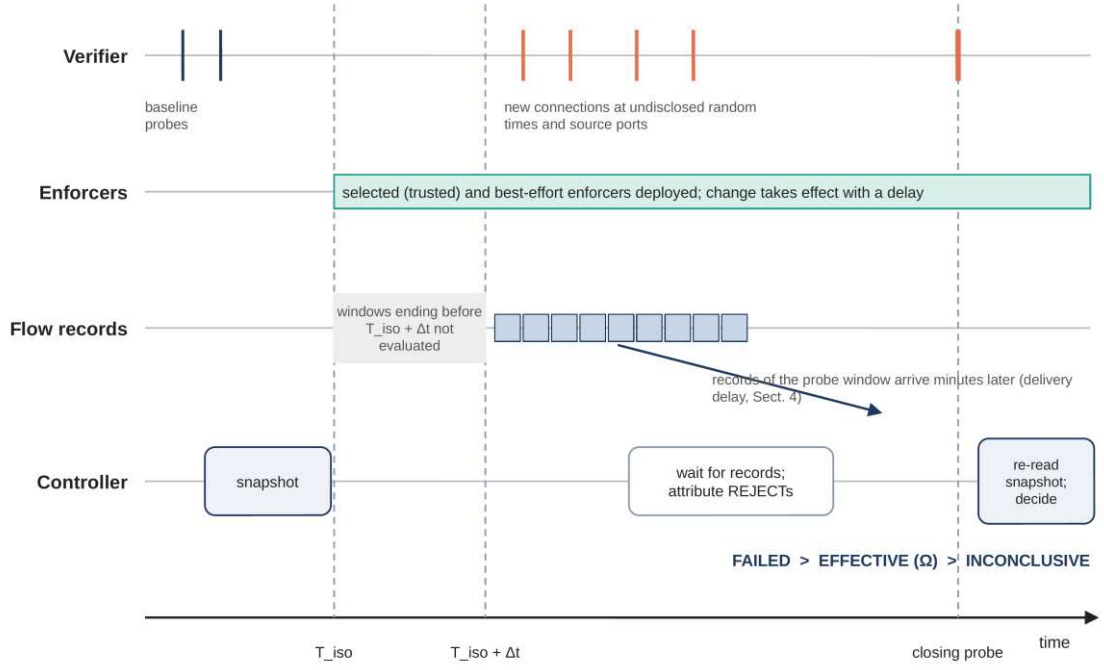


Fig. 3 Verification timeline for one round. A configuration snapshot is taken before deployment and read again at the decision; probes are new connections at random times and source ports; records whose windows end before T_iso + Δt are not evaluated; the decision waits for the records of the probe window and ends with closing probes

After T_iso + Δt the verifier sends several probes per probe class, each a new connection with a source port and time that it chooses at random and does not disclose, so that the timing and source ports of test traffic are less predictable to an enforcer. We do not claim that probes are indistinguishable from other traffic, and the closing probe, which follows the wait for records, is easier to anticipate than the others. A probe class is a pair (side, family) for which the verifier has an interface, hosts exist, and the selected enforcers claim coverage.

Attribution. A REJECT record names an interface and a direction but not the rule or the layer that rejected the packet (Sect. 4.4, requirement M4). The controller therefore attributes a record to a selected enforcer e only if, according to the configuration snapshot, which is read again at the decision, e is the only layer whose rules deny the probe at the recording interface in the recorded direction. For deny entries in the verifier's subnet this requires that the verifier's own security group allows the probe and that no other entry of that network ACL denies it. A rejection recorded at the target's interface is attributed to the security group attached there only if the network ACL of the target's subnet allows the probe, and it is never attributed to a network ACL while an isolation security group is attached. A record must in addition belong to the probe: its 5-tuple must carry the probe's undisclosed source port, its TCP flags, which the provider combines over the aggregation interval [10], must be SYN alone, and its aggregation window must lie entirely after $T_{iso} + \Delta t$ and before the closing probe, so that it cannot merge packets of an earlier connection, of a retransmission or of a packet that arrived after a connection was closed, which the provider also records as rejected [10]. The snapshot condition must hold at deployment and at the decision, and no configuration change event may concern the recording interface or its subnet in between; because the provider's interfaces are eventually consistent, a change recorded within Δt of either read also voids the condition. A record that fails any of these conditions is not evidence for any enforcer. Rejections without a record, such as packets dropped by a route or by a service that does not answer, never produce evidence.

When the records of the probe window have had time to arrive, the verifier sends one closing probe per class and the controller decides, in this order:

- **FAILED** for the round if any probe was answered, or any record shows accepted traffic of X on a path class that the selected enforcers claim to block. Such records are read at the interface beyond the last trusted layer, which for paths covered by network ACLs is the peer's interface (requirement M3); traffic on an allow list for management and recovery is exempt;
- **EFFECTIVE** for a probe class if the round has not FAILED and a REJECT record of one of the verifier's own probes of that class is attributed to a selected enforcer;
- **INCONCLUSIVE** for a probe class otherwise. The class is probed again; records of earlier probes that arrive late still count as long as the deployment and the snapshot have not changed.

The order makes the states exclusive, and the middle rule is the essential one: silence is never success, and a rejection of unknown origin is not success either. A hung service, a delayed record, a skipped interval and an unattributable rejection all lead to INCONCLUSIVE, not to EFFECTIVE. The scope Ω of an EFFECTIVE declaration follows from what was observed. A rejected new connection verifies the class for new connections. An established connection is verified only if the rejecting layer is stateless, because a stateful layer that rejects a new connection says nothing about flows it already tracks (requirement M1). Path classes that no probe can reach, such as established connections inside the target's subnet or a subnet without a verifier interface, remain outside Ω and are reported as unverified.

Proposition 3 (no silent success). Assume the observer's records and the configuration snapshot are correct. If the controller declares a probe class EFFECTIVE, then after $T_{iso} + \Delta t$ at least one fresh connection of that class was rejected at an interface where only a selected enforcer could reject it, no probe was answered up to the decision, and no accepted traffic of X on a covered path class was recorded in the evaluated intervals. *Proof.* Immediate from the ordered rules and the attribution condition: EFFECTIVE requires the absence of both FAILED conditions and the presence of an attributed REJECT record. □

Propositions 2 and 3 together give the property that matters to the operator. If the access analysis is sound, the trusted base holds and the observer's records are correct, then a path class declared EFFECTIVE was blocked when it was probed, by an enforcer whose policy the adversary cannot change afterwards; that the policy still covers X's traffic after a change of X's addresses or interfaces requires in addition $M(X) \cap A(X) = \emptyset$ (Sect. 7). An EFFECTIVE declaration is thus a statement about the time of probing; the propositions do not by themselves guarantee that the class stays blocked for the horizon H of objective (1), which is what the evaluation measures. Verification alone establishes only the first half, at one point in time; selection alone establishes only the second, without showing that the block exists.

9 Approval-Gated Escalation

A FAILED round leads to one redeployment and a new round; evidence about the old deployment is discarded. A second failure, repeated INCONCLUSIVE rounds, an empty selection, or path classes that remain outside Ω lead to escalation. The next containment means must not share a control path with X and must block at least one path class that is still open or unverified. In the example provider the order is security group and network ACLs,

then a routing change, then stopping the instance after a snapshot for evidence. A route to an unattached interface leaves no REJECT record (Sect. 4.3), so path classes blocked only by a routing change remain INCONCLUSIVE and are reported as unverified. Service impact is judged separately from blocking power: stopping an instance always requires approval, and a routing change requires it when other workloads in the subnet are affected. A stop has two costs beyond the service. An instance of an Auto Scaling group that is stopped is replaced [63], so it has to be moved to standby first, as the provider's runbook does [57]; otherwise the group can terminate it, which may delete its volumes and with them the disk evidence, and launches a replacement with the same role, which reopens the exposure if the path of compromise persists. A snapshot of the volumes also does not preserve memory, so volatile evidence is lost unless it is captured before the stop, the boundary that CUSTOS addresses [23]. Our implementation performs neither step, and Auto Scaling membership is neither part of the emulated semantics of Table 2 nor a condition of Table 7. The request to the operator states the unverified scope and the reason. If the request is denied or times out, the controller declares EFFECTIVE for the verified scope only, or UNRESOLVED when nothing was verified. After an approved stop the controller confirms through the trusted provider interface that the instance has reached the stopped state, which is the evidence for declaring EFFECTIVE for all of Π; the absence of flow records for the interface is only a consistency check, because records can be missing for other reasons.

The loop therefore ends in one of three declared states: EFFECTIVE for Π, EFFECTIVE for a scope Ω with the remainder listed, or UNRESOLVED. The second and third are honest outcomes that hand the residual risk to the operator, which is the purpose of objective (1) taking priority over objective (3).

10 Evaluation

10.1 Method

We did not stage repeated incidents with a privileged adversary in a production cloud, and the conditions that decide the outcome (which role can reach which, whether a management plane is compromised, when the adversary reverts) are properties of the environment and the adversary, not of the provider. We therefore evaluate the loop with a discrete-event emulator of the provider semantics of Table 2. The reference implementation of the model, the trigger, selection and the ordered decision rules is used inside the emulator; the code, inputs and

outputs are available as stated under Data availability. The emulator, the reference implementation and the measurement scripts were written with a generative AI assistant (Claude Opus 5.5, Anthropic), which also ran the live measurements of Sect. 4 through the provider's scripting interface. The author specified each experiment, reviewed the code and the outputs of the runs, and checked the numbers reported here against the result files. Three checks did not rest on that review alone. The emulated rates of the four configurations that accept an acknowledgement were compared with closed-form expectations computed separately from the conditions of Table 7, and agree within one Monte Carlo standard error (Sect. 10.5); the 72-hour delivery measurement of Sect. 4.5 was aggregated twice, in two separate sessions with separately written scripts and different batch sizes, with identical results; and drafts of the manuscript, including the tables, were reviewed by a second model as described in the statement on generative AI.

The emulator encodes the live measurements of Sect. 4 as follows. Flow-record delivery delay is drawn from 40–306 s after the interval end, the 5th to 95th percentile of the initial run, with an additional delay of 5–30 min for a configurable fraction of intervals and a configurable fraction of skipped intervals. The 72-hour measurement of Sect. 4.5 falls outside this base range at both ends: 9.6% of the records in Seoul and 9.1% in Northern Virginia arrived later than 306 s, all but six within 369 s, and the six, in three objects, after 1,031–1,211 s, which is beyond the nine to eleven minutes of the sensitivity run in Sect. 10.6 but within the additional delay of 5–30 min that the emulator adds to a fraction of intervals (0.05 per interval in the random mix, 0.5 in S8). An enforcement change is acknowledged after 2 s and takes effect after a further delay drawn from 0–60 s; Δt is 120 s. This range does not bound the live observations: if the accepted record whose 26–32 s window ended about 82 s after a change (Sect. 4.3) shows a packet that passed the network ACL, the change had not yet taken effect for that flow some 50 s after it was made, and the emulator tests neither longer delays nor a Δt close to the longest delay. Established flows survive a security-group change and are cut by deny entries for other subnets, and tracked connections reopen when the adversary removes those entries; UDP and ICMP flows are treated like TCP in this respect, which overstates exposure (Sect. 4.3). The controller does not know which layer rejected a probe. It sees the interface of each REJECT record and the configuration snapshot, and it applies the attribution and class-splitting rules of Sects. 3.3 and 8. To isolate their effect, the loop and A1 are also run with the earlier rules, which attribute a record by its interface alone, do not split classes and use $\Delta t = 60$ s. A stop is confirmed through the provider's instance state 30 s after it is

issued. False-effective reports are counted from 60 s after a declaration for every configuration, so that the propagation of a change does not count against any of them.

Three limits apply to every number in this section. First, the trigger (Sect. 6) and containment (Sects. 7–9) are evaluated in separate experiments, and every containment incident in Sects. 10.5–10.7 starts from a triggered target; Sect. 10.6 adds an integrated run from trigger to declaration. Second, the routing step of Sect. 9 is not emulated, so escalation goes from security group and network ACLs directly to the instance stop, and baseline probing is not modelled because the emulator knows which path classes exist. Third, the incident conditions are assumptions. The controller has not been run against a reverting adversary in the provider; the live evidence in this article concerns the provider semantics and observability on which the decision rules rest (Sect. 4), not the loop's end-to-end behaviour.

The adversary of Sect. 3.2 reverts the policy of every enforcer it can modify, either 60–180 s after the change takes effect or, when strategic, 2,000–3,000 s after it, which is after verification normally ends; the strategic adversary also sends no traffic while verification runs. The operator approves a disruptive action with probability 0.85 after 2–30 min; otherwise the request times out after 30 min. The trigger experiment uses a synthetic estate of 60 nodes with one to three workloads each, three agent management planes, and instance roles shared by node groups of four.

In the random mix (S0) the conditions of Table 7 are sampled independently, except that a role that reaches every role also reaches the roles that modify security groups and network ACLs. The network structure is fixed: besides the target's subnet there is one other subnet, whose network ACL the containment role can change and in which the verifier always has an interface (S12 adds a second one); Table 7 samples only whether the verifier has an interface in the target's subnet. The random mix therefore contains no estate with several other subnets whose network ACLs differ. Class splitting (Sect. 3.3) makes each such subnet a class of its own that needs a verifier interface, so such an estate can produce more unverified classes and stops than the random mix of Sect. 10.5 reports; we have not run that variant (Sect. 11). The probabilities are scenario assumptions, not estimates from incident data; Sect. 10.7 varies them one at a time. We drew five independent sets of 2,000 incidents and presented the same incidents to every configuration, so configurations are compared with exact McNemar tests on paired outcomes. In the targeted scenarios S1–S14 (Sect. 10.5), 2,000 incidents each, one condition is fixed on a base with other hosts in the target's subnet, a verifier interface in that

subnet, IPv4 only, no established connections and no additional adversary capability. S12–S14 add three conditions: a second other subnet whose network ACL the containment role cannot change and in which the verifier has no interface (S12); a security-group change that is acknowledged but never applied, while the target, which can change the network ACL of its own subnet, adds a deny entry for the verifier's addresses there and stays silent (S13); and the acknowledged but unapplied change alone (S14).

Table 7 Conditions sampled in the random mix (S0). A role that reaches every role also reaches the other two roles; the options of S12–S14 are off

Condition	Probability
Adversary can impair the guest agent	0.7
Target's role can reach a role that modifies security groups	0.3
… and the access analysis misses that edge	0.05
Management-plane credential can modify security groups	0.2
Adversary also controls the target's management plane	0.5
Target's role can reach the role that modifies network ACLs	0.05
Target's role can reach every role, including instance start	0.02
Established (tracked) connections exist	0.6
Other hosts exist in the target's subnet	0.4
Verifier has an interface in the target's subnet	0.7
Target has an IPv6 address	0.3
Target answers no connection attempts (hung service)	0.1
Additional record delay / skipped interval, per interval	0.05 / 0.02
Strategic adversary (silent, late reversion)	0.5

10.2 Configurations

We compare the loop with configurations that reflect current practice and with ablations of the loop (Table 8). B2 uses the containment action of the provider's managed service and incident response guidance [4, 5], modelled as a single replacement of the security groups; the intermediate step of the provider's runbook [57] is not modelled (Sects. 4.1 and 10.5). Operational guidance for tag-based isolation already notes that whether established sessions survive the replacement depends on the rules of the original group [64]. Taking the acknowledgement as its success criterion is our model of an operator who records the change as done, not a statement about how the provider's service closes incidents. The same provider page describes a separate containment of an access-management principal by a deny-all policy and lists revoking credentials next to network isolation as short-term containment [5]. No configuration of Table 8, the loop included, revokes the target's credentials; Sect. 10.5 gives the expected effect of revocation on the baselines. B4 adds network ACL entries for IPv4, and B5 for both address families; B5 is the strongest configuration that relies on the acknowledgement. B3 adds a single connectivity check after the action, as a simple playbook step would. B1 is agent-based isolation with the agent's own status report; it fails whenever

the adversary can impair the agent and is included as a reference point, not as a competitor. A1 and A2 each remove one half of the loop; A1, which applies security group and network ACLs for both address families and verifies them, is the strongest configuration without selection.

Table 8 Configurations compared

Configuration	Enforcers	Success criterion	Basis
B1	Guest agent	Agent reports isolated	Agent-based isolation
B2	Isolation security group	Interface acknowledges	Provider's managed containment action, as a single replacement
B3	Isolation security group	One fixed probe after 120 s is not answered; up to two redeployments	Playbook connectivity check
B4	Security group and IPv4 ACL entries	Interface acknowledges	Provider's incident response guide
B5	Security group and ACL entries, both families	Interface acknowledges	Strongest acknowledgement-based configuration
A1	Security group and ACL entries, both families, all trusted	Verification and escalation of Sects. 8–9	Ablation: no selection
A2	Selected enforcers only	Interface acknowledges; approval for stop if nothing is selected	Ablation: no verification
Full loop	Selected enforcers trusted, others best effort	Verification and escalation of Sects. 8–9	Sects. 6–9

10.3 Metrics

All rates of the containment experiments are fractions of all incidents of an experiment and are reported with their integer counts; the trigger rates of Sect. 10.4 are percentages of the 4,000 trials of each case, or of the first 1,500 in Fig. 4b and 4c. The false-effective rate is the number of incidents that were declared EFFECTIVE, for Π or for a scope Ω, and in which at least one path class of the declared scope was open at some time between 60 s and one hour after the declaration (δ and H of objective (1)), divided by the number of incidents; baselines declare for Π. Because the loop may declare a narrower scope than Π or none, the rate is read together with the declared states and with the false-effective rate among incidents with an EFFECTIVE declaration. Residual exposure is the fraction of incidents with any path class open one hour after the declaration, whatever was declared, and we report it by declared state. Exposure time is the number of minutes during the two hours after detection for which at least one path class is open. Operational cost is measured by the time to an EFFECTIVE declaration, approvals requested and instances stopped per incident, stops that were avoidable in hindsight (security group and ACL entries, had they been trusted, would have closed every path class and would not have been reverted), and the management-plane calls, probes, record

queries and configuration snapshots per incident. For the trigger we report the false-containment rate under fabricated evidence and the detection rate for real bypasses. Intervals are 95% Wilson intervals.

10.4 Trigger: fabricated evidence and real detections (RQ4)

Table 9 reports how often fabricated mismatches about a healthy workload trigger containment, for 4,000 trials per case. Counting witnesses contains the healthy workload in every case except when the single compromised node happens to host one workload. Giving each node one vote prevented false containment in all trials only in the single-node case. The independence-aware trigger stops cases a–d, as Proposition 1 states, and the ablations show which condition is responsible: without the control quorum a single stolen credential used on nodes in two planes succeeds in 100% of trials, and without the plane cap a single management plane succeeds in 100%. Cases e and f lie outside the threat model, and the trigger fails there in 72% and 100% of trials. In case f the target's own plane contributes up to the cap to the score of Eq. (4) and one further node in another plane satisfies the plane quorum, so the threshold is reached. Figure 4a shows the same data.

Table 9 False-containment rate under fabricated evidence about a healthy workload, 4,000 trials per case. Cases e and f involve two separate compromises and are outside the threat model

Fabricating sources	Witness count	One vote per node	Ours	Ours, no plane cap	Ours, no control quorum	Stopped by
a One node	67%	0%	0%	0%	0%	node quorum
b Two nodes in the target's plane	100%	77%	0%	0%	0%	plane quorum, cap
c One credential on nodes in two planes	100%	100%	0%	0%	100%	control quorum
d Four nodes of one other plane	100%	100%	0%	100%	0%	plane cap
e One node in each of two other planes	100%	76%	72%	72%	76%	—
f Target's plane and one node elsewhere	100%	100%	100%	100%	100%	—

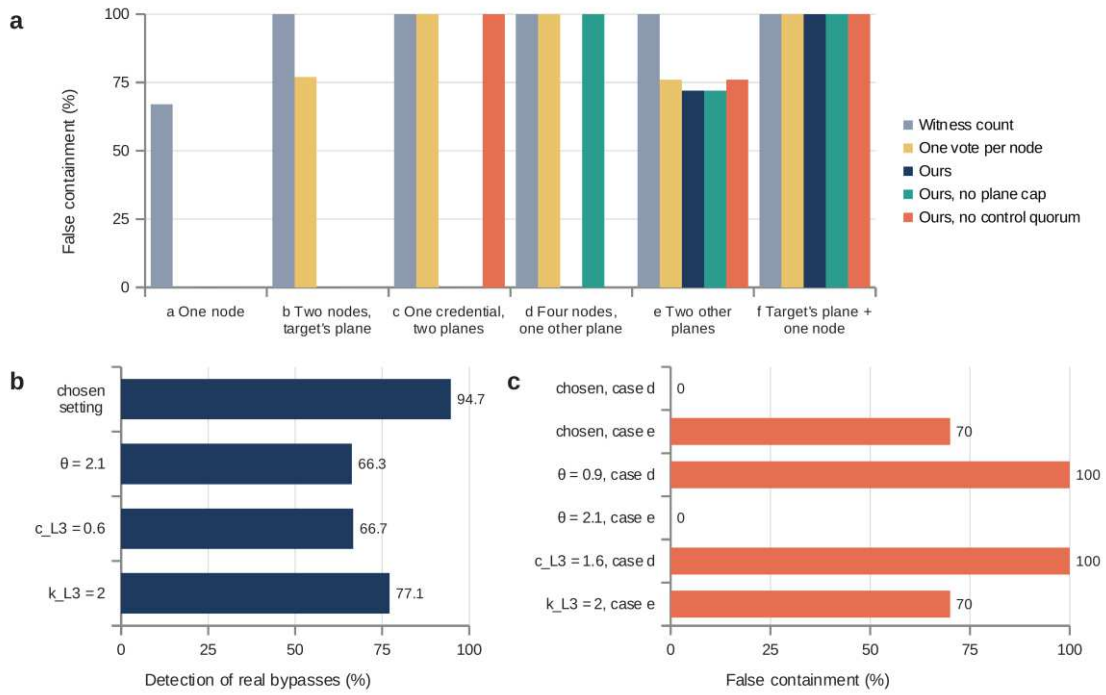


Fig. 4 Trigger. a False-containment rate by fabrication case and trigger variant (Table 9). b Detection rate of real bypasses and c false-containment rate for fabrication cases d and e at the threshold, cap and plane-quorum settings reported in Sect. 10.4, on the first 1,500 trials of the same random streams; the chosen setting is θ = 1.5, c_L3 = 1.0, k_L3 = 1

Resistance has a price in detections. For real bypasses with the target's management plane intact, witness counting triggers in 98.6% of 4,000 trials and the independence-aware trigger in 95.1%; when the plane is compromised and silences its agents, the figures are 80.3% and 73.0%. Most of the difference is the node quorum: in 22.3% of the trials with a compromised plane fewer than two independent node groups had observed the flow. The trigger then holds, and the contradiction between the plane's records and the flow log is raised as a separate signal instead of a containment.

Figures 4b and 4c show the settings of the two parameters that Proposition 1 constrains, and of the plane quorum, on the first 1,500 trials of each stream, which is why the values at the chosen point (70% for case e, 94.7% detection) differ slightly from Table 9 and the text above. With the cap at 1.0, any threshold above the cap stops the single-plane case; at θ = 0.9, single-plane fabrication triggered false containment in every trial. Raising the threshold further also suppresses the two-plane case (70% at θ = 1.5, 0% at θ = 2.1), but detection falls from 94.7% to 66.3%. With the threshold at 1.5, a cap above it readmits the single-plane case (100% at c_L3 = 1.6), and a cap of 0.6 loses detections (66.7%). Requiring two plane groups instead of one leaves the two-plane case unchanged at 70% and lowers detection from 94.7%

to 77.1%. The chosen values are therefore one admissible point, c_L3 < θ, at which detection is still close to its maximum. They are not optimal in any stronger sense.

A healthy workload can also lose its own records. When honest sources then report flows that the workload's records lack, the trigger without the attribution condition of Sect. 6 contains the workload in 100% of trials; with the condition, and a gap that is visible in 95% of losses, the rate is 4.3%.

10.5 Containment: false reports and residual exposure (RQ1, RQ2)

Table 10 Random mix (S0), five independent sets of 2,000 incidents (10,000 in total), each starting from a triggered target. False-effective count and rate with 95% Wilson interval, and its range over the five sets; declared states EFFECTIVE for Π / EFFECTIVE for a scope / UNRESOLVED; approvals, stops and avoidable stops per incident. The last column gives the closed-form expectation of the false-effective rate under the independent conditions of Table 7 where the failure condition does not depend on timing (Sect. 10.5); a dash means that it does

Config.	False-effective, k (%) [95% CI]	Range per set (k of 2,000)	Residual exposure	Declared (%)	Exposed min in 2 h	Time to EFFECTIVE, median / p95 (min)	Approvals	Stops	Avoid. stops	Closed form (%)
B1	6,997 (70.0%) [69.1–70.9]	1,382–1,411	70.0%	100.0 / 0.0 / 0.0	84.1	0.1 / 0.1	0.00	0.00	0.000	70.00
B2	7,561 (75.6%) [74.8–76.4]	1,484–1,531	75.6%	100.0 / 0.0 / 0.0	87.5	0.0 / 0.0	0.00	0.00	0.000	75.30
B3	7,540 (75.4%) [74.6–76.2]	1,478–1,527	75.6%	99.8 / 0.0 / 0.2	87.5	2.0 / 2.0	0.00	0.00	0.000	—
B4	4,621 (46.2%) [45.2–47.2]	893–956	46.2%	100.0 / 0.0 / 0.0	53.2	0.0 / 0.0	0.00	0.00	0.000	46.06
B5	3,397 (34.0%) [33.1–34.9]	648–695	34.0%	100.0 / 0.0 / 0.0	38.9	0.0 / 0.0	0.00	0.00	0.000	33.52
A1	575 (5.75%) [5.31–6.22]	108–126	9.8%	95.0 / 3.6 / 1.3	16.6	11.1 / 41.1	0.33	0.28	0.021	—
A2	3,085 (30.9%) [30.0–31.8]	602–638	33.3%	97.5 / 0.0 / 2.5	40.6	0.0 / 0.0	0.02	0.02	0.000	—
Full loop	12 (0.12%) [0.07–0.21]	1–5	7.1%	92.5 / 5.1 / 2.5	14.9	11.1 / 41.1	0.35	0.30	0.028	—

Table 10 answers the first half of RQ1 for the random mix. For the four configurations that accept an acknowledgement, the failure condition is a Boolean function of the conditions of Table 7 and the semantics of Table 2. Let est, hosts, v6 be the conditions that established connections exist, that other hosts are in the target's subnet and that the target has an IPv6 address, sg that the adversary can revert the security group (the target's role reaches its administration, reaches every role, or the plane credential can modify it and the plane is compromised), and acl that it can revert the network ACLs. Then B1 fails if the agent is impaired, B2 if est ∨ sg, B5 if (est ∨ sg) ∧ (hosts ∨ acl), and B4 if (est ∨ sg) ∧ (hosts ∨ acl ∨ v6). Treating the conditions of Table 7 as independent events and using their probabilities gives the expectations in the last column of Table 10 (the script is available with the other materials). Each expectation differs from the corresponding emulated rate by less than one Monte Carlo standard error ($|z| \le 0.95$). The semantics and the assumed frequencies thus

determine the expected rates of these four configurations, and the agreement of the emulated rates with the expectations is a check of the emulator's implementation, not a finding. The same expressions bound the effect of the runbook step that we did not measure (Sect. 4.1): if a two-step replacement ended every tracked connection, est would drop out of the conditions, and the expectations under Table 7 would be 38.26% for B2 and 17.59% for B5 instead of 75.30% and 33.52%, with reversion by the adversary as the remaining cause. The same expressions give the dependence on rule shape. If only a fraction f of the incidents with established connections have tracked ones, because the rules of the others leave them untracked [8] and the replacement ends them, the expectations are linear in f: (38.26 + 37.04 f)% for B2, (23.79 + 22.26 f)% for B4 and (17.59 + 15.93 f)% for B5, with f = 1 in Table 7. The same expressions bound what revoking the target's credentials would change. If revocation removed every reversion that goes through the target's role or a role reachable from it, sg would reduce to the management-plane condition and acl would vanish, and the expectations under Table 7 would be 64.00% for B2, 37.12% for B4 and 25.60% for B5. This is the optimistic end: a deny-all policy on the target's role does not end sessions of other roles that the target has already assumed, it leaves the management-plane credential untouched, and the contained principal can remove the policy during the delay before it takes effect, reported as about four seconds [65]; if revocation removed none of the reversions, the expectations are those of Table 10. We have not emulated revocation for any configuration. The sensitivity analysis of Sect. 10.7, which varies the share of incidents with established connections, shows the same dependence in the emulator. The rule shapes that leave a flow untracked are visible in the controller's configuration snapshot, although whether a flow is tracked automatically depends on the path to its peer [8], so the snapshot does not decide every case; the loop does not use them and treats an established flow as verified only when a stateless layer rejects it (Sect. 8). For B3, A1, A2 and the full loop, the outcome depends on the timing of reversion relative to verification, on approvals and on the delivery of flow records. The emulation captures these dependencies; we have not derived the corresponding rates in closed form. Accepting the acknowledgement for an isolation security group (B2) gave a false-effective report in 75.6% of incidents. With a fixed probe (B3), the rate was 75.4% because the probe tests one new connection across subnets, which the security group blocks until it is reverted. Adding network ACL entries lowered the rate to 46.2% for IPv4 only (B4) and to 34.0% for both families (B5); the remaining third are incidents with established connections inside the target's subnet, which no network ACL filters, or with an adversary that reverts the entries. The full loop reported falsely in 12 of 10,000 incidents (0.12%; one to five per set), in all of

them because the access analysis had missed an edge and the adversary reverted late, the case that Proposition 2 excludes by assumption. This outcome is partly built into the emulator: the adversary's ability to revert an enforcer and the controller's computed closure come from the same reachability relation and differ only by the injected miss, so the loop can fail only through that miss combined with late reversion. Its rate is therefore a consequence of the assumed miss rate, not an independent finding; an adversary model computed separately from the controller's analysis, for example by evaluating real access policies, has not been tested. Of its residual exposure of 7.1% (707 incidents), 697 incidents were declared UNRESOLVED (246) or EFFECTIVE for a scope only (451) after an approval was denied or timed out, so 98.6% of the remaining exposure was reported to the operator; the other 10 are false reports. The 12 false-effective incidents and these 10 differ by definition: objective (1) counts a declared path that was open at any time in the window, residual exposure only a path open at its end, so by the definitions each of the 10 is among the 12, and the other 2 had a declared path open during the hour but not at its end. We have not tabulated the two measures at incident level. Mean exposed time fell from 87.5 minutes for B2 and 38.9 for B5 to 14.9.

The ablations answer RQ2. Selection without verification (A2) still reported falsely in 30.9% of incidents: it relies on enforcers whose policy the adversary cannot revert but claims paths that those enforcers do not cover, such as established connections inside the subnet. Verification without selection (A1) reported falsely in 5.75%: verification is a check at one point in time, and an adversary that waits until it has ended is not seen. On the same incidents A1 gave a false report in 563 incidents in which the loop did not and the reverse never occurred (paired difference 5.63 percentage points, 95% interval 5.19–6.10 by Newcombe's method for paired proportions [66]; exact McNemar test, two-sided p = 2^-562); against B5 the counts are 3,385 and 0 (33.85 points, 32.93–34.78). These intervals and p-values quantify only the Monte Carlo error of one generating process with the conditions of Table 7: they treat incidents as independent draws and reflect how many we drew, not the uncertainty of the assumed conditions or of any real incident population. Paired counts per set and an analysis that treats the five sets as replicates are not reported; Sect. 10.7 varies the conditions only one at a time. Among incidents with an EFFECTIVE declaration the false-effective rate was 0.12% for the loop and 5.8% for A1.

The rules introduced in Sects. 3.3 and 8 did not change any report in the random mix: the loop with the earlier rules gave the same 12 false reports on the same incidents and the same declared states, and A1 with the earlier rules gave 604 instead of 575. In the deployment of

Sect. 4.1 the network ACL entries sit in the peers' subnets and the security group at the target's interface, so the recording interface already separates the layers. The rules matter where this does not hold, which S12 and S13 test.

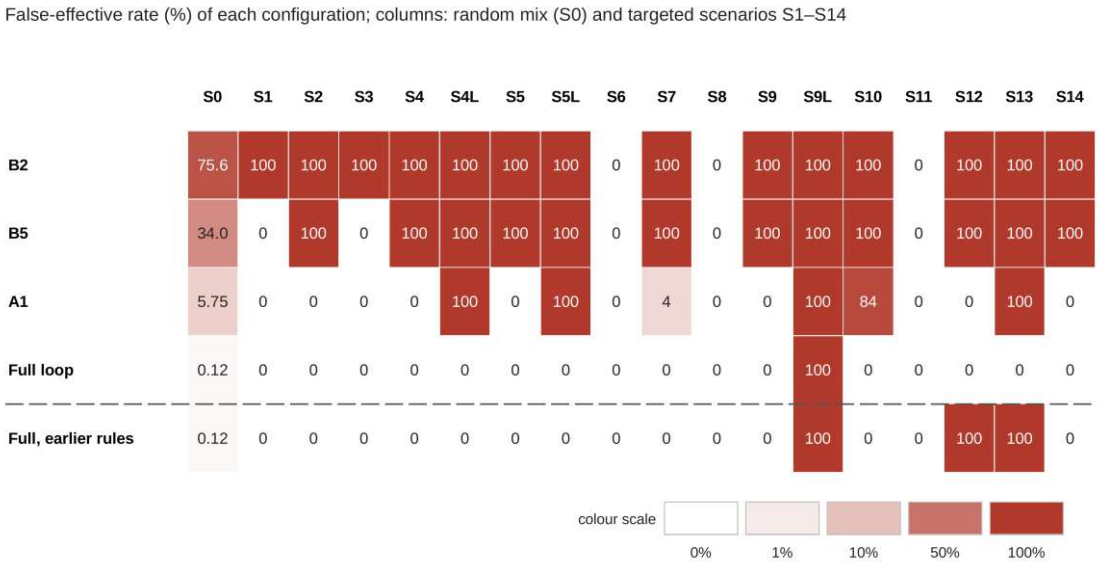


Fig. 5 False-effective rate by configuration and scenario (S0 random mix, 10,000 incidents; S1–S14 as in Table 11, 2,000 incidents each). The last row uses the earlier attribution and class rules

Table 11 Targeted scenarios, 2,000 incidents each: false-effective rate of B2, B5, A1, the full loop and the full loop with the earlier rules; declared states, stops per incident and residual exposure of the full loop. Declared states EFFECTIVE for Π / EFFECTIVE for a scope / UNRESOLVED, rounded to integers

Scenario	Condition	B2	B5	A1	Full	Full, earlier rules	Full declared (%)	Full stops	Full residual
S1	Established connections to other subnets; no hosts in the target's subnet	100%	0%	0%	0%	0%	100 / 0 / 0	0.00	0%
S2	Established connections inside the target's subnet	100%	100%	0%	0%	0%	84 / 16 / 0	0.84	16%
S3	IPv6 and established connections; no hosts in the target's subnet	100%	0%	0%	0%	0%	100 / 0 / 0	0.00	0%
S4	Target's role reaches security-group administration; early reversion	100%	100%	0%	0%	0%	85 / 15 / 0	0.85	15%
S4L	As S4, strategic adversary	100%	100%	100%	0%	0%	85 / 15 / 0	0.85	15%
S5	Compromised management plane can modify security groups; early reversion	100%	100%	0%	0%	0%	85 / 15 / 0	0.85	15%
S5L	As S5, strategic adversary	100%	100%	100%	0%	0%	85 / 15 / 0	0.85	15%
S6	Management plane could modify security groups but is intact	0%	0%	0%	0%	0%	86 / 14 / 0	0.86	0%
S7	As S4, and the target answers nothing	100%	100%	4%	0%	0%	85 / 15 / 0	0.85	15%
S8	Half of the intervals delayed further, one in five skipped	0%	0%	0%	0%	0%	100 / 0 / 0	0.02	0%
S9	As S4, and the access analysis misses the edge	100%	100%	0%	0%	0%	85 / 0 / 15	0.85	15%
S9L	As S9, strategic adversary (completeness assumption of Proposition 2 does not hold)	100%	100%	100%	100%	100%	100 / 0 / 0	0.00	100%
S10	Target's role reaches every role	100%	100%	84%	0%	0%	0 / 0 / 100	0.00	100%
S11	No verifier interface in the target's subnet	0%	0%	0%	0%	0%	85 / 15 / 0	0.85	0%
S12	Second other subnet with an ACL the containment role cannot change; tracked connections	100%	100%	0%	0%	100%	83 / 17 / 0	0.83	17%
S13	Security-group change not applied; target denies the verifier in its subnet's ACL, silent	100%	100%	100%	0%	100%	86 / 0 / 14	0.86	14%
S14	Security-group change acknowledged but not applied	100%	100%	0%	0%	0%	85 / 0 / 15	0.85	15%

The targeted scenarios (Table 11, Fig. 5) show the mechanisms. S1–S3 are pure semantics. Established connections defeat every configuration that relies on the security group alone; IPv6 defeats B4 but not B5; and in S2 no non-disruptive enforcer covers established connections inside the subnet, so the loop asks for a stop and, when it is not approved, declares the other path classes only (16% residual exposure, none of it reported as contained).

S4–S5L isolate the effect of selection. With early reversion, verification without selection (A1) observes the reversion and escalates, and its false-effective rate is 0%. With strategic reversion, A1 gave a false report of containment in every incident of S4L and S5L, and the full loop in none: it never relied on the security group, treated the subnet-internal paths as uncovered, and asked for a stop. S7 adds a hung service. The fixed probe of B3 reads the

silence as success in every incident; A1 depends on accepted-traffic records and probes and missed 4% (34% with $\Delta t = 60$ s, because a shorter wait more often places all probes before an early reversion); the full loop is unaffected because it does not trust the enforcer whose policy the adversary reverts in the first place.

S12–S14 isolate the rules of Sects. 3.3 and 8 and the case of an acknowledged change that never takes effect. In S12 the earlier rules extend the result for the probed subnet to the second subnet, whose established connections stay open, and report falsely in every incident; with class splitting the second subnet is a class without a verifier, so the loop leaves it unverified and asks for a stop. In S13 the earlier rules count the rejection that the target's own network ACL entry produces at the target's interface as a rejection by the security group and report falsely in every incident; the attribution rule finds a second layer in the snapshot, the probe class stays INCONCLUSIVE, and the loop escalates. A1 fails in S13 as well, because the target can also remove the network ACL entries that A1 relies on. In S14 the probes are answered, both verifying configurations record FAILED rounds and escalate, and every acknowledgement-based configuration reports falsely.

Three scenarios show limits. In S9 the access analysis misses the edge that lets the target revert the security group. With early reversion, verification catches the consequence (0% false reports); with strategic reversion (S9L) the loop gives a false report in 100% of incidents; the completeness assumption of Proposition 2 does not hold here, so the proposition gives no guarantee. In S10 no independent enforcer exists; the loop declares UNRESOLVED at once in every incident, whereas A1 asks for approvals and stops instances that the adversary restarts (0.84 stops per incident) and reports falsely in 84%. S8 shows that degraded records delay but do not mislead: 1,994 of 2,000 incidents still end EFFECTIVE for Π and 6 for a scope, with a median of 20.2 minutes instead of 11.1.

10.6 Integrated run and operational cost (RQ3)

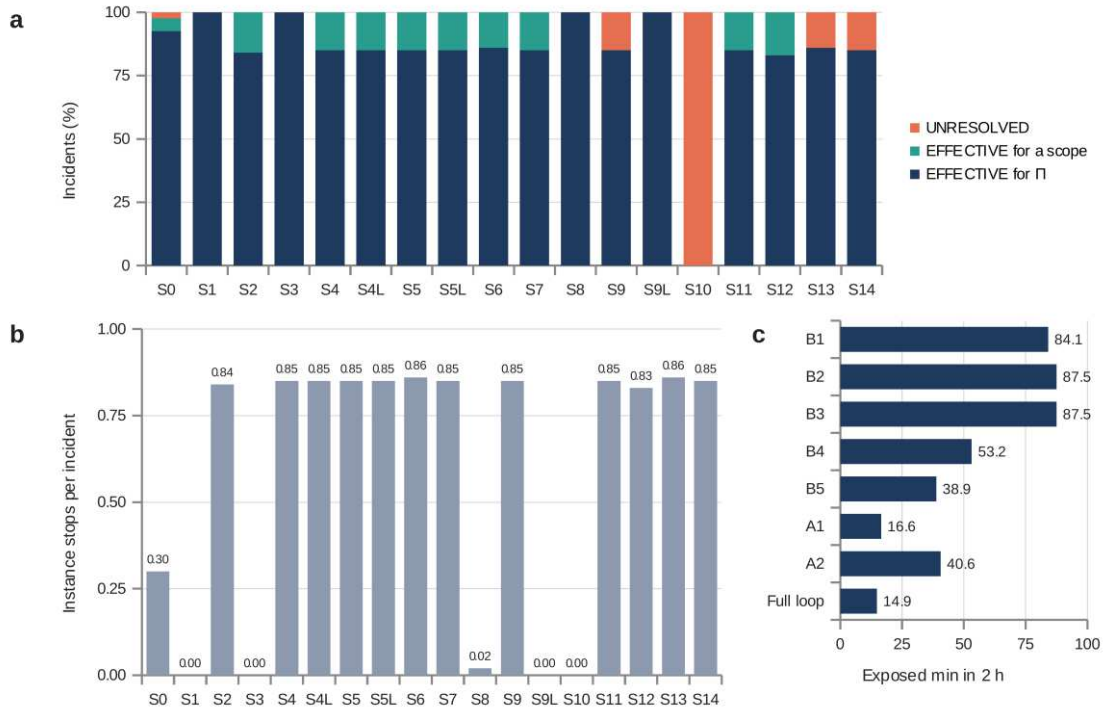


Fig. 6 a Declared outcome of the full loop in the random mix (S0) and in each targeted scenario, percent of incidents, and b instance stops per incident (Tables 10 and 11). c Mean exposed minutes in the two hours after detection, random mix (Table 10)

Table 12 Integrated run from trigger to declaration on the first set of S0 incidents (2,000 real bypasses). The trigger of Sect. 6 decides first; held incidents are not contained

Configuration	Held by the trigger	Contained: EFFECTIVE for Π / for a scope / UNRESOLVED	False-effective	Residual exposure	Exposed min in 2 h
B2	304	1,696 / 0 / 0	1,256 (62.8%)	1,560 (78.0%)	90.8
B5	304	1,696 / 0 / 0	553 (27.7%)	857 (42.9%)	49.8
A1	304	1,615 / 66 / 15	103 (5.2%)	465 (23.3%)	31.6
Full loop	304	1,572 / 85 / 39	1 (0.05%)	411 (20.6%)	29.9

Table 12 combines the two experiments. The trigger held 304 of 2,000 real bypasses (15.2%), 245 of them (81%) in incidents whose management plane was compromised and silenced its agents (Sect. 10.4); these are not contained by any configuration that waits for the trigger, and they dominate the residual exposure of the loop from trigger to declaration (304 of 411 incidents). Among the triggered incidents the loop behaved as in Table 10. The integrated run uses only the first of the five sets; the other four have not been run from trigger to declaration, and none of it was run in a live cloud. The end-to-end view therefore separates two management questions: whether containment starts, which is a detection trade-off controlled by the quorum parameters, and whether a containment that started is reported correctly, which the loop addresses.

The loop is slow by the standards of inline prevention. In the random mix the median time to an EFFECTIVE declaration was 11.1 minutes and the 95th percentile 41.1 minutes (Table 10). The median is one verification round, dominated by Δt and the wait for flow records; the tail is operator response time for approvals. Blocking itself is not delayed: enforcers are deployed at detection, and only the declaration waits; Fig. 6c shows the resulting exposed minutes. These times rest on the record delay we measured. With a simulated delay of nine to eleven minutes around the documented typical value of about ten minutes [10] instead, the median rises to 17.0 minutes and the 95th percentile to 47.0 minutes, and the loop gave 1 false report in 2,000 incidents against 116 for A1.

The larger cost is disruption. The loop requested 0.35 approvals per incident and stopped the instance in 30.1% of incidents (3,005 of 10,000), against 27.6% for A1 and none for the acknowledgement-based configurations. These figures count stops only; the loss of volatile evidence and the replacement of an Auto Scaling member that was not moved to standby (Sect. 9) are not included. Most stops were needed in the sense that no non-disruptive means could close or verify the remaining paths: 2.8% of incidents ended in a stop that was avoidable in hindsight. Figure 6b and Table 11 locate the avoidable ones: S6 and S11 end in a stop about as often as the scenarios with a capable adversary, although nothing would have been reverted. In S6 the management plane could have modified the security group but was not compromised; the loop cannot know that, excludes the group, and asks for a stop to cover the subnet-internal paths (0.86 avoidable stops per incident, without added exposure in the meantime, because the group is still applied as best effort). In S11 the verifier has no interface in the target’s subnet and the same happens (0.85).

Both causes are in the hands of the platform administrator of Sect. 3.1, and the model makes them visible before any incident: removing the permission to modify security groups from the agent management server, separating instance roles from network administration, and giving the verifier an interface in each protected subnet each turn a stop into a non-disruptive containment. The avoidable stops in the emulated scenarios S6 and S11 suggest that the compromise relation could also support an audit, outside incident response, of how readily each workload can be contained. We have not run such an audit on a real or public account configuration; what is shown here is only that the stops in these two emulated scenarios were avoidable.

Table 13 Management overhead per incident in the random mix (means over 10,000 incidents)

Configuration	Verification rounds	Management-plane calls	Probe connections	Record queries	Configuration snapshots
B2	0.00	1.0	0.0	0.0	0.0
B5	0.00	8.8	0.0	0.0	0.0
A2	0.00	14.2	0.0	0.0	0.0
A1	1.12	16.6	7.2	23.2	2.2
Full loop	0.99	15.2	5.9	21.7	2.0

Table 13 gives the overhead in the controller's own units. An incident handled by the full loop used on average 15 management-plane calls, 6 probe connections, 22 record queries and 2 configuration snapshots, against one call for B2 and 9 for B5. These volumes are small against provider rate limits; we do not convert them to money because prices and free allowances vary by region and over time.

10.7 Sensitivity to the assumed conditions

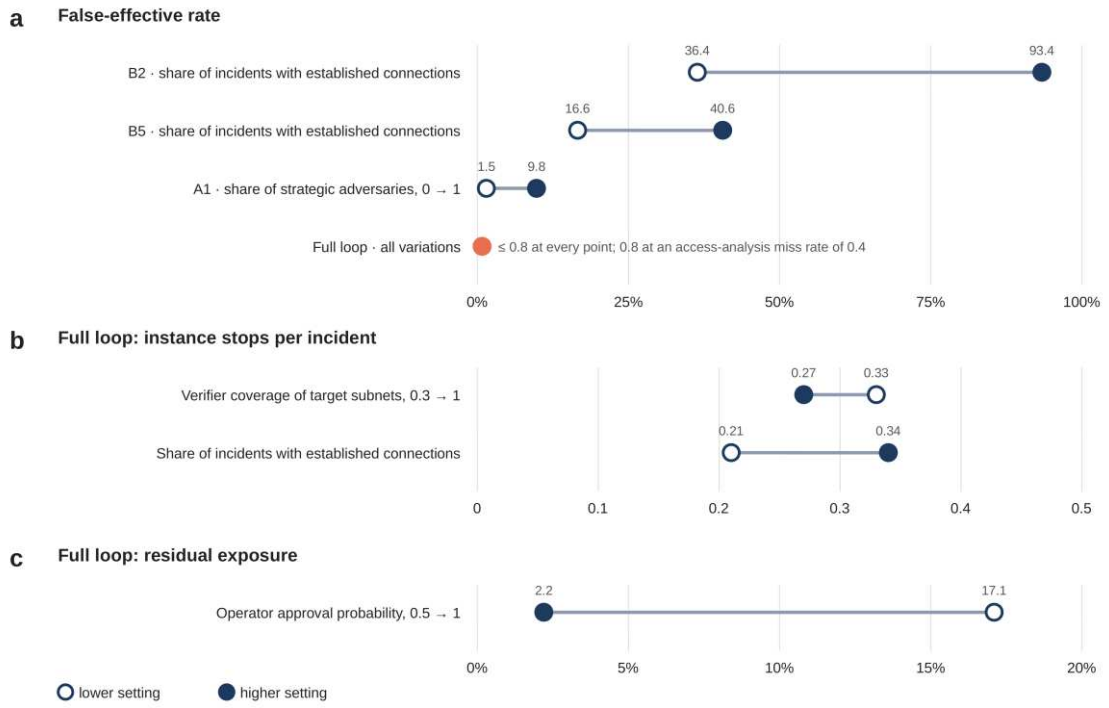


Fig. 7 One-at-a-time variation of conditions of the random mix, 1,000 incidents per point: values reported in Sect. 10.7 at a lower and a higher setting of each varied condition. a False-effective rate. b Instance stops per incident and c residual exposure of the full loop

Because the probabilities of Table 7 are assumed, Fig. 7 varies six of them one at a time with 1,000 incidents per point. At every point the false-effective counts were ordered full loop ≤ A1 ≤ B5 ≤ B2. The rate of B2 moved between 36.4% and 93.4% with the share of incidents that have established connections, which is the condition it is most sensitive to, and that of B5 between 16.6% and 40.6%. The rate of A1 rose from 1.5% to 9.8% as the share of strategic

adversaries went from 0 to 1. The rate of the full loop stayed at or below 0.8% everywhere and reacted only to the miss rate of the access analysis (0.8% at a miss rate of 0.4).

The cost side is more sensitive. Stops per incident fell from 0.33 to 0.27 as verifier coverage of target subnets rose from 0.3 to 1, and rose from 0.21 to 0.34 with the share of incidents that have established connections. When operators approve only half of the requests, residual exposure of the full loop is 17.1%; when they approve every request, it is 2.2%. The absolute numbers in Tables 10–12 should therefore be read as conditional on Table 7; the comparisons between configurations on the same incidents are what the emulation supports.

10.8 Computation at scale (RQ4)

The relation is evaluated against the target only, and control groups are formed with a union–find structure keyed by principal, in time close to linear in the total number of principals over all sources and memory proportional to the number of distinct principals, so the trigger needs no pairwise matrix over the evidence sources. On one core of a commodity virtual machine, in unoptimised Python 3.13.18, evaluating share, grouping and scoring took a median of 0.9 ms for 1,000 sources and 204 ms for 100,000. The role-assumption closure took 5.4 ms on a graph of 10,000 roles and 30,000 edges and 150 ms on 100,000 roles, and selection over 10,000 candidate enforcers took 6.2 ms. Computation is negligible against the minutes that record delivery takes. The cost that these timings do not include is obtaining the policy data: evaluating conditions on real access policies requires the provider’s policy evaluation interface or an offline reasoner [45], and its precision is the assumption on which Proposition 2 rests.

11 Discussion, Limitations and Future Work

11.1 Limitations

- **Emulation.** Only the provider semantics and the record delay were measured on a live system. Incident conditions, adversary timing and operator behaviour are modelled, the frequencies in Table 7 are assumptions, and the absolute rates of Sect. 10 hold only under them; what the emulation supports are comparisons between configurations on the same incidents. The emulator treats UDP and ICMP like TCP after a deny entry is removed, does not model routing or baseline probing, and knows the path classes that a real controller has to discover. The controller itself has been exercised only inside the emulator.

- **Trigger and containment together.** In the run from trigger to declaration, the trigger held 15.2% of real bypasses, most of them with a compromised management plane, and these incidents dominate the remaining exposure (Table 12). Containment assurance does not help with bypasses that never start containment; that is a detection trade-off governed by the quorum parameters (Sect. 10.4).
- **Scale of the live measurements.** The runs of Sects. 4.2–4.4 used one region and one account on an almost idle network: three instances of one family for less than an hour, ten instances of two families for about 90 minutes, and three instances for the attribution run. The delivery measurement of Sect. 4.5 added a second region and ran for 72 hours, with two instances per region in the same account and also at low traffic; it measured delivery only, not enforcement. Record delivery under load or during provider incidents may differ, which is why the loop treats delivery time as unbounded and S8 tests long delays. The loop itself has not yet been run against a live adversary.
- **Two-step group replacement.** The provider's runbook applies a temporary group that allows all inbound traffic before the containment group [57]. Untracked flows are documented to stop when the rule that allows them is removed [8], and in our runs flows that were untracked from the start did stop (Table 4), but we did not measure whether the intermediate step ends connections that were already tracked, and neither B2 nor the loop uses it. If it does, acknowledgement-based containment fails less often than Table 10 reports (Sect. 10.5), and where the security group is a selected enforcer the loop could close established connections inside the target's subnet without a stop. Reversion of the group by the adversary, attribution and the need for verification are unaffected.
- **Attribution.** Flow records do not name the rejecting layer. The snapshot condition of Sect. 8 keeps an unattributable rejection from counting, at the price of more INCONCLUSIVE rounds in estates where several layers filter the same path; we have not measured how often that occurs in real estates. In the emulation the rule changed no outcome of the random mix and mattered only in the constructed case S13 (Sect. 10.5); we therefore present it as a safeguard, not as a source of the reported rates. The conditions that the rule and Proposition 3 rest on, a correct snapshot, complete configuration change events and a sufficient consistency margin, have not been tested by a live negative control in which a second filtering layer, a racing change or a stale snapshot is introduced deliberately.

- **Soundness of the access analysis.** Proposition 2 holds only for a complete C(X). A missed edge combined with late reversion defeats the loop (S9L). The closure follows role assumption and treats unevaluable conditions as satisfied; we did not test it on a real account against service control policies, permission boundaries, resource-based policies or session tags. That such edges are easy to miss is visible in a test account we prepared: the provider's policy simulator reported an implicit deny for the action that changes the security groups of an interface when asked about a workload role, although the trust policy of a second role, which holds that permission, names the workload role as a principal that may assume it. An analysis that evaluates only the policies attached to the workload role does not see this path. Escalation paths other than role assumption, such as passing a role to a launched resource or writing a policy [59], are handled by the rules of Sect. 5, which the reference implementation does not contain; Table 7 does not sample a target role with such permissions, and in the emulator its effect is represented only by the condition that the target's role reaches every role, which the loop answers with UNRESOLVED (S10). How often real accounts contain such paths, and therefore how often the rule turns an incident into UNRESOLVED, we have not measured. Re-associating a subnet with another network ACL is, in the emulator, one instance of the single abstract capability to revert an ACL enforcer, so it has the same effect there as removing the entries; the modifier computation of Sect. 5 over-approximates by evaluating every ACL of the virtual network, and we have not checked it against the provider's policy evaluation in a real account. Proposition 2 also assumes that the adversary gains no principal beyond C(X) and the plane credential. An adversary that escapes to a node shared with other workloads, or that runs commands through a compromised management plane, obtains the principals of those workloads; extending the selection condition to $C(e) \cap A(X) = \emptyset$ is direct, but it has not been implemented or evaluated, and no scenario of Table 11 contains such a workload. Planner-based analyses of real access policies [48] and privilege-escalation analysers [46, 47] are ways to obtain a closure against which ours could be compared; we have not made that comparison. Because C(p) joins the principals reachable from p with those that may modify p, an operations role that may modify both X and an enforcer, and that X cannot reach, still excludes the enforcer; this errs towards UNRESOLVED and stops, and a selection condition on the principals reachable from X alone would avoid it, which we have not evaluated. Proposition 2 further covers the selected enforcers' policies and bindings but not

changes of the target's addresses or interfaces, including an interface attached while the security group is trusted (Sect. 7), which neither the implementation nor the emulator handles.

- **Credential revocation.** The provider also contains access-management principals with a deny-all policy and treats revoking credentials as short-term containment [5]; our baselines and the loop contain the network only. Revocation shrinks the principals available to the adversary, A(X) in Proposition 2, and would lower the baselines' rates (Sect. 10.5), but it does not reach sessions of other roles already assumed, the management-plane credential, or a principal that removes the policy before it takes effect [65], so which enforcer can be trusted is still decided by selection.
- **More than one compromise.** Fabricated evidence from two management planes, or from the target's plane and one further node, triggers containment of a healthy workload (Table 9, e and f). A higher threshold suppresses the first case at a cost in detections (Fig. 4).
- **Trusted base.** The provider's control plane, its flow-log service and account B are trusted. A principal allowed to delete or change the flow log, account administration of A among them, can stop future records. If such a principal is reachable from X, the flow log shares compromise with X, through a common control principal or, for account administration, through the account rule of Sect. 5; it is then not admitted as observer (Sect. 7), and the incident ends UNRESOLVED; otherwise the controller detects missing intervals and cannot declare EFFECTIVE, because silence is never success (Sect. 8), but it cannot verify either. Table 7 does not sample a target role that can change flow logs.
- **Observation gaps.** Flow logs do not record some traffic, for example to the instance metadata service and the provider's name service [67], and security groups and network ACLs do not apply to queries to that name service [68]; a declaration therefore says nothing about name resolution as a channel, which has to be closed by other means that we did not evaluate. Probes cannot test connections that already exist. Results are stated for a scope, and what lies outside is escalated, not assumed blocked. Paths that leave the virtual network are outside Π altogether (Sect. 3.3): an established session to an external peer survives the isolation security group, and the deny entries of Sect. 4.1 sit only in other subnets, so a deployment that faces the internet needs at

least a stateless entry in the target's own subnet and a verifier outside the network, which we have not evaluated.

- **Disruption.** The loop stops instances that a less careful process would leave running (30% of incidents in the random mix). Whether that is acceptable depends on the workload; the approval gate leaves the decision with the operator, and the permission changes named in Sect. 10.6 reduce the need. We did not measure the service impact of a stop or of a routing change on other workloads, and Tables 10–13 do not count two further costs of a stop: the replacement of an Auto Scaling member that is not moved to standby first, and the loss of volatile evidence that is not captured before the stop (Sect. 9). Nor did we compare against a policy that stops every triggered instance at once; without it, the evaluation does not show how much of the loop's value lies in fewer false reports and how much in fewer stops than such a policy.
- **Network structure and rule quotas.** The random mix has one other subnet with a verifier interface (Sect. 10.1), whereas real virtual networks span several subnets and zones. Every subnet whose network ACL differs forms a class of its own and needs a verifier interface, which costs interfaces or instances that Table 13 does not count. Network ACLs have a default quota of 20 rules per direction, adjustable to at most 40 [69], and the deny entries of Sect. 4.1 are written with a lower rule number than existing entries, which is impossible when rule number 1 is in use or the quota is reached, for example in a network ACL shared by concurrent incidents. Rule quotas and numbering are not part of the emulated semantics of Table 2. An entry that cannot be written has to count as an enforcer that was not deployed, so that its classes stay unverified and escalate; we have not tested this case.
- **Latency.** Decisions take minutes because records do. The loop assures a management action and does not replace inline enforcement.
- **Portability.** Other providers offer flow logs and filtering primitives with different state handling, aggregation and delivery. We expect the model and the loop to carry over but have not tested this; Table 2 and its measurements would have to be redone for each provider.
- **Record matching.** The trigger takes attributed mismatches as input. How records of heterogeneous points are matched, and how often benign causes produce mismatches in production, are not evaluated here.

11.2 Future work

The limitations above define the work that remains. We list it in the order in which we expect it to change the conclusions of this article.

1. **End-to-end runs in a live cloud.** Running the controller against a reverting adversary on labelled cases, in accounts whose permission structure is taken from practice rather than from Table 7, would replace the assumed incident conditions by measured ones. The same runs can serve as the negative controls that the attribution rule still lacks: a second filtering layer at the recording interface, a configuration change that races the decision, and a stale snapshot.

2. **The provider's two-step group replacement.** A measurement of whether the temporary group of the provider's runbook [57] ends connections that are already tracked would settle the largest open term in the closed-form expectations of Sect. 10.5, and, if it does end them, would let the loop close established connections inside the target's subnet without a stop.

3. **A complete and checked access analysis.** Implementing the escalation edges and the account rule of Sect. 5, extending the adversary's principals A(X) to co-located workloads and to the workloads of a compromised management plane, and comparing the computed closure with privilege-escalation analysers [46, 47] and planner-based analysis [48] on real account configurations would test the assumption on which Proposition 2 rests, and would give an adversary model that is computed separately from the controller's own analysis.

4. **Coverage after the declaration.** Adding the condition $M(X) \cap A(X) = \emptyset$ to selection, or invalidating a declaration on any change of the target's addresses or interfaces, together with periodic re-verification, would close the gap of Sect. 7 and bound how long a declaration that has become false can stand.

5. **Richer estates and costs.** A random mix with several subnets and zones, network ACL quotas and rule numbering, Auto Scaling membership and capture of volatile evidence before a stop, together with a baseline that stops every triggered instance at once, would show what the loop costs in realistic estates and how much of its value lies in fewer stops.

6. **Paths beyond the virtual network.** Covering paths that leave the virtual network, with a stateless entry in the target's subnet and a verifier outside the network, and closing name resolution as a channel, would remove the largest exclusion from the scope Π.

7. **Credential revocation combined with network containment.** Revoking the target's credentials first and computing A(X) afterwards, and emulating the effect of revocation on every configuration, would replace the closed-form bounds of Sect. 10.5 by measured rates.

8. **Generality of the measurements.** Repeating the measurements of Sect. 4 under load, in further regions, with other default timeouts and on other providers, and evaluating record matching on production records, would show how far Table 2 and the trigger's inputs carry over.

12 Conclusion

Containment in a zero trust cloud is a management action whose outcome should be verified and reported with its scope, not inferred from an acknowledgement. Live measurements showed why. As the provider documents, tracked flows survived the isolation rule, here for three protocols that exchanged data every 5 s, in every unit of five repetitions on two architectures. Meanwhile a configuration analysis reported no path, a route change left no rejection record, and records did not separate two filtering layers at the same interface; over 72 hours in two regions, a few record deliveries took more than a quarter of an hour. We proposed one compromise model, computed from placement, management planes and access-policy reachability, that decides which evidence may start containment, which enforcers' policies the target cannot revert, and which verifiers and observers may confirm the result, and we argued that a rejection should count only when the configuration attributes it to a selected enforcer. In an emulation of the measured semantics, under assumed conditions and starting from a triggered target, the loop reported containment falsely in 12 of 10,000 incidents. In each of them an injected miss in the access analysis coincided with late reversion, a combination that the emulator admits by construction. The strongest configuration that accepts acknowledgements reported falsely in 34.0% of incidents, a rate within one Monte Carlo standard error of its closed-form expectation under the assumed conditions (33.52%), and verification without selection in 5.75%. None of these configurations revoked the target's credentials; under the same conditions revocation would

put the expectation for the strongest one between 25.60% and 33.52%, depending on how many reversions it removes. Selection was what prevented false reports against an adversary that reverts after verification (S4L, S5L), and the attribution and class rules prevented them in two constructed scenarios in which two layers filter the same interface or a subnet has no verifier (S12, S13). In the random mix these rules changed no outcome, so they are a safeguard whose value depends on how often such configurations occur, which we have not measured. The price was a decision time of minutes and an instance stop in 30% of incidents; in the emulated scenarios S6 and S11, permission changes that the model exposes would have made the stops unnecessary. The evaluated controller follows role assumption only; the further escalation rules and the account rule of Sect. 5 are specified but not evaluated. The loop fails when the access analysis is unsound and the adversary reverts late, and when evidence is fabricated from two separately compromised domains. The rates depend on the assumed incident conditions, and the controller has yet to be run against labelled cases in a live cloud. Section 11.2 lists the work that remains, beginning with such runs against a live adversary and a measurement of the provider's two-step group replacement.

Statements and Declarations

Funding No funding was received for conducting this study.

Competing interests The author is employed by ESTsecurity Corp., a cybersecurity company. The method described in this article originated in an invention disclosure that the author prepared for a patent application. No patent application covering the method has been filed. The company had no role in the design of the study, in the collection, analysis and interpretation of data, in the writing of the manuscript, or in the decision to submit it. The author has no other competing interests to declare.

Author contributions The sole author conceived the study, designed the method, the measurements and the experiments, reviewed the code, the runs and their outputs, and wrote and approved the manuscript. The measurement and emulation runs were executed by a generative AI assistant as described in Sects. 4.1 and 10.1 and in the statement on generative AI below.

Data availability The measurement scripts, phase timestamps, aggregated flow-log results and console outputs of the live measurements, the reference implementation and the emulator, and the result files used to generate the tables and figures are available from the

corresponding author on reasonable request. Account identifiers are removed from the shared records.

Use of generative artificial intelligence The author used Claude Opus 5.5 (Anthropic) to search the literature, implement the emulator, the reference implementation and the measurement scripts, run the live measurements through the provider's scripting interface, and draft the text and the figures. The author used GPT-6.1 Sol (OpenAI) as an independent reviewer of drafts: it checked claims against the cited sources, the consistency of numbers between text and tables, and the language, and its comments were addressed in later drafts. Neither tool is an author of this article. The author reviewed and edited all content and takes full responsibility for it.

Ethics approval Not applicable. The study involved no human participants or animals; the live measurements used only resources in the author's own cloud account.

References

1. Rose S, Borchert O, Mitchell S, Connelly S (2020) Zero trust architecture. NIST Special Publication 800-207. National Institute of Standards and Technology, Gaithersburg. https://doi.org/10.6028/NIST.SP.800-207
2. Chandramouli R, Butcher Z (2023) A zero trust architecture model for access control in cloud-native applications in multi-location environments. NIST Special Publication 800-207A. National Institute of Standards and Technology, Gaithersburg. https://doi.org/10.6028/NIST.SP.800-207A
3. Nelson A, Rekhi S, Souppaya M, Scarfone K (2025) Incident response recommendations and considerations for cybersecurity risk management: a CSF 2.0 community profile. NIST Special Publication 800-61 Revision 3. National Institute of Standards and Technology, Gaithersburg. https://doi.org/10.6028/NIST.SP.800-61r3
4. Amazon Web Services (n.d.) Source containment. AWS Security Incident Response Guide (AWS whitepaper). https://docs.aws.amazon.com/whitepapers/latest/aws-security-incident-response-guide/source-containment.html. Accessed 16 November 2026
5. Amazon Web Services (n.d.) Contain. AWS Security Incident Response User Guide. https://docs.aws.amazon.com/security-ir/latest/userguide/contain.html. Accessed 16 November 2026
6. Gambo ML, Almulhem A (2026) Zero trust architecture: a systematic literature review. J Netw Syst Manag 34(1):25. https://doi.org/10.1007/s10922-025-09998-x
7. Syed NF, Shah SW, Shaghaghi A, Anwar A, Baig Z, Doss R (2022) Zero trust architecture (ZTA): a comprehensive survey. IEEE Access 10:57143–57179. https://doi.org/10.1109/ACCESS.2022.3174679
8. Amazon Web Services (n.d.) Amazon EC2 security group connection tracking. Amazon EC2 User Guide. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-connection-tracking.html. Accessed 12 November 2026

9. MITRE Corporation (n.d.) Disable or modify tools, technique T1685. MITRE ATT&CK v20. https://attack.mitre.org/techniques/T1685/. Accessed 12 November 2026
10. Amazon Web Services (n.d.) Flow log records. Amazon VPC User Guide. https://docs.aws.amazon.com/vpc/latest/userguide/flow-log-records.html. Accessed 10 November 2026
11. Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36(1):41–50. https://doi.org/10.1109/MC.2003.1160055
12. Inayat Z, Gani A, Anuar NB, Khan MK, Anwar S (2016) Intrusion response systems: foundations, design, and challenges. J Netw Comput Appl 62:53–74. https://doi.org/10.1016/j.jnca.2015.12.006
13. Nespoli P, Papamartzivanos D, Gómez Mármol F, Kambourakis G (2018) Optimal countermeasures selection against cyber attacks: a comprehensive survey on reaction frameworks. IEEE Commun Surv Tutor 20(2):1361–1396. https://doi.org/10.1109/COMST.2017.2781126
14. Islam C, Babar MA, Nepal S (2019) A multi-vocal review of security orchestration. ACM Comput Surv 52(2):37. https://doi.org/10.1145/3305268
15. Huang Z, Robin J, Herbaut N, Ben Rabah N, Le Grand B (2026) Toward an intent-based and ontology-driven autonomic security response in security orchestration automation and response. In: Enterprise design, operations, and computing (EDOC 2025). Lecture Notes in Computer Science, vol 16213. Springer, Cham, pp 266–283. https://doi.org/10.1007/978-3-032-15140-7_15
16. Jahan S, Riley I, Walter C, Gamble RF, Pasco M, McKinley PK, Cheng BHC (2020) MAPE-K/MAPE-SAC: an interaction framework for adaptive systems with security assurance cases. Future Gener Comput Syst 109:197–209. https://doi.org/10.1016/j.future.2020.03.031
17. Robles-Enciso A, Bernabé Murcia JM, Molina Zarca A, Skarmeta Gomez A (2025) Dynamic multi-method allocation for intent-based security orchestration. J Netw Syst Manag 33(1):21. https://doi.org/10.1007/s10922-024-09896-8
18. Pizzato F, Bringhenti D, Sisto R, Valenza F (2026) Intent-driven network isolation for the cloud computing continuum. J Netw Syst Manag 34(1):6. https://doi.org/10.1007/s10922-025-09986-1
19. Bringhenti D, Pizzato F, Sisto R, Valenza F (2024) A looping process for cyberattack mitigation. In: Proceedings of the 2024 IEEE international conference on cyber security and resilience (CSR), pp 276–281. https://doi.org/10.1109/CSR61664.2024.10679501
20. Khozam S, Blanc G, Tixeuil S, Totel E (2025) QoSentry: a reinforcement learning framework for QoS-preserving DDoS mitigation in software-defined networks. J Netw Syst Manag 33(4):97. https://doi.org/10.1007/s10922-025-09971-8
21. Khanal B, Kumar C, Ansari MSA (2025) Real-time anomaly detection framework to mitigate emerging threats in software defined networks. J Netw Syst Manag 33(2):26. https://doi.org/10.1007/s10922-025-09904-5
22. Filho AG, Viegas EK, Santin AO, Geremias J (2025) A dynamic network intrusion detection model for infrastructure as code deployed environments. J Netw Syst Manag 33(4):75. https://doi.org/10.1007/s10922-025-09940-1
23. Srinivasan A, Paramadilok J (2026) CUSTOS: toward forensic-ready zero trust at the capture-containment boundary. arXiv:2608.17068. https://doi.org/10.48550/arXiv.2608.17068
24. Wang YM, Beck D, Vo B, Roussev R, Verbowski C (2005) Detecting stealth software with Strider GhostBuster. In: Proceedings of the 2005 international conference on dependable systems and networks (DSN'05), pp 368–377. https://doi.org/10.1109/DSN.2005.39
25. Garfinkel T, Rosenblum M (2003) A virtual machine introspection based architecture for intrusion detection. In: Proceedings of the network and distributed system security symposium (NDSS 2003)

26. Jiang X, Wang X, Xu D (2007) Stealthy malware detection through VMM-based “out-of-the-box” semantic view reconstruction. In: Proceedings of the 14th ACM conference on computer and communications security, pp 128–138. https://doi.org/10.1145/1315245.1315262

27. Sailer R, Zhang X, Jaeger T, van Doorn L (2004) Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX security symposium, pp 223–238

28. Coker G, Guttman J, Loscocco P, Herzog A, Millen J, O’Hanlon B, Ramsdell J, Segall A, Sheehy J, Sniffen B (2011) Principles of remote attestation. Int J Inf Secur 10(2):63–81. https://doi.org/10.1007/s10207-011-0124-7

29. Lamport L, Shostak R, Pease M (1982) The Byzantine generals problem. ACM Trans Program Lang Syst 4(3):382–401. https://doi.org/10.1145/357172.357176

30. Castro M, Liskov B (1999) Practical Byzantine fault tolerance. In: Proceedings of the 3rd symposium on operating systems design and implementation (OSDI ’99), pp 173–186

31. Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secure Comput 1(1):11–33. https://doi.org/10.1109/TDSC.2004.2

32. Littlewood B, Popov P, Strigini L (2001) Modeling software design diversity: a review. ACM Comput Surv 33(2):177–208. https://doi.org/10.1145/384192.384195

33. Garcia M, Bessani A, Gashi I, Neves N, Obelheiro R (2014) Analysis of operating system diversity for intrusion tolerance. Softw Pract Exp 44(6):735–770. https://doi.org/10.1002/spe.2180

34. Zhai E, Chen R, Wolinsky DI, Ford B (2014) Heading off correlated failures through Independence-as-a-Service. In: Proceedings of the 11th USENIX symposium on operating systems design and implementation (OSDI ’14), pp 317–334

35. Vasilomanolakis E, Karuppayah S, Mühlhäuser M, Fischer M (2015) Taxonomy and survey of collaborative intrusion detection. ACM Comput Surv 47(4):55. https://doi.org/10.1145/2716260

36. He J, Yu D (2026) The illusion of independent quorums: epistemic fault domains and correlated cognitive failures in agentic quorums. arXiv:2609.02925. https://doi.org/10.48550/arXiv.2609.02925

37. He J, Yu D (2026) Semantic quorum assurance: collective certification for non-deterministic AI infrastructure. arXiv:2606.08021. https://doi.org/10.48550/arXiv.2606.08021

38. He J, Yu D (2026) The honest quorum problem: epistemic Byzantine fault tolerance for agentic infrastructure. arXiv:2607.16109. https://doi.org/10.48550/arXiv.2607.16109

39. Kazemian P, Varghese G, McKeown N (2012) Header space analysis: static checking for networks. In: Proceedings of the 9th USENIX symposium on networked systems design and implementation (NSDI ’12), pp 113–126

40. Khurshid A, Zou X, Zhou W, Caesar M, Godfrey PB (2013) VeriFlow: verifying network-wide invariants in real time. In: Proceedings of the 10th USENIX symposium on networked systems design and implementation (NSDI ’13), pp 15–27

41. Fogel A, Fung S, Pedrosa L, Walraed-Sullivan M, Govindan R, Mahajan R, Millstein T (2015) A general approach to network configuration analysis. In: Proceedings of the 12th USENIX symposium on networked systems design and implementation (NSDI ’15), pp 469–483

42. Panda A, Lahav O, Argyraki K, Sagiv M, Shenker S (2017) Verifying reachability in networks with mutable datapaths. In: Proceedings of the 14th USENIX symposium on networked systems design and implementation (NSDI ’17), pp 699–718

43. Wang Y, Madi T, Majumdar S, Jarraya Y, Alimohammadifar A, Pourzandi M, Wang L, Debbabi M (2017) TenantGuard: scalable runtime verification of cloud-wide VM-level network isolation. In: Proceedings of the network and distributed system security symposium (NDSS 2017). https://doi.org/10.14722/ndss.2017.23365

44. Backes J, Bayless S, Cook B, Dodge C, Gacek A, Hu AJ, Kahsai T, Kocik B, Kotelnikov E, Kukovec J, McLaughlin S, Reed J, Rungta N, Sizemore J, Stalzer M, Srinivasan P, Subotić P, Varming C, Whaley B (2019) Reachability analysis for AWS-based networks. In: Computer aided verification (CAV 2019). Lecture Notes in Computer Science, vol 11562. Springer, Cham, pp 231–241. https://doi.org/10.1007/978-3-030-25543-5_14

45. Backes J, Bolignano P, Cook B, Dodge C, Gacek A, Luckow K, Rungta N, Tkachuk O, Varming C (2018) Semantic-based automated reasoning for AWS access policies using SMT. In: Proceedings of formal methods in computer aided design (FMCAD 2018), pp 1–9. https://doi.org/10.23919/FMCAD.2018.8602994

46. Steringer E (2022) Principal Mapper (PMapper), version 1.1.5. NCC Group. https://github.com/nccgroup/PMapper. Accessed 16 November 2026

47. Hu Y, Wang W, Khurshid S, Tiwari M (2023) TAC: hybrid IAM privilege escalation detection. arXiv:2304.14540. https://doi.org/10.48550/arXiv.2304.14540

48. Kazdagli M, Tiwari M, Kumar A (2024) CloudLens: modeling and detecting cloud security vulnerabilities. arXiv:2402.10985. https://doi.org/10.48550/arXiv.2402.10985

49. Kuźniar M, Perešíni P, Kostić D (2015) What you need to know about SDN flow tables. In: Passive and active measurement (PAM 2015). Lecture Notes in Computer Science, vol 8995. Springer, Cham, pp 347–359. https://doi.org/10.1007/978-3-319-15509-8_26

50. Perešíni P, Kuźniar M, Kostić D (2015) Monocle: dynamic, fine-grained data plane monitoring. In: Proceedings of the 11th ACM conference on emerging networking experiments and technologies (CoNEXT ’15), article 32. https://doi.org/10.1145/2716281.2836117

51. Shukla A, Fathalli S, Zinner T, Hecker A, Schmid S (2020) P4Consist: toward consistent P4 SDNs. IEEE J Sel Areas Commun 38(7):1293–1307. https://doi.org/10.1109/JSAC.2020.2999653

52. Sasaki T, Pappas C, Lee T, Hoefler T, Perrig A (2016) SDNsec: forwarding accountability for the SDN data plane. In: Proceedings of the 25th international conference on computer communication and networks (ICCCN 2016), pp 1–10. https://doi.org/10.1109/ICCCN.2016.7568569

53. Bu K, Yang Y, Guo Z, Yang Y, Li X, Zhang S (2021) Securing middlebox policy enforcement in SDN. Comput Netw 193:108099. https://doi.org/10.1016/j.comnet.2021.108099

54. Clemm A, Ciavaglia L, Granville LZ, Tantsura J (2022) Intent-based networking – concepts and definitions. RFC 9315, Internet Research Task Force. https://doi.org/10.17487/RFC9315

55. Ojha A (2026) Differential observability: detecting silent state divergence in eBPF-based cloud networks. Google Research publication. https://research.google/pubs/differential-observability-detecting-silent-state-divergence-in-ebpf-based-cloud-networks-2/. Accessed 12 November 2026

56. Ping An Technology (Shenzhen) Co Ltd (2023) Network isolation validity verification method, device, equipment and storage medium. Chinese patent CN114978954B. https://patents.google.com/patent/CN114978954B/en. Accessed 13 November 2026

57. Amazon Web Services (n.d.) AWSSupport-ContainEC2Instance. AWS Systems Manager Automation Runbook Reference. https://docs.aws.amazon.com/systems-manager-automation-runbooks/latest/userguide/automation-awssupport-containec2instance.html. Accessed 13 November 2026

58. Amazon Web Services (n.d.) ReplaceNetworkAclAssociation. Amazon EC2 API Reference. https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_ReplaceNetworkAclAssociation.html. Accessed 15 November 2026
59. Gietzen S (2018) AWS IAM privilege escalation – methods and mitigation. Rhino Security Labs. https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/. Accessed 15 November 2026
60. Amazon Web Services (n.d.) Network ACL rules. Amazon VPC User Guide. https://docs.aws.amazon.com/vpc/latest/userguide/nacl-rules.html. Accessed 13 November 2026
61. Amazon Web Services (n.d.) AssignPrivateIpAddresses. Amazon EC2 API Reference. https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_AssignPrivateIpAddresses.html. Accessed 13 November 2026
62. Amazon Web Services (n.d.) AttachNetworkInterface. Amazon EC2 API Reference. https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_AttachNetworkInterface.html. Accessed 13 November 2026
63. Amazon Web Services (n.d.) Troubleshoot unhealthy instances in Amazon EC2 Auto Scaling. Amazon EC2 Auto Scaling User Guide. https://docs.aws.amazon.com/autoscaling/ec2/userguide/ts-as-healthchecks.html. Accessed 14 November 2026
64. Obando J (2021) Automate Amazon EC2 instance isolation by using tags. AWS Security Blog. https://aws.amazon.com/blogs/security/automate-amazon-ec2-instance-isolation-by-using-tags. Accessed 14 November 2026
65. Agavriloae E (2025) IAM persistence through eventual consistency. Hacking The Cloud. https://hackingthe.cloud/aws/post_exploitation/iam_persistence_eventual_consistency/. Accessed 16 November 2026
66. Newcombe RG (1998) Improved confidence intervals for the difference between binomial proportions based on paired data. Stat Med 17(22):2635–2650. https://doi.org/10.1002/(SICI)1097-0258(19981130)17:22%3C2635::AID-SIM954%3E3.0.CO;2-C
67. Amazon Web Services (n.d.) Flow log limitations. Amazon VPC User Guide. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs-limitations.html. Accessed 11 November 2026
68. Bentzen M, Ismail M (2021) Secure your Amazon VPC DNS resolution with Amazon Route 53 Resolver DNS Firewall. AWS Networking and Content Delivery Blog. https://aws.amazon.com/blogs/networking-and-content-delivery/secure-your-amazon-vpc-dns-resolution-with-amazon-route-53-resolver-dns-firewall. Accessed 16 November 2026
69. Amazon Web Services (n.d.) Amazon VPC quotas. Amazon VPC User Guide. https://docs.aws.amazon.com/vpc/latest/userguide/amazon-vpc-limits.html. Accessed 16 November 2026