

The L0–L4 SOC Automation Maturity Model

A pragmatic framework for graduating autonomous response, with audit trails

AiSOC project · v1.0 · Released 2026-05-13

MIT licensed · github.com/beenuar/AiSOC

Executive summary

Modern SOC face two structural problems at once. The first is **volume**: mature detections, broader telemetry, and more sources of truth produce more alerts than analysts can read, let alone investigate. The second is **trust**: the same agentic tooling that could absorb the volume is, in practice, gated by every operator's reasonable refusal to grant blanket autonomy to a system that could disable a CEO's account, isolate a production database host, or block egress for an entire VPC at 2 a.m. on a Sunday.

Most autonomy-vs-volume discussions collapse the question into a single dial: *is the SOC autonomous, yes or no?* That framing is wrong. It encourages overclaiming on one side ("*fully autonomous response*" usually means "*fully autonomous notifications*") and underdelivery on the other ("*human-in-the-loop on every action*" usually means "*the same analyst queue you had five years ago, now with chatbots*").

The L0–L4 SOC Automation Maturity model breaks the dial into five positions, each defined by exactly two things:

1. The **blast radius** of an action — MINIMAL, LOW, MEDIUM, HIGH, or CRITICAL.
2. The **set of blast radii** that are permitted to execute autonomously at this tier; everything outside that set is routed to the human approval queue, and everything inside the set is logged on execution.

The five tiers are:

- **L0 — Observe.** The agent reasons, recommends, and drafts. It does not execute. Every action is queued for analyst approval.
- **L1 — Notify.** The agent is permitted to execute MINIMAL actions — notifications, tickets, ChatOps user verification, read-only SIEM searches — autonomously.
- **L2 — Contain.** The agent is permitted to execute MINIMAL and LOW actions — file quarantine, IOC blocklist, AV scan, forensics capture. This is where most production deployments land today.
- **L3 — Remediate.** The agent is permitted to execute MEDIUM actions — block IP/domain, kill process, reset password, force MFA, run a playbook — in addition to L2's scope. Human review is retrospective.
- **L4 — Automate.** The agent is permitted to execute HIGH-radius actions — host isolation, account disablement, session suspension, remote script execution — but **only** when the (action, target) pair matches a per-tenant whitelist entry with explicit constraints.

This paper documents the model precisely enough that an operator can choose a tier, an auditor can verify the choice, and an engineer can contribute new action types without breaking the gate. It is also, deliberately, an opinionated document: it argues that any vendor's claim of "autonomous SOC" is incomplete without a published, audited, per-action gate, and it shows what such a gate looks like in code.

Why a maturity model

The case against a single autonomy switch

The simplest model of SOC automation is binary: either the agent fires actions, or it doesn't. The problem with binary is that it forces operators into one of three positions, none of which are defensible:

1. **All-manual.** Everything goes through the human queue. The alert-to-incident ratio is irrelevant because the alert-to-action ratio is bounded by analyst working hours. MTTR ceilings are measured in minutes for the lucky shift and hours for the unlucky one. This is the status quo for the majority of SOC's as of 2026.
2. **All-auto with vendor-defined safety rails.** Everything fires automatically, gated by whatever the platform considers safe. The operator doesn't choose the rails; the rails are a marketing commitment. False positives are the platform's fault. This is the "autonomous SOC" claim made by most autonomy-forward vendors. It is accurate for narrow demos and inaccurate for production tenants.
3. **All-auto with feature flags.** Everything fires automatically, but operators can disable specific actions. This is a per-flag approval system, not a maturity model. It scales linearly in operator attention: 30 action types means 30 flags to reason about.

None of these models give an operator a single coherent answer to "what is my SOC allowed to do autonomously today, and what changes when I flip a single switch?"

The case for action blast radius as the unit of trust

The insight underlying L0-L4 is that **operator trust is granted per action class, not per product**. Operators have stable, defensible intuitions about action blast radius:

- Sending a Slack message about an alert is not the same as quarantining a file.
- Quarantining a file is not the same as resetting a password.
- Resetting a password is not the same as isolating a host.
- Isolating a host is not the same as disabling a user account on the identity provider.

Each step up that ladder represents an order-of-magnitude increase in the cost of a false positive. A wrong Slack message costs attention. A wrong file quarantine costs an analyst's morning. A wrong host isolation costs a developer's day. A wrong account disablement costs a meeting and sometimes a contract.

Operators don't trust products. They trust *specific products to take specific action classes against specific targets*. A maturity model encodes that intuition as a small set of tiers; the per-action whitelist encodes the (target, constraint) fine-grain at the top tier.

The case for a written, audited gate

The third design choice is that the gate must be **inspectable in code and audit-logging per fire**. Operators must be able to read the gate function, look at the gate log for the last 24 hours, and

form a verifiable belief about what the agent did and didn't do. Without this, the maturity model is just marketing.

In AiSOC, the gate is a single function — `evaluate_gate(request, config)` in `services/actions/app/services/maturity.py` — that returns one of `auto`, `queued_approval`, or `blocked`, along with a structured rationale. Every decision is written to `remediation_gate_log` with the tier, blast radius, action type, target, and human-readable explanation. This log is queryable via `/api/v1/remediation/gate-log` and is the source of truth for tier-graduation conversations.

The five tiers in detail

L0 — Observe

L0 is the safe-by-default position. The agent stack runs at full capability — it triages alerts, enriches observables, correlates across sources, drafts case notes, suggests playbooks, and proposes actions — but it never executes. Every action a playbook, recommendation, or human operator requests is routed to the approval queue.

Entry criteria. A working AiSOC tenant with at least one connector emitting alerts. No special trust signals required. L0 is the default for new tenants and for tenants in regulated environments where the compliance team needs an audit window before any autonomous action is permitted.

What is auto-executed. Nothing.

What is gated. Every action, regardless of blast radius. The approval queue is the operating mode.

MTTR target. Bounded by human attention. The agent cannot reduce median MTTR below the time it takes an analyst to read an alert, evaluate the suggested actions, and click approve. For high-volume tenants, MTTR at L0 is functionally identical to MTTR before AiSOC was deployed; the agent buys investigation depth, not response speed.

FP tolerance. Very high. Because no autonomous action fires, false positives carry only the cost of analyst attention.

Honest characterisation. L0 is a starting position, not a destination. SOCs that stay at L0 forever are getting only a fraction of the value an autonomous agent provides. But operators who skip L0 entirely — who turn on autonomous response before they've reviewed a single agent recommendation — are setting up for a high-cost false-positive event that destroys trust faster than any feature can rebuild it.

L1 — Notify

L1 is the first tier where the agent fires actions autonomously. The permitted actions all share a single property: their only side effect is information movement, not infrastructure mutation.

Entry criteria. The tenant has wired at least one notification connector — Slack, Teams, PagerDuty, Opsgenie, or an ITSM connector for ticket creation — and has opted into L1 explicitly via the `/api/v1/remediation/config` endpoint.

What is auto-executed. `notify_slack`, `create_ticket`, `chatops_verify`, `search_siem`. These are the MINIMAL-blast-radius actions defined in `ACTION_BLAST_RADIUS`.

What is gated. Every LOW, MEDIUM, and HIGH action is queued.

MTTR target. Time-to-notify drops from minutes to seconds; time-to- contain is still bounded by human review. For the most common case — analyst gets paged, reads the agent's draft, approves the suggested containment — L1 trims the page-to-read step from "minutes of dashboard hunting" to "seconds of reading a structured Slack message."

FP tolerance. Medium-high. A wrong Slack post is recoverable and embarrassing. A wrong ticket gets closed in two clicks. ChatOps verification of a user is itself a feedback signal — if the user says "that wasn't me," the case escalates and the false positive becomes a true positive.

Honest characterisation. L1 is the floor for any production SOC deployment. Tenants that don't reach L1 within their first deployment cycle should treat that as a deployment problem, not a maturity problem.

L2 — Contain

L2 is where AiSOC's median production tenant operates. The agent is permitted to execute reversible, single-resource, low-impact containment actions in addition to L1's notification scope.

Entry criteria. The tenant has been at L1 long enough to accumulate a sample of agent-suggested decisions (usually one deployment cycle, or "a couple of weeks of real alert volume"). The operator has reviewed the gate log, sampled the agent's reasoning, and is satisfied with the false-positive rate. The rollback semantics in `services/actions/app/executors/` for each LOW action have been read and accepted.

What is auto-executed. Everything at L1 plus: `quarantine_file`, `capture_forensics`, `add_ioc_to_blocklist`, `run_av_scan`, `create_notable_event`.

What is gated. Everything MEDIUM and above is queued.

MTTR target. MTTR for the auto-execution path drops to single-digit minutes end-to-end. The bulk of AiSOC's published benchmark numbers come from L2-tier tenants, because that is where the platform's autonomous behaviour first becomes load-bearing.

FP tolerance. Medium. A wrongly-quarantined file is recoverable from the gate log: the rollback metadata in `ActionRequest.rollback_data` is designed so an analyst can release the quarantine in a single API call. A wrongly-blocked IOC has a similar rollback path. The cost is analyst time, not user disruption.

Honest characterisation. When a prospect asks "where is AiSOC autonomous today," the truthful answer is "L2 for the median tenant, L3 on selected action classes for mature tenants." Anyone marketing a higher number is either running a narrow demo, counting recommendations as actions, or describing a roadmap.

L3 — Remediate

L3 expands the agent's autonomous scope to **MEDIUM**-blast-radius actions — actions that affect a single resource but whose impact is more visible than file quarantine.

Entry criteria. The tenant has been at L2 for an extended period with a satisfactory false-positive rate. For each **MEDIUM** action class the tenant intends to enable, the rollback story is tested end-to-end: "a wrongly-fired `reset_password` can be undone in N minutes via the following API call." ITSM integration is live so the agent's retrospective audit trail lands on the analyst's queue automatically. Compliance has reviewed and accepted that the agent's actions are auditable in `remediation_gate_log` and in connector-native logs (e.g. firewall rule changes are auditable on the firewall, password resets are auditable in the IdP audit log).

What is auto-executed. Everything at L2 plus: `block_ip`, `block_domain`, `kill_process`, `reset_password`, `force_mfa`, `run_playbook`, `allow_ip`, `block_ioc`, `sync_detection_rule`, `update_watcher`.

What is gated. Everything **HIGH** is queued.

MTTR target. MTTR for **MEDIUM** actions drops below five minutes on well-instrumented connectors. The analyst's role shifts from prospective ("should the agent act?") to retrospective ("was the agent right?").

FP tolerance. Low-to-medium. A wrong password reset interrupts a real user. A wrong DNS block interrupts a real service. Rollback is supported but not free; the cost is measured in user-visible disruption plus analyst recovery time.

Honest characterisation. L3 is not a binary upgrade from L2. In practice, AiSOC tenants reach L3 *per action type* rather than as a global tier bump. The recommended path is to set the tenant tier to L2 and use `force_auto` overrides on specific **MEDIUM** action types (e.g. `block_ip` for known-malicious IPs from high-confidence threat intel) while leaving the rest of the **MEDIUM** tier in the approval queue. This keeps the tier setting honest and surface-visible.

L4 — Automate

L4 is the highest tier, and it is the most operationally demanding. The agent is permitted to execute **HIGH**-blast-radius actions — host isolation, user disablement, session suspension, remote script execution — **only** when the (action, target) combination matches a whitelist entry in `remediation_whitelist`.

The whitelist is the operator's way of saying:

"For these specific action types, applied to these specific target patterns, I have pre-approved autonomous execution. For everything else at HIGH radius, queue it like you would at L3."

The whitelist supports constraints on the action's target — for example, `target_prefix`: `"10.0.0."` constrains an `isolate_host` whitelist entry to internal IP ranges only. Expiry is supported via the `expires_at` column so an L4 whitelist entry can be time-bounded (a 24-hour "isolate hosts in the breached subnet" automation during an incident).

Entry criteria. The tenant has been at L3 for an extended period with a clean gate log. The operator has at least one closed-loop scenario where the (action, target) combination is provably safe — the canonical example being `isolate_host` on workstations tagged `quarantine-eligible` in the asset inventory. The whitelist entry has explicit constraints, an expiry date, and a ChatOps notification path so every fire reaches a human within minutes.

What is auto-executed. Everything at L3 plus, *for whitelisted (action, target) combinations only*: `isolate_host`, `disable_user`, `suspend_session`, `run_script`. Any HIGH-radius action that does not match a whitelist entry falls back to `queued_approval` exactly as it would at L3.

What is gated. Every HIGH-radius action without a matching whitelist entry. Every CRITICAL-radius action regardless of tier (no whitelist override is possible for CRITICAL).

MTTR target. For the whitelisted closed loops, MTTR drops below two minutes end-to-end. The agent owns the response; the human reviews at audit time.

FP tolerance. Very low for whitelisted actions. The whitelist is the operator's signed contract that this specific combination has been de-risked, with rollback, notification, and expiry built in.

Honest characterisation. As of v8.0, a small number of pilot AiSOC tenants are operating closed-loop L4 automations for narrow scenarios (host isolation on quarantine-tagged workstations, session suspension on tokens from impossible-travel events). The platform supports L4 fully, but the operator burden of building, testing, and maintaining the whitelist is high enough that "L4 by default" is not a goal. The goal is "L4 for the actions that deserve L4, L3 for the rest."

How the gate evaluates an action

For every action a playbook, agent, or human submits, `evaluate_gate(request, config)` runs the following decision tree. This is the actual code path, reproduced here so the model is unambiguous.

1. Look up the action's blast radius from ACTION_BLAST_RADIUS.
2. Look up the tenant's tier and per-action overrides from the tenant's RemediationMaturity row.
3. If action_overrides[action_type] has block=true:
→ decision = "blocked"; record rationale; stop.
4. If action_overrides[action_type] has force_auto=true:
→ decision = "auto" with overrides_applied=["force_auto_override"]; record rationale; stop.
5. If blast_radius == HIGH and tier == L4:
 - a. Scan remediation_whitelist for an entry matching this action_type and target_prefix.
 - b. If no match: decision = "queued_approval"; stop.
 - c. If match: continue, with overrides_applied=["whitelist:<id>"].
6. Compare blast_radius against the tier's allow-set:
 - L0 = {}
 - L1 = {MINIMAL}
 - L2 = {MINIMAL, LOW}
 - L3 = {MINIMAL, LOW, MEDIUM}
 - L4 = {MINIMAL, LOW, MEDIUM, HIGH}
 - (CRITICAL is never in any allow-set.)
7. If blast_radius is in the allow-set: decision = "auto".
Else: decision = "queued_approval".
8. Record the decision to remediation_gate_log with tier, action_type, blast_radius, decision, rationale, actor.

Every decision — `auto`, `queued_approval`, or `blocked` — is logged. The `actor` field is `system` for agent-initiated actions and the user UUID for human-initiated ones; the gate runs uniformly regardless.

How to assess your SOC's current tier

These are the questions AiSOC's onboarding flow asks. They are the same questions a security architect should ask before flipping the tier dial.

1. **Notification readiness.** Is at least one notification connector configured and actively read? If the agent's Slack post falls in an abandoned channel, L1 buys nothing. Verdict: stay at L0 until notifications are real.
2. **Alert-to-incident ratio.** AiSOC's public benchmark gates at 50:1. Tenants with a higher ratio will amplify noise, not reduce it, when the agent starts firing autonomously. Verdict: stay at L1.
3. **Rollback discipline.** For each LOW-blast-radius action class you intend to auto-fire, is the rollback documented and tested? File quarantine release is trivial; IOC blocklist removal is one API call. If you've never tested a release, you'll learn the hard way. Verdict: stay at L1 until rollback is real.
4. **Gate log review cadence.** Is there a person or scheduled job that reviews the last 24 hours of `/api/v1/remediation/gate-log` every morning? If no, L3 is operationally dangerous. Verdict: stay at L2.

5. **Whitelist hygiene.** For L4, do you have at least one specific (action, target) combination with constraints, expiry, and a notification path? Have you mapped the rollback story end-to-end?
Verdict: stay at L3.

The honest answer for most SOC's as of 2026 is that they are at L0 with an ambitious roadmap to L2. The honest answer for most AiSOC tenants as of v8.0 is that they are at L2 with selective L3 overrides on specific high-confidence action types.

Migration paths

Tiers should move gradually and reversibly.

L0 → L1

- Wire at least one notification connector.
- Read the gate log after a full alert cycle.
- PUT /api/v1/remediation/config { "maturity_tier": 1 }.

L1 → L2

- Run at L1 for one deployment cycle (typically 2–4 weeks).
- Sample the gate log; confirm the agent's recommended LOW actions would have been correct.
- Read and accept rollback semantics for each LOW action class.
- PUT /api/v1/remediation/config { "maturity_tier": 2 }.

L2 → L3 (recommended path: per-action, not global)

- Identify a specific MEDIUM action class with high-confidence input (e.g. block_ip when threat intel confidence > 0.95).
- Add { "force_auto": true } to action_overrides for that action class.
- Keep the tier at L2.
- After several weeks with a clean gate log on that action class, add another. Promote the tier only when most MEDIUM action classes are already auto-approved via overrides.

L3 → L4

- Identify one closed-loop scenario (action_type, target_prefix).
- Add a whitelist entry with explicit constraints, expiry, and ChatOps notification on every fire.
- Run the loop for an incident or a quarterly fire drill.
- Promote the tier to L4 only when the whitelist is exercised and reviewed in production.

The dial moves both ways. A tier drop is a single API call away. AiSOC treats tier *downgrades* — for example, dropping back to L2 during a sensitive period such as a financial audit or after a recent operational incident — as a normal, lossless operation. The gate picks up the new tier on the next action with no service restart.

Open questions and where the industry is heading

The maturity model is not a finished artefact. Several open questions will shape it over the next 12–24 months.

1. Per-target trust at L3. L4 has whitelist-level fine-grain via target prefix constraints; L3 does not. There is a credible argument for extending whitelist-style constraints down to L3 so a tenant can say "auto-fire `block_ip` for sources in known-malicious feeds but queue auto-fires against private RFC1918 ranges." This would replace the current `force_auto` boolean with a constraint object; the cost is a modest schema change on `action_overrides`.

2. Tier-aware playbooks. AiSOC's playbook engine currently does not parameterise its steps by the executing tenant's tier. A playbook author who wants step 4 to "fire if tier $\geq$ L3, else queue" must encode that conditional in the playbook itself. A future revision may let playbooks declare a per-step minimum tier and let the engine route accordingly, making playbooks portable across tenants at different tiers.

3. Tier as a marketplace property. Plugins in the AiSOC marketplace declare which action types they offer. A future revision could let plugins declare a recommended *minimum tier* for their actions, so operators see at install time whether the new connector's actions fit the tenant's current automation posture.

4. Tier as a compliance artefact. Some regulated environments (financial services, healthcare, government) need a sworn statement of "what could the agent have done autonomously during this incident window." `remediation_gate_log` is a partial answer, but a more auditor-friendly export — tier history, whitelist diff, override diff, gate log — would close the loop. The compliance evidence export at `/api/v1/compliance` is the likely home for this.

5. Cross-vendor reversibility taxonomy. Action blast radius is currently a project-internal taxonomy. The MITRE D3FEND defensive technique catalog and the OASIS CACAO action vocabularies do not encode blast radius. There is a credible case for proposing a shared taxonomy to those communities so that "this CACAO action is HIGH blast radius" is a portable claim across implementations.

The bigger industry shift the model anticipates is the move from *autonomous SOC* as a marketing tier to *per-action autonomy* as the default vocabulary. The vendor that says "we are L4 on these six action types and L2 on the rest, here is the gate log" is more credible than the vendor that says "we are autonomous." Operators are catching up; auditors are ahead of operators; regulators are starting to ask.

References and further reading

The model is informed by, but not aligned to, the following industry frameworks:

- **MITRE D3FEND.** The defensive-technique catalog organises responses by mechanism (isolate, evict, restrict, restore). The D3FEND mechanism hierarchy maps roughly onto AiSOC's blast-radius taxonomy: `restrict` techniques tend to be `LOW/MEDIUM`, `isolate` and

evict techniques tend to be HIGH. The mapping is not 1:1 — D3FEND is about technique identity, AiSOC is about operational impact — but the framings rhyme. See: <https://d3fend.mitre.org/>.

- **OASIS CACAO.** The Collaborative Automated Course of Action Operations specification (v2.0) is the open-standard format for describing playbooks across security platforms. AiSOC's playbook engine is format-compatible with CACAO v2.0, which means a CACAO playbook can be loaded by the engine and gated by the L0–L4 model exactly as a native AiSOC playbook would be. See: <https://www.oasis-open.org/standard/cacao/>.
- **SAE J3016 (driving automation levels).** The automotive industry's five-tier model for driving automation (L0 monitoring only through L5 fully autonomous) is the original "level model" for human-AI shared responsibility. The analogy to SOC operations is loose — security actions are largely reversible while driving actions are not — but the framing of "who is responsible at each tier" is portable, and the L0–L4 naming is a deliberate echo. See: https://www.sae.org/standards/content/j3016_202104/.
- **NIST SP 800-61 (Computer Security Incident Handling Guide).** The classical four-phase incident response lifecycle (preparation, detection and analysis, containment-eradication-recovery, post-incident activity) maps onto the AiSOC tier model: L0/L1 belong to detection and analysis, L2/L3 belong to containment and eradication, L4 closed loops belong to fully-instrumented recovery. See: <https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final>.

Appendix A — Tier-by-tier comparison table

Property	L0	L1	L2	L3	L4
Auto-execute MINIMAL	no	yes	yes	yes	yes
Auto-execute LOW	no	no	yes	yes	yes
Auto-execute MEDIUM	no	no	no	yes	yes
Auto-execute HIGH	no	no	no	no	whitelisted only
Auto-execute CRITICAL	no	no	no	no	no
Required connectors	none	one notification	one notification + one containment	+ one MEDIUM-class executor	+ tested rollback at HIGH
Required process	analyst reads	analyst reads	analyst reviews gate log	analyst reviews gate log daily	whitelist owner reviews per fire
Typical MTTR ceiling	hours	minutes (notify)	minutes (contain)	minutes (remediate)	seconds–minutes (whitelist)
Typical FP cost	analyst attention	analyst attention	analyst time	user disruption	severe (mitigated by whitelist)

Appendix B — Worked example: closed-loop session suspension

A pilot tenant runs L3 for the tenant default and an L4 whitelist for a single closed loop:

"Suspend the session token if AiSOC detects impossible travel."

The whitelist entry is:

```
{
  "action_type": "suspend_session",
  "blast_radius": "high",
  "constraints": {
    "target_prefix": "session:"
  },
  "expires_at": "2026-12-31T23:59:59Z"
}
```

When the impossible-travel detection fires and the agent submits a `suspend_session` action against target `session:eyJhbGc...`, the gate runs:

1. Blast radius is `HIGH`.
2. Tier is `L4`.
3. Whitelist scan finds the entry; `target_prefix` matches.
4. Decision: `auto`, with `overrides_applied=["whitelist:<entry-id>"]`.
5. Action executes against the IdP. Rollback metadata is captured.
6. A ChatOps message goes to the on-call channel within seconds.
7. The gate log row is queryable for retrospective review.

If the same agent had submitted `disable_user` (`HIGH` radius, no whitelist entry), the gate would have returned `queued_approval` and the action would have landed in the analyst queue.

This is the model in production. The dial moves slowly. The audit log is always on. The operator owns the trust posture; AiSOC owns the gate.

The AiSOC project is MIT-licensed and community-maintained. The code underlying this paper is at <https://github.com/beenuar/AiSOC>; contributions to the maturity model — new action types, tier-aware playbook syntax, marketplace metadata — are welcome.