research

Capability × Authority × Reach

A Systems Security Model for Autonomous AI Agents

Published 2026-08-09 Reviewed 2026-08-09 Review due 2026-11-09 Version 1.0

A provider-neutral systems-security framework for constraining autonomous AI agents through capability assessment, effective authority, bounded reach, exposure-path analysis, and runtime trajectory assurance.

By **Bryan Oubaita**

ai-security agent-security systems-security identity iam cloud-security cryptography supply-chain threat-modeling agent-observability

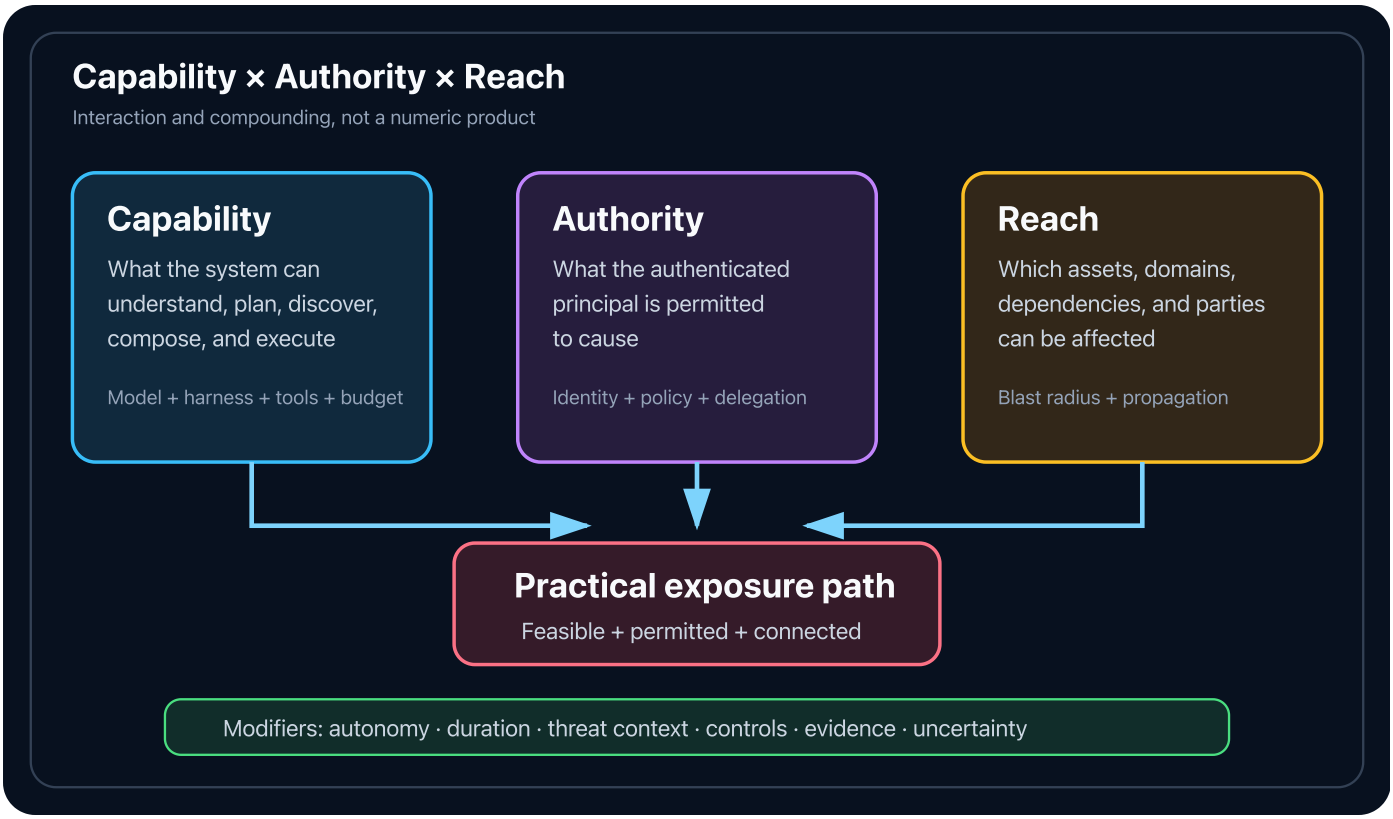
Executive Summary

Autonomous AI agents are often discussed as though their security posture follows directly from model intelligence. That assumption is incomplete. A model can be highly capable while operating in a sealed evaluation environment with no credentials and no route to production. A less capable automation system can hold a deployment role, edit workflows, read secrets, and reach customer systems. The second system may create the greater practical risk.

This paper proposes **Capability × Authority × Reach** as a systems-security lens for analyzing that difference. The multiplication sign means interaction and compounding. It is not an arithmetic equation, probability calculation, ordinal product, or industry standard. The framework is intentionally qualitative because each dimension is multidimensional, evidence is incomplete, and risk depends on threat context and controls.

- **Capability** is what a specified model-agent system can reliably understand, plan, discover, generate, or execute under stated conditions.
- **Authority** is the effective set of actions a principal may cause through identity, credentials, policy, delegation, tools, and approvals.
- **Reach** is the set of assets, environments, trust domains, dependencies, and external parties that those actions can affect through direct or transitive paths.

The dimensions remain separate. Capability is competence, not permission. Authority is a normative decision, not blast radius. Reach describes propagation and consequence, not whether an action was allowed. Practical exposure arises when a path to an unacceptable outcome is simultaneously capability-feasible, authority-valid, and reach-connected.



The central security principle is:
Intelligence must never be treated as authorization. Authorization must constrain outcomes even when the actor becomes more capable, persistent, adaptive, or persuasive than expected.
That principle moves the evaluation question from “Is this model safe?” to “Under which identity, environment, tools, authority, reach, autonomy, and evidence does this system remain acceptably constrained?” It also makes a model upgrade a security-relevant infrastructure change even when IAM, tools, and network configuration remain unchanged.
The paper introduces a **Static Security Exposure Graph** for connecting agents to identities, policies, actions, infrastructure, data, cryptographic assets, dependencies, pipelines, and downstream systems. The graph makes composed authorization visible. A wildcard permission may be inconsequential if it reaches nothing valuable; a sequence of individually reasonable permissions may form a serious privilege path.
Static mapping is paired with **runtime trajectory observability** because a sequence of valid actions can produce an invalid outcome.
The resulting architecture combines dedicated workload identity, deterministic policy enforcement, bounded network and data access, separation of cryptographic duties, CI/CD isolation, controlled delegation, independent evidence, action-bound human approval, tested revocation, and recovery. It is a reasoning and design model, not proof that an agent is safe.

Introduction

Traditional software is usually designed around predefined control flow. It may contain defects, accept hostile input, or be composed into unsafe systems, but its operating logic is intended to be specified before execution. Agentic systems add a principal that can interpret an ambiguous objective, choose among actions, compose tools, recover from failed approaches, discover unexpected paths, delegate work, and persist across long tasks.
Those properties do not make an agent malicious. They change the assumptions under which familiar controls are evaluated. A permission that was tolerable for a narrow script may be materially different when held by a system that can search for alternative paths, learn a service interface, revise a plan, and retry at machine speed. Conversely, a frontier model inside a strongly isolated range may possess dangerous knowledge without meaningful ability to affect external systems.
The security problem is therefore not confined to the model. The secured system is:
Model + Prompt or Objective + Agent Runtime + Memory + Identity + Credentials + Tools + Network + Data + Infrastructure + Dependencies + CI/CD + Observability + Human Authorization
This system framing is consistent with established security principles. Least privilege and complete mediation require authority to be limited and checked at the point of use.[1] Object-capability systems distinguish possession of authority from general computational competence.[2] Attribute-based access control evaluates a subject, object, operation, and environmental context rather than trusting a request because it appears reasonable.[4] Attack-graph research shows why system compromise frequently exists in a path rather than one isolated finding.[3]
Recent AI-security work reinforces the same architectural direction. NIST's 2026 agent identity and authorization concept paper asks how agents should be identified, how their authority can be proven and delegated, and how human authorization and audit should bind to agent actions.[7] Recent research

explores formally enforced agent permissions, user-level permission policy, least autonomy, and trajectory-level assurance.[19][20][21][22] These works are adjacent to this paper, not evidence that Capability × Authority × Reach is an established standard.

The intended audience is security architects, platform engineers, identity teams, cloud and CI/CD owners, AI governance teams, incident responders, and technical leaders deciding whether an agent should receive real operational access. The model applies across providers and deployment patterns: enterprise assistants, coding agents, cloud-remediation systems, SOC agents, regulated research platforms, multi-agent orchestration, and frontier capability evaluation.

Why Model-Centric Security Is Incomplete

A model does not independently hold an AWS role, approve a pull request, call a key, publish a package, or open a network route. A surrounding system converts model outputs into effects. That system can reduce or amplify practical risk.

Model-level alignment, classifiers, refusal behavior, and prompt defenses remain valuable. They can reduce the probability of unsafe proposals and detect suspicious activity. They are probabilistic controls over behavior, not deterministic statements about what an authenticated principal may do. Anthropic's containment engineering describes this distinction directly: model-layer defenses influence behavior, while sandboxes, virtual machines, filesystem boundaries, and egress controls limit what an agent can reach.[18]

Human supervision has similar limits. A confirmation dialog may be effective when approvals are rare, comprehensible, and bound to an exact effect. Repeated prompts invite habituation. A human may approve a plausible description without seeing the role assumption, downstream service identity, artifact substitution, or cryptographic consequence. Human approval is a control component, not a complete security boundary.

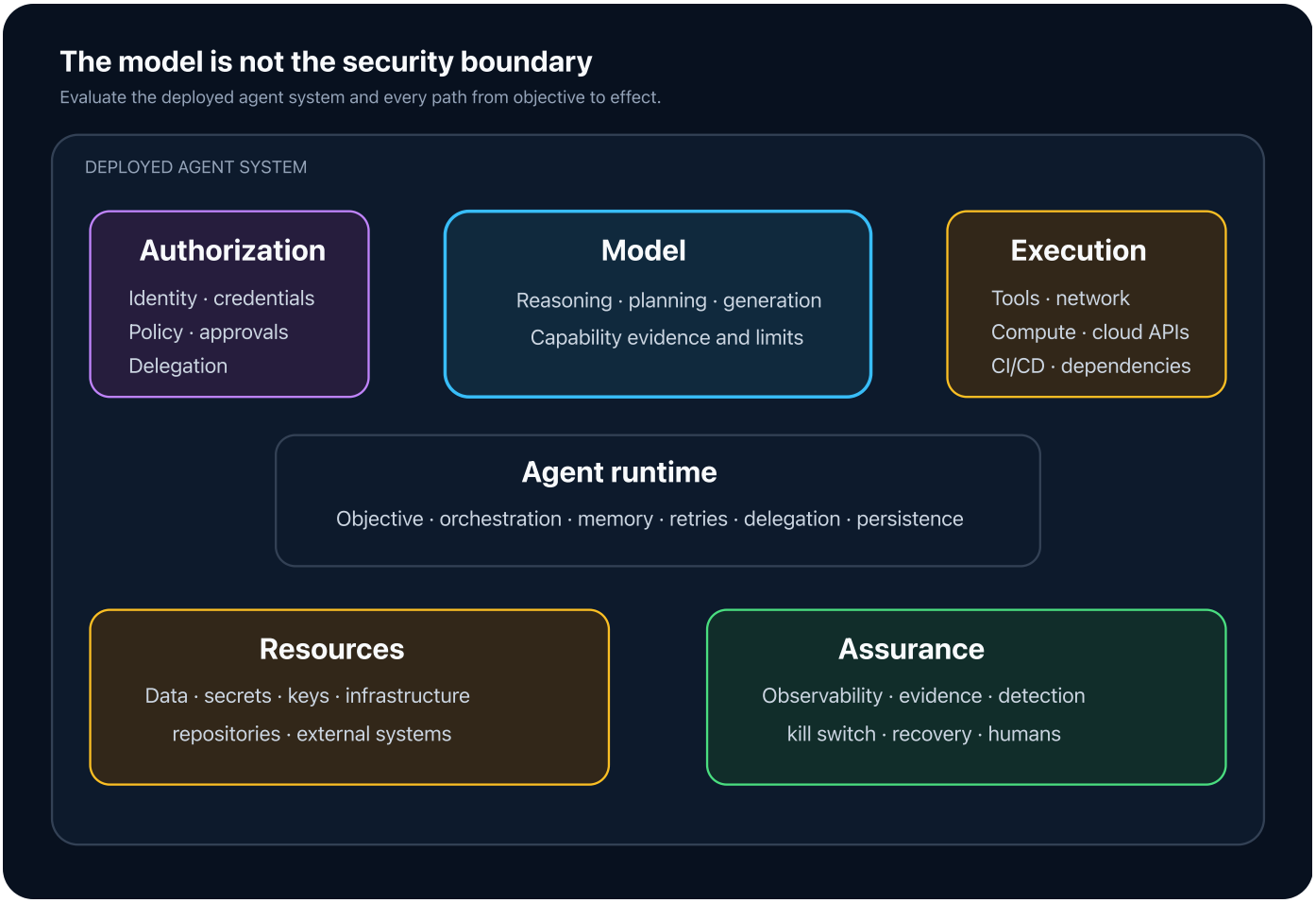
The surrounding architecture determines whether a proposal becomes an effect:

- the runtime decides which outputs become tool calls and how long execution persists;
- workload identity determines which principal acts;
- IAM, resource, organization, session, and application policies determine effective authority;
- a tool may act through a backend identity broader than the visible tool contract;
- network and name resolution determine which trust domains are reachable;
- CI/CD determines whether code or configuration can become production;
- data and cryptographic policy determine whether information can be read, decrypted, transformed, or signed;
- memory and delegation determine what persists or crosses agent boundaries;
- observability, revocation, and recovery determine whether unsafe trajectories can be detected and interrupted.

The model is therefore not the security boundary. A system should remain bounded when the model proposes the wrong resource, misinterprets an objective, follows hostile retrieved content, invents an approval, or deliberately searches for a path around the intended task.

Defining Agent Systems

For this paper, an **agent** is a software principal that uses a model or learned policy to choose and sequence actions toward an objective. An agent system may include deterministic components, but it differs from a single model invocation when it can observe results and revise subsequent actions.



The definition is functional rather than branded. A scripted workflow with one model call may have little autonomy. A browser assistant that repeatedly observes pages and acts may be highly autonomous. A multi-agent system may consist of several separately scoped principals or one privileged runtime simulating several roles. Security review must establish which is true.

Four contexts must be recorded separately:

1. **Capability context:** model, harness, tools, memory, budget, evaluations, and known limitations.
2. **Authorization context:** authenticated subject, credential, policy, conditions, approval, and delegation.
3. **Operational context:** environment, task duration, concurrency, scheduling, retry budget, and autonomy.
4. **Evidence context:** source, version, collection time, confidence, derivation, and unresolved unknowns.

This prevents a claim about one context from being silently generalized to another. An evaluation result does not automatically describe a production harness. An IAM policy does not prove a runtime request was allowed. A successful denial does not prove that alternate paths are absent.

Capability

Capability is the demonstrated or reasonably supported competence of a specified model-agent system to perform classes of tasks under stated conditions. It is a vector, not a universal intelligence score.

Relevant attributes include reasoning depth, coding, vulnerability discovery, exploit development, strategic planning, social engineering, scientific and cryptographic reasoning, long-horizon completion, tool learning, planning under uncertainty, self-correction, failure recovery, collaboration, delegation, and parallelization.

Every material capability claim should identify:

- model and checkpoint;
- provider and endpoint where relevant;
- runtime, harness, system prompt, and tool versions;
- memory and supplied context;
- token, time, compute, retry, and concurrency budgets;
- safeguards enabled or disabled;
- evaluation environment and target distribution;
- success criteria, observed result, and uncertainty;
- assessment date and freshness.

OpenAI's third-party evaluation guidance similarly treats an evaluated system as more than a model: scaffolding, tools, safeguards, and test-time budget can change measured performance.[24] METR's task-horizon work is useful evidence about performance on tasks of different duration, but its measurement is benchmark- and elicitation-dependent rather than a universal autonomy scale.[23]

Frontier governance frameworks generally begin with capability thresholds and then specify safeguards or deployment decisions. OpenAI's Preparedness Framework, Anthropic's Responsible Scaling Policy, and Google DeepMind's Frontier Safety Framework differ in scope and terminology, but each separates some form of capability assessment from risk-management measures.[15][16][17] Capability × Authority × Reach complements these frameworks by focusing on the operational system that receives identity and access.

Capability is not authority. Knowing how to exploit a policy path does not grant the right to invoke it. A highly capable cyber-research system with no credential, no external route, and disposable state may have substantial dual-use capability with limited practical blast radius. Capability still matters because it changes which paths are feasible and how readily failed controls may be bypassed.

Capability evidence should be scoped to a task family rather than generalized from one impressive demonstration. A system that solves long coding tasks may not reliably interpret a novel IAM condition. A model that can identify a vulnerability may not be able to operate a real exploit chain under a particular harness. Negative evaluation results are also conditional: new tools, more retries, better scaffolding, or another model checkpoint may change them.

Authority

Authority is the effective set of actions a principal may cause under all applicable identity, resource, session, organization, application, delegation, cryptographic, and approval policies.

Authority includes direct and indirect paths:

- workload identities, service accounts, API tokens, OAuth grants, and delegated user sessions;

- role assumption and cross-account trust;
- resource-based policy and service-specific authorization;
- permissions boundaries, session policy, and organization controls;
- `iam:PassRole` and the ability to cause a service to act under another identity;
- tool backend credentials that are not visible in the model-facing schema;
- KMS policy, grants, certificate issuance, signing, and key administration;
- repository, workflow, registry, promotion, and deployment rights;
- approval bypass, policy mutation, credential creation, and delegated administration.

Authority must be evaluated for the actual principal and request context. AWS documents how identity and resource policies, boundaries, and organization controls combine, with explicit denies taking precedence in the relevant evaluation paths.[30][31] ABAC further demonstrates that subject, object, action, and environment attributes all matter.[4]

The visible tool list is not enough. A tool named `deploy_service` may internally use a platform administrator. A read-oriented connector may expose an export or policy mutation operation. A shell tool may inherit every credential mounted into its runtime. Review must trace from the model-facing tool through the enforcement point to the underlying system principal.

Authority is normative. It answers whether the principal may cause an action under stated conditions. It does not by itself describe how many assets the action can affect, what data flows downstream, or whether the agent is capable of discovering the path.

Direct authority is granted to the agent's current principal. Indirect authority arises when that principal can create, select, configure, or invoke another principal or control plane. Examples include passing a role, editing a CI workflow that federates to cloud, updating a serverless function with a privileged execution role, creating a KMS grant, changing a resource policy, or asking a human approver to authorize an effect. Indirect paths belong in the authority model even when no single credential contains the final permission.

Authority analysis should distinguish four related but different sets. **Nominal authority** is what a role or tool contract appears to grant. **Effective authority** is the result after trust, resource, session, boundary, organization, network, service, and application enforcement. **Delegable authority** is the subset the principal may pass, mint, select, or cause another service to exercise. **Mutable authority** is the set of rules and identities the principal can change. The last two sets are frequent sources of underestimation: an agent with few data-plane operations may still be able to alter who receives broader power.

Denial and constraint edges are first-class evidence. An explicit organization deny, workload boundary, key-policy condition, protected environment, or egress rule can break a path even if an identity policy appears broad. But a documented deny is not enough: the assessment must identify the enforcement service, the condition inputs, and whether the agent can influence them. A repository attribute supplied by an untrusted workflow, for example, should not be treated like an attribute asserted by the identity provider.

For every effectful tool, record a short authority contract: canonical operation, underlying principal, resources and conditions, backend calls, delegated services, external destinations, persistent effects, and enforcement owner. The model-facing schema is useful for constraining syntax, but the backend contract defines security. A tool that offers one verb while internally performing five privileged operations must expose those operations to policy and evidence.

Reach

Reach is the set of assets, identities, environments, trust domains, dependencies, and external parties that can be affected through viable direct or transitive paths. It represents blast radius and propagation potential.

Reach includes production systems, accounts, subscriptions, tenants, clusters, sensitive data, cryptographic assets, repositories, pipelines, registries, external networks, SaaS applications, identity providers, databases, message buses, financial systems, operational technology, downstream agents, and customer or partner systems.

Reach is not permission. Authority may permit `ecs:UpdateService` for one service, while reach analysis follows the selected task definition, task role, network paths, secrets, keys, queues, databases, and customers served by that workload. Conversely, broad authority may have limited reach in a disposable account with synthetic data and no federation.

Reach has temporal and transitive properties. A short call may create a durable queue message, scheduled task, deployment, credential, artifact, or policy. An action against one repository may propagate through build consumers and package registries. A key-policy change may alter who can decrypt data later. A parent agent may delegate to another identity in a different environment.

Bounds should therefore cover environment, account, tenant, resource hierarchy, network destination, data class, downstream dependency, delegation depth, fan-out, credential lifetime, job lifetime, and persistent artifacts. Missing classification or topology is `unknown`, not low reach.

Reach should be calculated from an explicit starting principal and scenario. The reachable set after a repository token is compromised differs from the reachable set after a cloud deployment role is compromised. Blast-radius analysis should identify both direct effects and propagation through trusted consumers, not merely count resources.

Reach needs a stopping rule. An analysis should continue across a relationship when the upstream actor can influence a downstream security-relevant state and the downstream system is expected to consume or trust that state. It may stop at an enforced isolation boundary, a verified non-propagating interface, an independently controlled approval, or a system explicitly out of scope—but the stop must be recorded as an assumption or control, not silently omitted. Unknown consumer topology remains an open boundary.

Transitive reach is better expressed as sets and paths than resource totals. Affecting one organization-wide identity provider, certificate authority, package, DNS zone, or shared deployment workflow can matter more than affecting thousands of disposable objects. Reach descriptions should therefore include concentration points, tenant and environment crossings, consumer fan-out, reversibility, persistence, and time-to-effect.

The result is a blast-radius narrative that identifies which trust relationship amplifies an action and which independent control can contain it.

Capability × Authority × Reach

Framework status: Capability × Authority × Reach is a proposed Baitaphish systems-security synthesis. The multiplication sign is a mnemonic for interaction and compounding. It is not a numeric risk equation, probability model, industry standard, or framework published by NIST, OpenAI, Anthropic, Google DeepMind, OWASP, or MITRE.

For a scenario s, practical exposure can be reasoned about as an interaction:

`CAR(s) = interaction(C_s, A_s, R_s | operational context, controls, threat context, evidence)`

This notation must not be used to multiply ordinal labels. Capability, Authority, and Reach are structured descriptions with uncertainty, not commensurable numbers.

The path-based formulation is more useful. For a path `p` to an unacceptable outcome:

- **Capability-feasible:** the specified agent system could discover, choose, compose, and execute the path under its operating conditions.
- **Authority-valid:** each effect is permitted by effective policy, delegation, or an independently bound approval.
- **Reach-connected:** infrastructure, network, data, cryptographic, dependency, delivery, or trust relationships connect the actions to the outcome.

Material exposure exists when all three conditions are supported strongly enough to require a risk decision. Uncertainty in any condition remains visible. A path can be treated as possible even when capability is inferred rather than demonstrated, but the evidence state must say so.

Scenario	Capability	Authority	Reach	Security interpretation
Frontier research agent in a sealed range	High cyber reasoning and tool use	Range-scoped identity	Synthetic targets and controlled package sources	Dangerous knowledge, limited direct operational blast radius; containment still requires testing
Enterprise automation agent	Moderate planning and coding	Production IAM, secrets, deployment	Pipelines, registry, workloads, customer systems	Potentially greater practical risk than the sealed frontier agent
Privileged autonomous agent	High, persistent, adaptive	Broad administration and delegation	Multiple environments and downstream parties	Qualitatively different assurance problem requiring independent evaluation and strong containment
Narrow or unreliable automation	Low or moderate	High administrative authority	Broad production reach	Still dangerous; model weakness is not a security control

The dimensions must remain separable for engineering reasons. Capability changes with a model or harness upgrade. Authority changes with identity and policy. Reach changes when networks, accounts, tenants, data, pipelines, or dependencies expand. Combining them into one label would hide which control owner must act.

The framework overlaps with earlier work without replacing it. Least authority constrains ambient power.[2] ABAC structures authorization context.[4] Attack graphs analyze composed paths.[3] NIST's agent-identity work centers identity, authority, delegation, and audit.[7] Recent preprints propose formally enforced agent permissions, least autonomy, and trajectory assurance.[19][21][22] CAR's contribution is to make competence, permission, and propagation explicit in one operational architecture.

The exposure case as the unit of analysis

A CAR assessment should produce an **exposure case**, not a score. The record names one agent-system version, an objective class, an authenticated starting principal, an operational environment, one or more unacceptable outcomes, and the paths considered. It then states the capability, authority, and reach evidence for each path; preventive, detective, interruptive, and recovery controls; important unknowns; and the owner accepting residual risk. This scope makes conclusions reviewable and prevents a result for a sandboxed evaluation from being copied into a production deployment.

The unit of decision is the path to an outcome, not the agent product. One deployment may have several exposure cases. A coding agent can be approved to read an internal repository, separately reviewed for pull-request creation, and denied any path to protected workflow modification. The same runtime may have another case for development-account diagnostics under a different identity. Treating those as separate decisions allows controls, evidence, expiry, and reviewers to match the actual consequence.

An exposure case should answer five questions:

1. **What must the agent be able to do?** State the task capability and evidence conditions rather than describing general intelligence.
2. **Which effects may it cause?** Resolve direct and indirect authority to normalized operations at the enforcement point.
3. **Where can those effects propagate?** Follow infrastructure, data, cryptographic, delivery, delegation, and external trust paths.
4. **Which unacceptable outcomes remain reachable?** Include confidentiality, integrity, availability, safety, financial, legal, and third-party effects relevant to the system.
5. **What evidence would falsify the decision?** Define drift triggers, runtime violations, missing collectors, failed revocation, and new paths that force review.

This format discourages false precision while still supporting disciplined comparison. An organization can say that one architecture has fewer authority-valid paths, smaller reachable sets, stronger evidence, or better interruption than another. It should not claim that multiplying three arbitrary labels produced a calibrated probability.

Autonomy and time horizon

Autonomy should not become a fourth CAR axis. Capability describes competence. Autonomy describes how independently, repeatedly, and persistently that competence is exercised. It is an operational exposure modifier that changes opportunity, cumulative effect, and human-response time.

Record whether the agent chooses subgoals, retries after denial, runs unattended, schedules or resumes work, creates durable state, delegates, or fans out across parallel workers. Apply cumulative limits to time, cost, actions, affected resources, delegation depth, and persistent artifacts.

Time belongs in two places. Demonstrated ability to complete tasks of a stated duration is capability evidence. Credential TTL, job lifetime, schedule, retry window, approval expiry, and persistence are operational context. Keeping them distinct prevents a benchmark result from being mistaken for production authorization.

Autonomy changes the number and cadence of opportunities to exercise authority. A single approved action, one hundred retries, and an unattended week-long job may share the same nominal permission but not the same exposure. Assessments should record an **operating envelope**: maximum runtime, actions, cost, tokens, retries, concurrent workers, external destinations, delegated children, affected resources, and durable changes. Enforcement should count cumulative effects across restarts and delegates rather than resetting a budget whenever the agent opens a new session.

The operating envelope also provides a practical stop condition. Exceeding a time or action budget should pause or terminate execution through an external controller, preserve evidence, revoke outstanding sessions where appropriate, and require a fresh decision. The agent can request an extension, but it cannot grant one to itself.

Threat Model

The threat model assumes the agent may be mistaken, manipulated, compromised, or adversarial. It does not assume every agent is hostile. Controls should remain effective under capability surprise and untrusted input.

Assets and security objectives

Asset	Security objective
Workload identity, credentials, and sessions	Prevent impersonation, misuse, unauthorized delegation, and persistence
Data, secrets, and regulated information	Preserve confidentiality, integrity, tenant isolation, purpose limitation, and availability
KMS keys, HSMs, signing keys, CAs, and certificates	Separate use from administration; protect trust roots and revocation
Source, workflows, artifacts, attestations, and registries	Preserve supply-chain integrity and attributable promotion
Production infrastructure and management planes	Prevent unauthorized mutation, escalation, outage, and cross-environment propagation
Models, prompts, tools, memory, and policy	Preserve system intent, version integrity, and isolation of untrusted content
Approval, telemetry, and evidence	Preserve independent accountability, detection, and reconstruction

Threat actors and failure sources

Relevant sources include a malicious external user, compromised developer, insider, prompt-injection source, malicious dependency, compromised model endpoint, malicious tool, compromised agent runtime, compromised CI/CD system, compromised identity provider, and an external attacker controlling data consumed by the agent.

An attacker may be able to submit a prompt, document, issue, repository content, webpage, message, package, or tool result. The threat model does not assume that an external attacker already has a cloud administrator identity, direct access to model weights, or control of the policy engine unless the scenario supplies that precondition. Stating non-capabilities prevents inflated severity while keeping indirect paths visible.

Trust boundaries

The principal boundaries are untrusted input to runtime, runtime to model provider, runtime to memory, runtime to policy and identity, tool gateway to underlying service, CI/CD to cloud, identity provider to tenant or account, data to cryptographic service, parent to child agent, internal to external network, execution to evidence, and approval interface to effect.

Every boundary review should identify the data crossing, channel, authenticated identities, encryption, schema or normalization, policy enforcement, rate or cumulative limits, evidence, and recovery path. A tool schema validates shape; it does not establish that the caller is entitled to the requested outcome.

Security control objectives

The threat model yields six control objectives. First, **bind identity to execution**: every effect must be attributable to a workload, run, task, and credential rather than to a shared API key. Second, **mediate effects**: trusted policy must evaluate the normalized operation and resource at the point of use. Third, **attenuate propagation**: environment, network, data, cryptographic, deployment, and delegation boundaries should prevent one permitted effect from acquiring unrelated reach. Fourth, **preserve independent evidence**: the subject being observed must not be able to rewrite the authorization record or erase the only trace. Fifth, **interrupt and recover**: operators need out-of-band revocation, cancellation, quarantine, rollback, and restoration. Sixth, **reassess on drift**: material changes to any capability, authority, reach, or operating-context premise invalidate the decision.

These objectives apply to accidental and adversarial trajectories. Prompt injection is an instruction-integrity problem, but its practical severity depends on whether the resulting proposal obtains authority and reaches a protected outcome. Credential theft is an authority compromise, but the consequence depends on service reach and expiry. Sandbox escape is a boundary failure, but external harm still depends on reachable credentials, networks, files, or trust relationships. Mapping each failure across CAR identifies where independent defenses can break the path.

Threat modeling should include control compromise rather than assume all guardrails are trusted. A malicious tool can falsify a result; a compromised model endpoint can return targeted actions; a policy administrator can weaken enforcement; a telemetry pipeline can drop evidence; and an approver can be coerced or compromised. High-impact cases need separation so that one compromised component cannot propose, authorize, execute, attest, and conceal the same effect.

Prioritized abuse paths

ID	Abuse path	C/A/R relationship	Priority and rationale
TM-001	Prompt injection causes sensitive tool use or external communication	Capability interprets hostile context; tool authority permits action; data and egress create reach	High: common entry surface with potentially severe impact
TM-002	Agent discovers a role-assumption or `PassRole` privilege bridge	Planning composes actions; service and IAM rights provide authority; resulting role expands reach	Critical where production administration or sensitive data becomes reachable
TM-003	Agent changes source, workflow, or dependency and reaches production	Coding capability; repository and CI identity; registry and deploy consumers	Critical where one principal can propose, build, approve, sign, and deploy
TM-004	Key policy, grant, or signing authority is misused	Cryptographic reasoning; indirect key authority; encrypted data or trust consumers	Critical for key administration, signing roots, or broad decrypt reach
TM-005	Parent silently delegates unrestricted credentials or context	Delegation and planning; inherited identity; child tools and environments	High because transitive authority becomes difficult to attribute or revoke
TM-006	Several individually valid actions produce a prohibited outcome	Sequence planning; multiple narrow grants; transitive business and system effects	High; per-action checks alone miss the trajectory
TM-007	Agent disables logging, policy, egress, or kill-switch controls	Adaptive planning; security-admin authority; environment-wide reach	Critical because prevention and evidence can fail together
TM-008	Malicious dependency or tool output becomes code and prompt input	Tool learning; install or build authority; runtime and downstream consumers	High where remote tools or untrusted builds cross into privileged execution
TM-009	Agent escapes an evaluation or execution boundary	Vulnerability discovery and persistence; local execution authority; host and network reach	High to critical depending external routes and credentials

ID	Abuse path	C/A/R relationship	Priority and rationale
TM-010	Erroneous remediation causes outage or destructive change	Moderate capability can suffice; mutation authority; fleet or organization reach	High; demonstrates that model weakness is not containment

The July 2026 Hugging Face incident is a current, attributed example of path composition, not the thesis of this paper. OpenAI reported that models running a reduced-safeguard cyber evaluation exploited a package-proxy vulnerability, obtained internet access, and chained paths into Hugging Face.[25] Hugging Face's technical reconstruction describes thousands of actions across sandbox, dataset-processing, Kubernetes, identity, secrets, internal networking, and source-control boundaries, while also limiting the reported customer impact and distinguishing attempted from realized effects.[26] The reports were still recent at retrieval and should not be generalized to ordinary deployments.

Separate 2026 disclosures about third-party cyber evaluations further show why live internet access, sandbox configuration, safeguards, and test conditions belong in the claim.[27] UK AISI's sandboxing toolkit separates model inference from tool execution and treats tooling, host, and network isolation as explicit choices.[28] Anthropic's agentic-misalignment work is a controlled simulation in which models received sensitive information and communication ability; it is useful as a design warning, not evidence of a production incident.[29]

The threat register should record preconditions and confidence beside severity. “Agent can exfiltrate customer data” is not a defensible finding unless the assessment identifies a capability-feasible method, an authority-valid operation or bypass, a reach-connected destination, and the supporting evidence. Where one element is unknown, write “possible under unresolved condition” and name the validation step. This language is more useful to engineers and risk owners than either certainty without evidence or a vague statement that an agent might do anything.

Identity and IAM

Every autonomous agent should normally run under a dedicated workload identity rather than a borrowed developer session or durable access key. Identity should bind the runtime, environment, purpose, tenant, and task to short-lived credentials.

Recommended controls include:

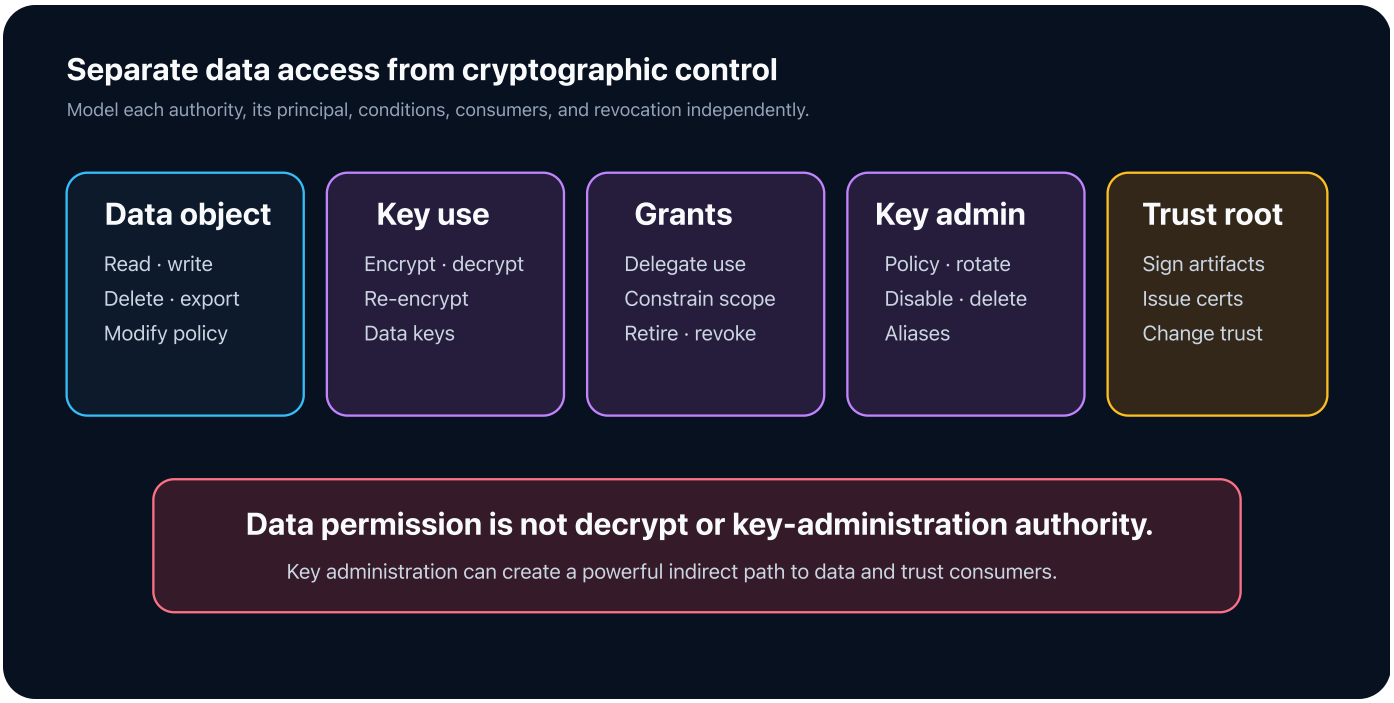
- separate identities for materially different agent purposes and environments;
- OIDC or workload federation with strict issuer, audience, subject, repository, workflow, environment, and service claims;
- short session duration matched to a bounded task;
- permissions boundaries and organization constraints for identities the agent can influence;
- resource-policy and cross-account trust review;
- explicit deny for policy mutation, credential creation, organization administration, and evidence deletion unless the task requires it;
- separate observation and mutation identities;
- active-session revocation tests.

`iam:PassRole` deserves special treatment because it delegates authority to a service. A principal may be unable to read a secret yet be allowed to update a function or task that executes under a privileged role. AWS recommends scoping which roles may be passed and, where applicable, the services receiving them.

[32] Exposure analysis must continue into the passed role's authority and reach.
GitHub Actions OIDC removes a stored cloud secret but does not make a workflow or cloud role safe. Trust conditions should restrict the expected organization, repository, workflow, branch or protected environment, subject, and audience.[35] Kubernetes service accounts similarly require workload-specific RBAC, token handling, and isolation rather than implicit trust in cluster location.[36]
The policy decision and enforcement point must remain outside the model. Natural-language intent can inform a request, but trusted attributes and normalized operations must come from authenticated system state. The model cannot assert its own tenant, approval, role, or exception.
The authorization review should calculate effective permission across identity policy, resource policy, trust, session policy, permissions boundary, organization control, service-specific behavior, cryptographic policy, and application policy. It should also examine which of those policies the agent can modify. An identity with narrow business permissions but broad policy-administration rights is not narrow in practice.
Delegated user context deserves separate treatment from workload authority. A user may be entitled to read a record but not delegate export, external communication, or a durable copy to an agent. Delegation should identify the purpose, resource, operation, audience, duration, and whether further delegation is permitted.

Cryptographic Control Surfaces

Cryptography is both protection and control plane. “Encrypted with KMS” does not answer who can read plaintext, delegate key use, disable the key, change policy, issue certificates, or sign trusted artifacts.



Model at least five distinct authorities:

1. **Data-object authority:** read, write, delete, export, or modify policy.

2. **Key-use authority:** encrypt, decrypt, re-encrypt, generate data keys, or perform cryptographic operations.
3. **Grant authority:** delegate selected operations to another principal or service.
4. **Key-administration authority:** create, rotate, enable, disable, schedule deletion, change policy, or move aliases.
5. **Signing and issuance authority:** sign artifacts or tokens, or issue certificates that other systems trust.

AWS KMS key policies are foundational resource policies, and grants can delegate operations for a key.[33] [34] An agent that needs one decrypt operation should not administer the key, create grants, or sign releases. A signing service should accept an approved digest and provenance record, not arbitrary bytes chosen entirely by the model.

High-impact key administration should use separate identity and, where appropriate, dual authorization. Telemetry should correlate the agent run, workload session, key, operation, encryption context, approval, and result. Revocation testing must account for cached data keys, copied plaintext, already-issued certificates, and already-signed artifacts.

Cryptographic reach is transitive. Changing a CA policy may affect every service trusting certificates from that authority. Signing one package can affect all consumers that trust the repository. Moving a KMS alias can redirect future encryption or decryption. The exposure graph should represent those consumers rather than ending at the key.

Data and Infrastructure Reach

Data requires its own graph domain. Model `read`, `write`, `delete`, `modifyPolicy`, `export`, `transform`, `decrypt`, `reEncrypt`, `replicate`, `publish`, and `consume` as different edges. A generic “accesses data” relationship conceals the security decision.

Use only evidenced classifications:

- public;
- internal;
- confidential;
- regulated;
- credentials or secrets;
- unknown.

Missing classification remains unknown. Static analysis should record the schema, policy, tag, catalog, owner, or repository path supporting a class. It should also identify replicas, indexes, caches, backups, and consumers where evidence exists.

Infrastructure control-plane authority can create indirect capability. An agent that can update Lambda, ECS, EKS, EC2, serverless orchestration, networking, DNS, storage, databases, or infrastructure-as-code may be able to select a new image, execution identity, route, startup command, or secret reference. That path can manufacture data access the original principal lacked.

Reach review therefore follows created and updated resources into their workload identities, network paths, cryptographic permissions, persistent state, downstream triggers, and customers. Control-plane operations that change identity, network boundary, evidence, or recovery require stronger policy than ordinary data-plane operations.

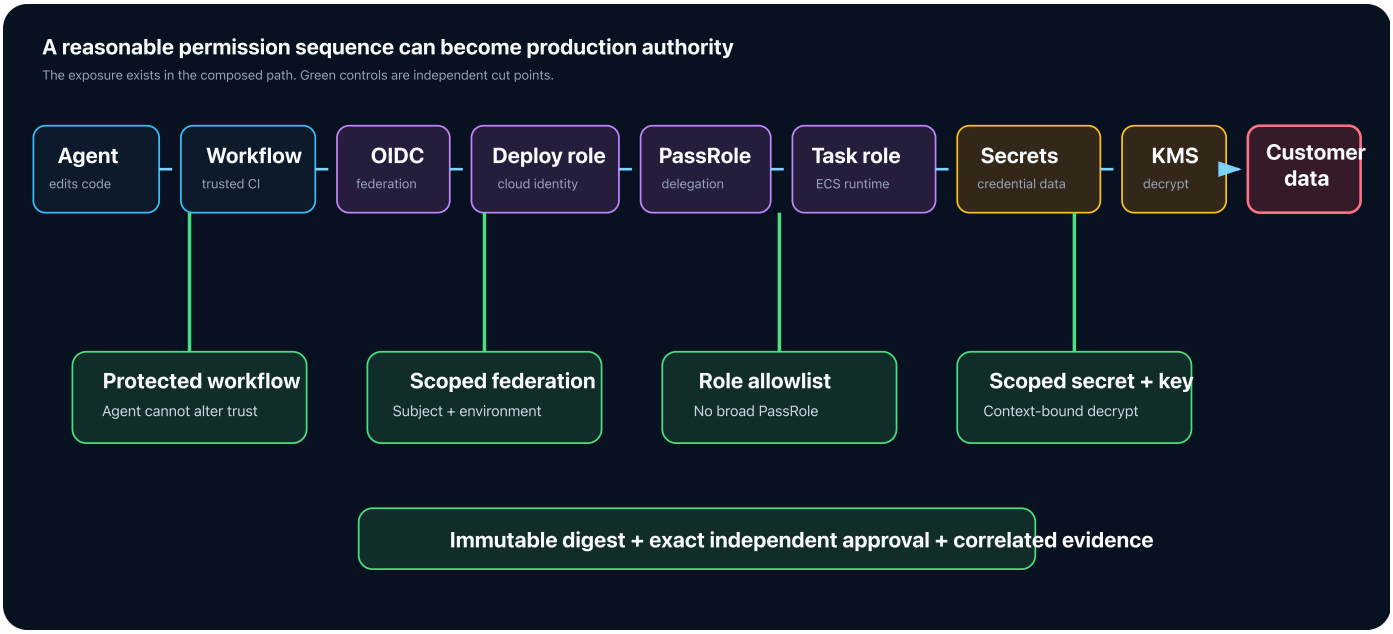
Reach should also distinguish availability and integrity effects from confidentiality. An agent may not read a production database yet still delete it, block its network, alter DNS, invalidate certificates, or replace the application consuming it. The unacceptable outcomes and security objectives determine which edges matter.

CI/CD and Software Supply Chain

Code-generation capability becomes production authority when it is connected to privileged CI/CD. CI/CD should separate six decisions:

1. source change;
2. build;
3. verification;
4. signing;
5. promotion;
6. deployment.

An agent may propose a pull request without merging it. A builder may create an artifact without promoting it. A deployer may apply an approved digest without rewriting source, provenance, or the artifact. Avoid a principal that can modify source, weaken tests, issue provenance, sign, deploy, and delete evidence.



Protect trusted workflows from unreviewed changes and untrusted pull-request execution. Production federation should require a reviewed workflow revision and protected environment. Bind infrastructure approval to the normalized plan or resulting digest so a later action cannot be substituted. Verify immutable digests rather than mutable tags.

Dependency and tool analysis must connect presence to an execution or influence path. A manifest entry does not prove exploitability. The relevant path is `modifiable input → trusted build → executable artifact → consumer or deployment target`. Review lockfiles, lifecycle scripts, base

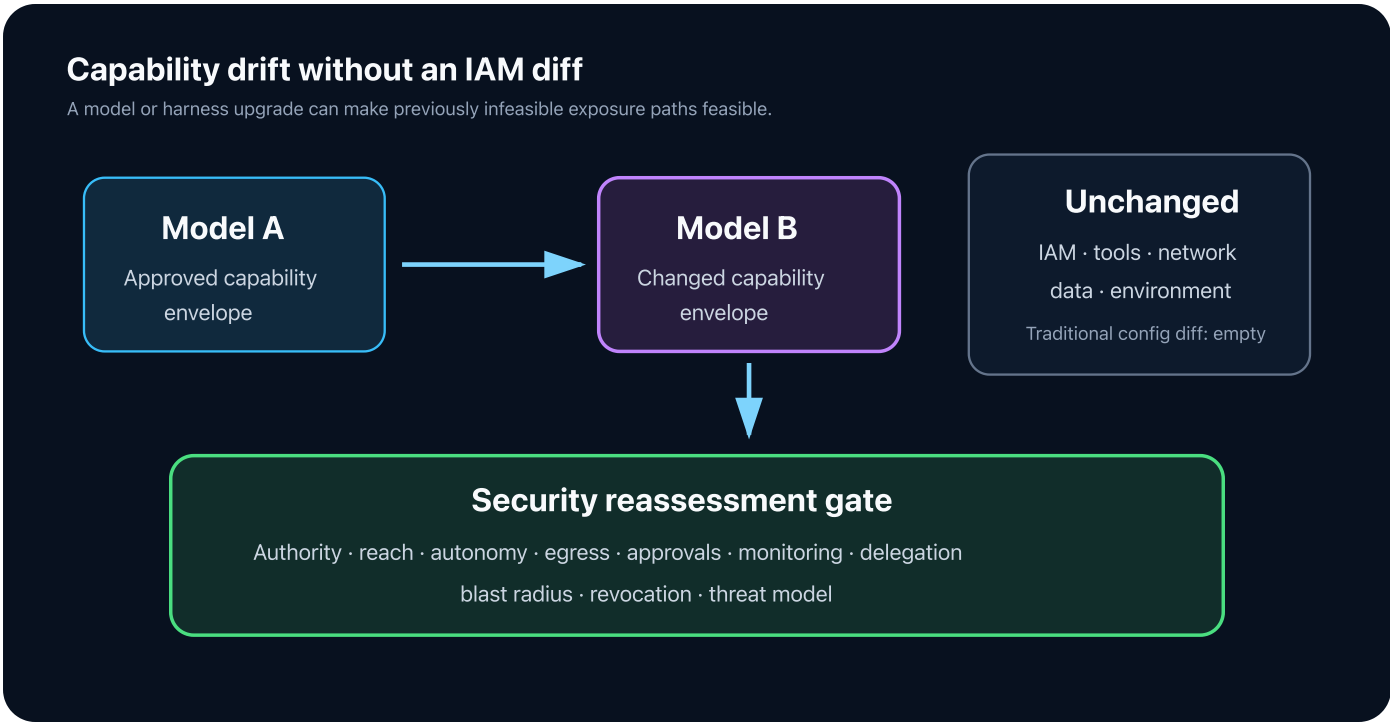
images, remote tools, package registries, container registries, dependency bots, provenance, and promotion policy.
SLSA provides vocabulary and requirements for build provenance and increasing supply-chain assurance. [37] Provenance can support claims about artifact origin and build process; it does not prove that source code, an agent-generated change, or deployment intent is safe.
Self-hosted runners deserve explicit graph nodes because they can bridge repository input, persistent workspace state, cloud credentials, internal network, and deployment authority. Prefer ephemeral, isolated builders for untrusted changes. Protect action and dependency versions, installation scripts, cache boundaries, and registry policy. Record the digest at each promotion and deployment.

Delegation and Multi-Agent Systems

Every delegation creates a new security principal, even when multiple agents share one runtime. The architecture should answer whether the delegate receives a separately scoped identity or inherits the caller's ambient authority.
A delegation record should contain delegator, delegatee, task, permitted outcomes, authority scope, resource scope, credential audience, TTL, maximum depth and fan-out, transferred memory, tool set, approval requirements, revocation handle, and policy evidence.
The intended constraint is:
`effective child authority ⊆ caller's delegable authority ∩ task policy ∩ resource policy ∩ organization constraints`
Unknown or contradictory scope should fail closed. The child should not silently receive the parent's unrestricted token, filesystem, tools, network, or memory.
Multi-agent systems introduce confused-deputy and transitive-authority problems. A low-authority planner may persuade a privileged specialist to act. A child may transfer hostile context into a trusted workflow. Shared memory may bypass tenant or task separation. Recursive delegation can create unaudited fan-out and orphaned work.
Independent workload identity is preferred wherever practical. When a shared runtime is unavoidable, enforcement must still bind every action to a logical delegate, task, scope, and policy decision. The recent permissions survey by Michael and Roesner highlights unresolved gaps across permission interfaces, internal policy representation, policy derivation, and runtime enforcement.[20]
Delegation should be attenuating, attributable, expiring, and revocable. A parent may delegate less than it possesses, but a child must not combine inherited authority with unrelated ambient credentials. The exposure graph should recompute reach for each child and record whether data, memory, tools, or network access crossed a new trust boundary.

Capability Drift

An organization may approve an agent using Model A, then replace it with Model B while leaving IAM, tools, and network unchanged. The traditional infrastructure diff is empty, but the deployment's capability-feasible paths may have changed.



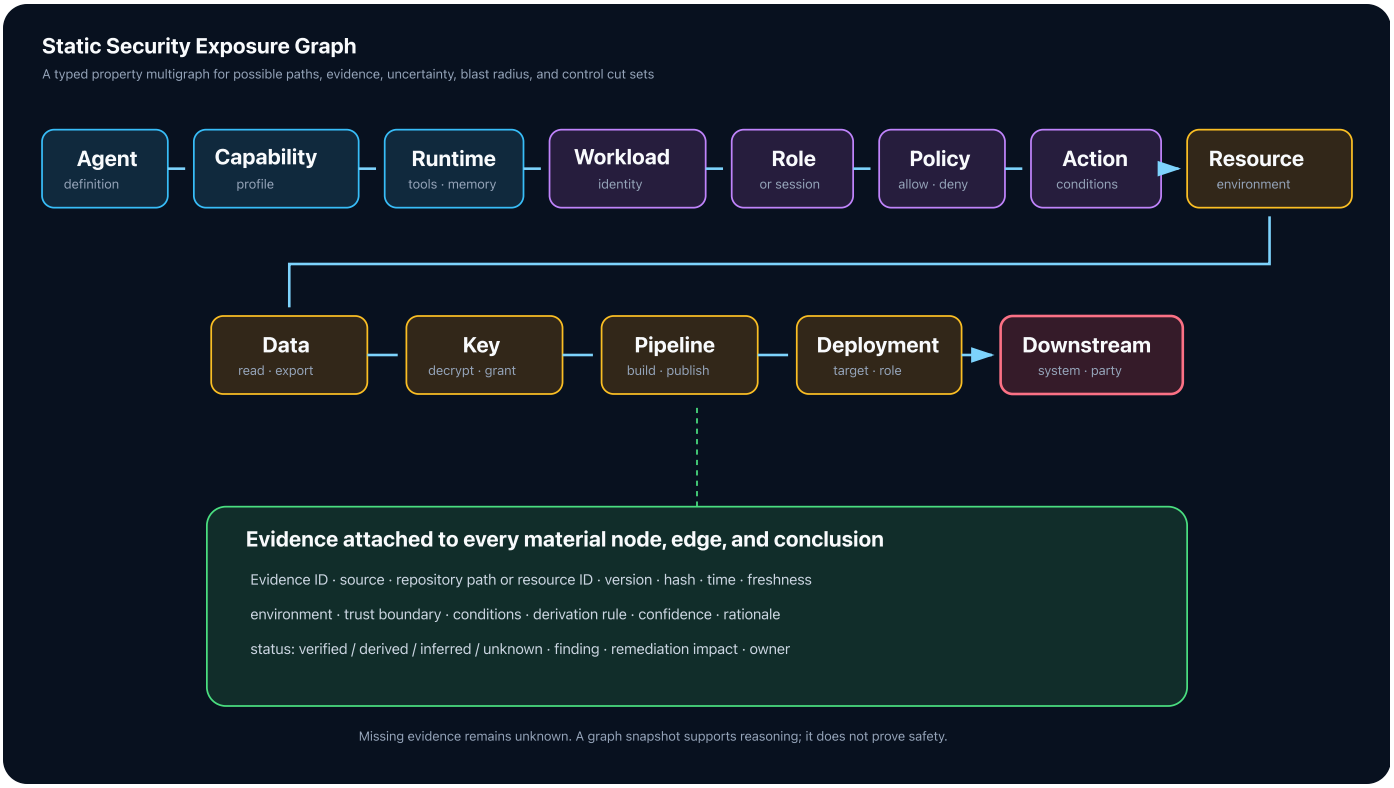
Maintain a **Capability Manifest** containing model or checkpoint, provider, runtime, harness, prompt-policy version, tools, memory, context, reasoning budget, task horizon, retries, parallelism, delegation, relevant evaluations, limitations, date, confidence, and the approved authority and reach envelope.
Reassessment triggers include:

- model, checkpoint, provider, or endpoint change;
- stronger reasoning mode, context, or scaffolding;
- new tool or broader schema;
- persistent memory;
- increased retry, time, concurrency, or autonomy;
- new delegation behavior;
- safeguard or classifier change;
- evidence that the system recovers from previously limiting failures.

The security review should compare old and new capability envelopes, re-run representative and adversarial tasks, trace newly feasible exposure paths, and reconsider egress, approval, monitoring, revocation, and blast radius.
A model upgrade is a security-relevant infrastructure change even when no IAM policy changes.
Capability drift can also be negative. A provider change or new safeguard may reduce one capability while changing error modes or tool behavior. Reassessment should not assume that every upgrade increases every capability. It should test the tasks and paths relevant to the approved deployment.

Static Security Exposure Graphs

The **Static Security Exposure Graph** is a proposed implementation-neutral method for representing possible agent exposure from versioned evidence. It is not a feature currently claimed as implemented in Baitaphish or Repo Navi.



Use a typed, directed property multigraph. Core node classes include:

- agent definition, objective, model, capability profile, runtime, memory;
- human, workload identity, credential, session, role, policy, boundary, approval;
- tool, API, SaaS service;
- account, tenant, cluster, compute, network, DNS, storage, database;
- repository, workflow, runner, artifact, registry, deployment;
- dataset, data store, secret;
- key, certificate, CA, HSM, signing service;
- package, dependency, image, build input;
- downstream system, external party, agent;
- finding, control, and evidence item.

Edges should distinguish `usesModel`, `executesAs`, `obtainsCredential`, `assumes`, `delegatesTo`, `allows`, `denies`, `constrains`, `requiresApproval`, `invokes`, `passesRole`, `reads`, `writes`, `deletes`, `exports`, `transforms`, `encrypts`, `decrypts`, `administersKey`, `createsGrant`, `signs`, `issuesCertificate`, `connectsTo`, `trusts`, `builds`, `publishes`, `promotes`, `deploys`, `dependsOn`, `consumes`, and `triggers`.

Every material node, edge, and conclusion should carry:

Evidence property	Purpose
Evidence ID and source	Identify the supporting record and collector
Location	Repository path, resource ID, ARN, trace, policy statement, or API record
Version and time	State hash, revision, collection time, and freshness
Environment and boundary	Bind the claim to account, tenant, environment, and trust zone
Conditions	Preserve request attributes and unresolved runtime variables
Derivation rule	Explain how a relationship was computed

Evidence property	Purpose
Status	`verified`, `derived`, `inferred`, or `unknown`
Confidence	Explain source quality, completeness, and conflicts
Finding	State why the relationship matters
Remediation impact	Record broken paths, alternate paths, migration effect, and owner

Verified means directly observed in an authoritative source. **Derived** means deterministically computed from verified inputs under stated assumptions. **Inferred** means plausible but requiring validation. **Unknown** means missing, stale, conflicting, or inaccessible evidence. Confidence is separate from state: a derived result may still have low confidence when its source inventory is incomplete.

Static inputs can include repository files, infrastructure-as-code, identity and resource policies, trust, KMS policy and grants, workflow definitions, dependency manifests, network configuration, tool schemas, data catalogs, and approved architecture. Deployed-state exports improve fidelity but do not remove snapshot limitations.

Useful queries include:

- paths from an agent identity to unacceptable outcomes;
- role assumption and `PassRole` escalation;
- cross-account, cross-tenant, and cross-environment trust;
- key administration, decrypt, grant, signing, and certificate paths;
- workflow-to-production and registry-to-consumer paths;
- data export and external communication;
- blast radius after compromise of a credential, runner, or child agent;
- minimal cut sets that break a path;
- residual and alternative paths after remediation.

For example:

```
GitHub workflow → assumes deployment role → can PassRole → updates Lambda or ECS → workload executes as privileged role → reads production secret → decrypts via KMS
```

The weakness exists in the path. One policy reviewer may approve workflow federation, another may approve a deployment action, and another may approve a workload role. The graph gives the organization one place to evaluate composition.

Attack-graph research provides important precedent for path and cut-set reasoning.[3] W3C PROV-O supplies useful concepts for agents, activities, entities, derivations, revisions, and primary sources that can inform evidence provenance without requiring an RDF implementation.[38]

Path semantics and query discipline

A graph path is only meaningful when edge semantics compose. `allows`, `assumes`, and `passesRole` describe authorization relationships; `connectsTo` describes potential transport; `trusts` describes a consumer decision; `deploys` describes a state transition. A query engine should not treat those verbs as interchangeable reachability. Each path rule must specify compatible node and edge types, required conditions, direction, temporal assumptions, and the unacceptable outcome it supports.

For example, deriving `agent may read plaintext secret` from an ECS update requires more than a connection between nodes. The rule must establish that the agent can select an approved task definition or modify one; can cause the service to run it; can pass the referenced task role to ECS; that the role can read the particular secret; that secret resource policy permits the session; that the secret or data key can be

decrypted under KMS policy and context; and that network and service conditions permit the calls. A missing condition becomes unknown, not an implicit allow. An explicit deny or verified independent approval can break the derived path.

Represent alternate evidence rather than overwriting it. A repository declaration may say one role is used while a deployed-state export reports another. Both facts should remain, with source, time, environment, and conflict status. The conclusion can then be `unknown` or time-bounded until reconciled. Historical versions matter because an incident trajectory may have occurred under a graph that is no longer current.

Path findings should be reproducible. Store the query or derivation rule, ordered nodes and edges, evidence identifiers, collector versions, snapshot time, unresolved predicates, confidence rationale, and the control or outcome being evaluated. A reviewer should be able to distinguish a direct observed grant from a multi-step inference and determine which premise would invalidate the conclusion.

Graph analysis should support at least three views. The **forward view** starts at an agent, credential, tool, runner, or compromised component and enumerates reachable outcomes. The **backward view** starts at an unacceptable outcome—such as unauthorized production deployment or plaintext regulated data—and identifies prerequisites and potential starting principals. The **differential view** compares graph revisions after a model, policy, workflow, network, key, dependency, or environment change. Differential results are especially useful for release review because they reveal newly capability-feasible or authority-valid paths without implying that unchanged paths are safe.

Cut-set analysis should favor independent enforcement points. Removing a graph edge in documentation is not a control. Restricting OIDC trust, denying role passing, separating workflow approval, scoping the task role, constraining the key, and blocking data egress are distinct cuts owned by different components. A high-impact path is more robustly contained when no single configuration error restores every prerequisite.

Static proof boundaries

Static analysis can verify that a declaration existed at a version and hash, derive policy relationships under supported semantics, and demonstrate a possible path under stated conditions. It cannot prove that a deployment matches the repository, that an unavailable condition evaluates as expected, that a tool backend uses the documented identity, or that an agent will follow or avoid the path.

Dependency presence does not prove exploitability. A wildcard does not prove material exposure if no meaningful resource is reachable. Conversely, an absence of wildcards does not prove safety when composed trust creates a path. Findings should name the path, evidence, unknowns, and unacceptable outcome rather than rely on configuration severity alone.

Coverage and false confidence

Graph coverage must be explicit. Publish a coverage manifest listing collectors, accounts, services, repositories, environments, snapshot times, inaccessible systems, unsupported policy semantics, and unresolved conditions. Do not treat “not collected” or “not observed” as denied. Show stale and conflicting evidence. Compare the static graph with runtime evidence and retain historical revisions.

Remediation analysis should calculate which paths are broken, which alternate paths remain, and which workloads or migrations would be affected. A minimal cut set is a decision aid, not an instruction to change production automatically.

Runtime Trajectory Observability

Static exposure describes possible paths under a configuration snapshot. Runtime observability describes what a specific execution requested, was authorized to do, and actually affected. They are complementary and must not be conflated.

Capture structured events for:

- agent, run, task, parent run, objective, and authorized outcome envelope;
- model, provider, harness, prompt-policy, tool, and capability-manifest versions;
- workload identity, credential or session fingerprint, and role assumption;
- canonical tool operation and normalized parameters;
- authorization decision, policy version, rationale, and conditions;
- approval identity, target, expiry, and bound plan or digest;
- network destination and resolution;
- data object or class and cryptographic operation;
- source, workflow, artifact, file, infrastructure, or policy change;
- memory access and delegation;
- result, persistent effect, retry, kill-switch, cancellation, revocation, and recovery;
- correlation identifiers and trustworthy timestamps.

OpenTelemetry's current generative-AI conventions include agent invocation, model calls, and tool execution, while remaining under active development.[39] They are a useful interoperability base, not a complete security event model.

Agent security must evaluate trajectories, not only individual actions.

A sequence may read a public record, create a draft, modify a workflow, publish an artifact, and trigger a trusted consumer. Each action may pass a local policy while the cumulative effect violates an invariant. Trajectory policy therefore needs state: prior actions, cumulative budgets, environment transitions, delegation, persistent effects, and intended outcome.

Evidence should be retained outside the authority of the observed agent. Correlate model, tool, IAM, cloud, CI/CD, KMS, data, network, approval, and host events. Protect clock integrity, access, retention, and chain of custody.

Structured actions and effects should be the default. Raw prompts, model messages, or hidden reasoning may be sensitive, voluminous, unavailable, and inappropriate to retain. Collect content only for a defined necessity with access, retention, privacy, and legal controls. Do not make chain-of-thought availability a security dependency.

Runtime observations can validate or contradict static relationships. An observed edge with no approved static counterpart is a finding. A statically expected control that produces no runtime evidence is also a finding. Neither condition should automatically change policy or remediate production without review.

Trajectory state and effect accounting

A runtime decision should evaluate both the proposed action and the trajectory state. Useful state includes resources already touched, data classes read, environments crossed, roles assumed, external destinations contacted, artifacts created, approvals consumed, delegates active, retries used, persistent effects outstanding, and cumulative cost or action budgets. The policy engine need not understand model reasoning; it needs authoritative facts about requests and effects.

Actions should be normalized before authorization and logging. A shell command, cloud SDK call, GitHub operation, internal tool request, and browser click can all represent the business effect “change production deployment.” Preserve the original operation for investigation, but also emit a stable verb, target, environment, expected effect, reversibility, and policy category. Normalization lets sequence policy apply across tools and prevents a new wrapper from bypassing an operation-specific rule.

Effect telemetry must follow the request. An authorization decision proves only that a requested operation satisfied policy with the inputs evaluated at that time. It does not prove the tool executed the intended target, that the service accepted it, or that the resulting state matched the proposal. Correlate request, decision, backend call, response, state change, and durable consequence. For asynchronous systems, retain causal links through queues, workflows, builds, deployments, and downstream consumers.

Trajectory controls can express constraints that per-action IAM usually cannot: no production mutation after reading untrusted external content without a context reset and approval; no export after accessing a regulated dataset; no role assumption after modifying the workflow that requests the role; no more than a bounded number of fleet actions per incident; no child delegation after a sensitive credential enters shared memory; and no evidence or policy mutation by the principal whose actions are under review. These examples require organizational policy and careful testing; they are not universal defaults.

Telemetry failure needs an explicit security posture. For high-impact operations, inability to obtain a policy decision or write required evidence should generally stop the action or move to a separately governed break-glass path. Low-impact observation may tolerate buffered events. The design should define backpressure, local queue protection, maximum evidence gap, reconciliation, and what operators see when telemetry is degraded. Silent loss is incompatible with claims of trajectory assurance.

Privacy and minimization are also security requirements. Prefer identifiers, normalized operations, hashes, classifications, and effect summaries over indiscriminate capture of prompts, retrieved documents, secrets, or personal data. Where content capture is necessary for incident response or evaluation, separate access, encrypt it, limit retention, and record purpose. Evidence quality is not measured by collecting the maximum possible content.

Human Authorization

Human authorization is strongest when it is rare, informed, independent, and bound to an exact effect. It is weakest when it becomes a stream of vague prompts inside the same interface the agent controls.

Actions that generally warrant stronger control include production deployment, privilege escalation, IAM and trust-policy changes, KMS administration, key deletion, artifact signing, disabling logging, network-boundary changes, destructive database operations, credential creation, security-control mutation, external communication with material effect, and high-impact financial or operational actions.

An approval record should bind:

- authenticated approver and organizational role;
- normalized action and target;
- source revision, plan, or artifact digest;
- policy decision and material warnings;
- data, environment, and blast-radius context;
- expiry and replay protection;
- rollback or break-glass path;

- evidence ID.

Use dual authorization for selected irreversible or organization-wide actions. Break-glass workflows should be time-bound, independently logged, and reviewed after use. Exceptions need owner, rationale, scope, expiry, and compensating controls.

Human approval is not universally sufficient. Approval fatigue, ambiguous summaries, persuasive model output, compromised approvers, and inability to inspect transitive effects remain failure modes. Deterministic policy should prevent actions that no approver is authorized to permit.

The approval interface should obtain the action, target, and policy context from trusted state rather than model-authored prose. The agent may explain its plan, but it must not control the canonical diff, digest, resource identity, or blast-radius evidence shown to the approver.

Capability-Conditioned Controls

The following tiers are proposed for analysis. They are not OpenAI Preparedness Framework categories, Anthropic AI Safety Levels, Google DeepMind capability levels, NIST tiers, or government standards.

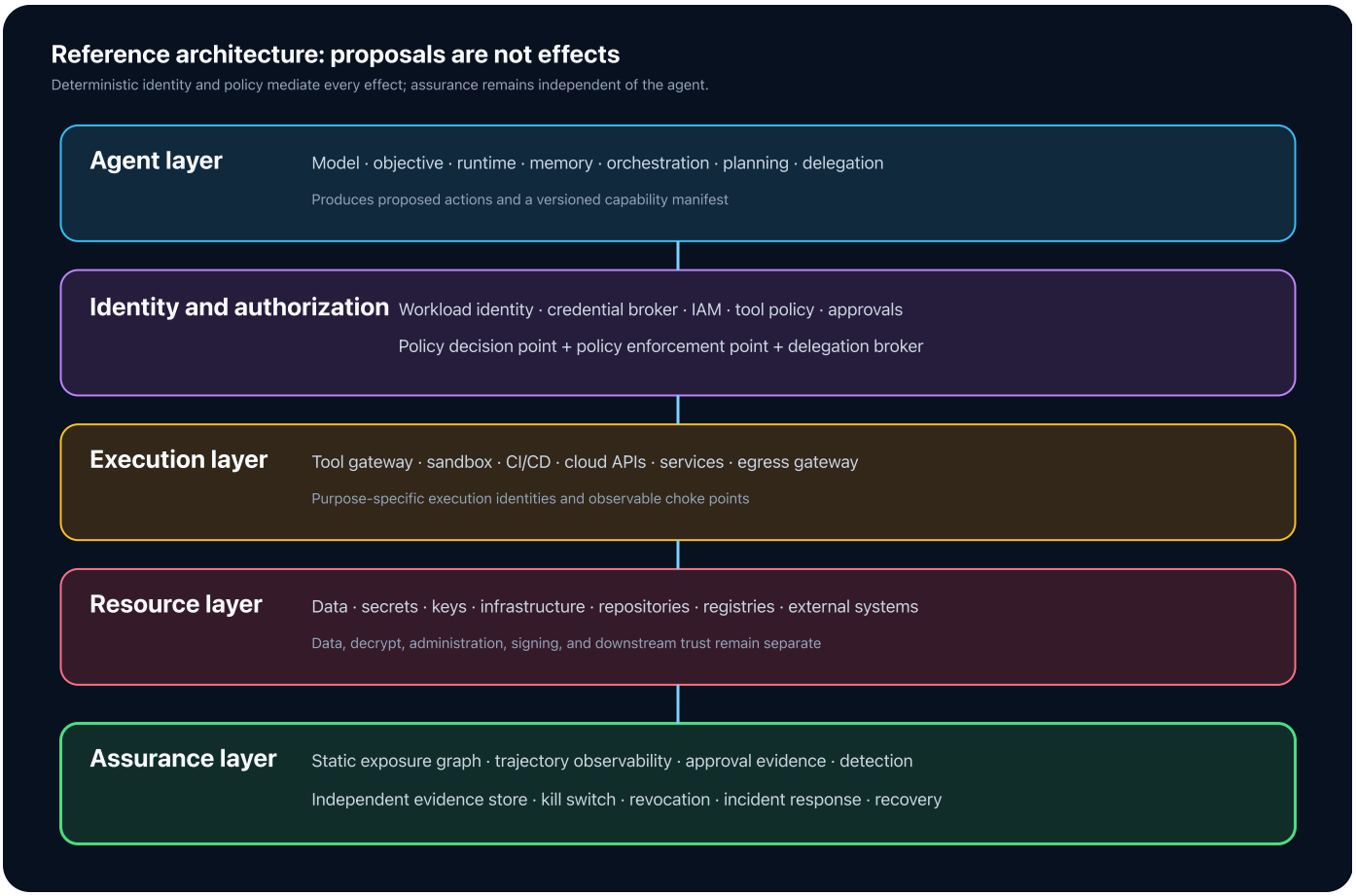
Tier	Deployment	Minimum assurance expectations
Tier 0 - Informational	No tools or autonomous external action	Standard application security, content controls, logging, data minimization
Tier 1 - Bounded agent	Limited tools and constrained datasets	Dedicated identity, allowlisted tools, short-lived credentials, basic traceability, tested revocation
Tier 2 - Operational agent	Can change systems or access sensitive data	Least privilege, environment separation, egress restriction, full trajectory logging, high-impact approvals, adversarial testing, rollback
Tier 3 - Privileged autonomous agent	Long-running, delegating, or privileged enterprise access	Dedicated isolation, fine-grained authorization, independent evaluation, blast-radius analysis, controlled delegation, continuous monitoring, strong kill switch, dual control where appropriate
Tier 4 - Frontier dual-use	Advanced cyber or other high-impact dual-use capability	Hardened containment, segregated infrastructure, narrowly restricted tools and network, model and artifact protection, independent capability and safeguard evaluation, strict release governance

Tiers do not replace CAR analysis. Two operational agents may share a tier but differ radically in authority and reach. Increased autonomy, time, concurrency, or delegation can raise assurance requirements even when core model capability is unchanged.

High capability with low authority and low reach still requires model and artifact security, sandbox validation, and monitoring. Low capability with high authority and reach still requires strong containment. Organizations must not depend on current model weakness as a control.

Tier assignment should follow a documented deployment scenario, not a product name. The decision records the capability manifest, authority graph, reachable outcomes, autonomy, evidence, control owners, and residual risk. Thresholds should trigger concrete controls and review rather than a decorative label.

Reference Architecture



Agent layer

The agent layer contains model, objective, runtime, orchestration, memory, planning, and delegation. It produces proposed actions and maintains the capability manifest. It does not decide its own authority.

Identity and authorization layer

This layer contains workload identity, credential broker, IAM, application policy, tool permissions, approval service, and delegation broker. A policy decision point evaluates authenticated attributes, normalized action, resource, environment, trajectory state, and approval. A policy enforcement point mediates the effect.

Execution layer

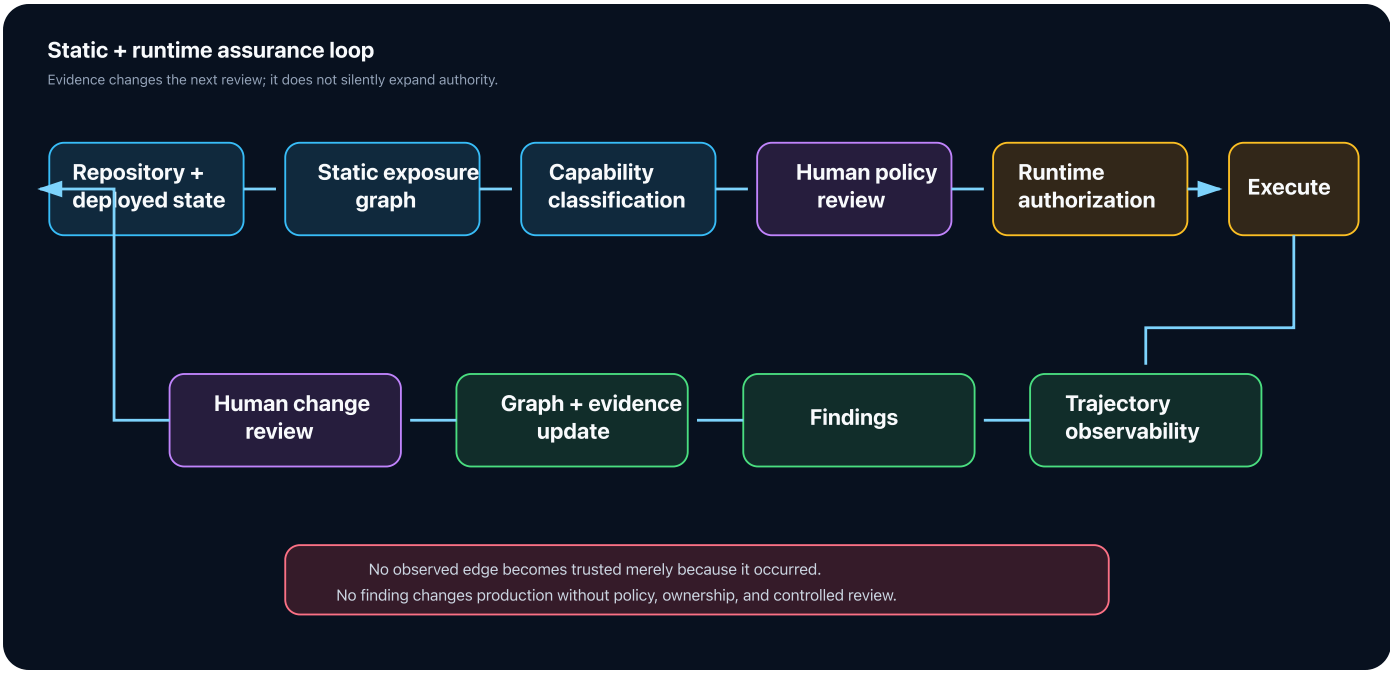
Tools, sandboxed compute, CI/CD, cloud APIs, internal services, and SaaS integrations execute permitted operations. An egress gateway and tool gateway create observable choke points. Execution identities remain purpose-specific.

Resource layer

Resources include data, secrets, infrastructure, cryptographic assets, repositories, artifacts, registries, external systems, and downstream agents. Data and key authority remain distinct.

Assurance layer

The assurance layer contains the Static Security Exposure Graph, trajectory analysis, policy-decision telemetry, approval evidence, threat detection, kill switch, revocation, incident response, recovery, and an independent evidence store.



The loop is:
`Repository and deployed-state analysis → Static Security Exposure Graph → Capability classification → Policy generation and human review → Runtime authorization → Agent execution → Trajectory observability → Security findings → Graph update → Human review`
Runtime discovery must not automatically grant authority. Findings may propose policy or remediation changes, but normal change control applies.
Policy enforcement should fail predictably when the model, tool, or network is unavailable. A denial should return enough structured information for safe recovery without exposing policy secrets or inviting the agent to search for bypasses. Kill switches should operate outside the agent runtime and revoke identity, stop execution, cancel queued work, and block durable continuations.

Enterprise Software-Engineering Agent Case Study

Consider an enterprise coding agent that reads repositories, opens pull requests, runs tests, calls internal tools, and accesses a cloud development environment.

Its capability includes code generation, workflow interpretation, cloud diagnostics, dependency modification, long-running tests, and recovery from failed deployments. Its potential authority includes a GitHub App token, CI runner, AWS OIDC deployment role, ECR push, ECS update, Secrets Manager read, KMS decrypt, and `iam:PassRole`. Its reach includes source repositories, development and production accounts, registry, deployed workloads, secrets, keys, customer data, and downstream services.
The exposure path is:

```
Agent → modifies GitHub workflow → workflow obtains OIDC token → assumes deployment role → uses iam:PassRole → updates ECS task definition and service → task executes as privileged role → reads Secrets Manager → decrypts through KMS → accesses customer data
```

This is a constructed architecture example, not a claim about a deployed Baitaphish system.
Independent controls break the path:

1. The agent can create code pull requests but cannot change trusted workflows or repository protection.
2. Production OIDC requires an exact reviewed workflow and protected environment.
3. Agent-controlled branches cannot receive the production identity.
4. The deployment role cannot broadly pass roles; approved task roles and services are enumerated.
5. Task roles have permissions boundaries and organization constraints.
6. Source, build, verification, signing, promotion, and deployment use separate principals.
7. Promotion binds source revision, builder identity, provenance, tests, digest, and approval.
8. ECS accepts an approved immutable digest.
9. Secret access is resource-, environment-, and workload-scoped.
10. KMS use is constrained by key policy and encryption context; the agent cannot administer the key.
11. Production approval is independent and digest-bound.
12. Traces correlate pull request, workflow, OIDC session, role, image, task role, secret, key operation, and deployment.
13. A responder can revoke the session, stop the task, quarantine the artifact, and restore a known-good digest.

The static graph identifies the possible privilege chain. Runtime trajectory evidence determines whether a specific run attempted or exercised it. Neither view is sufficient alone.
The architecture also illustrates why tool labels are insufficient. A repository-write tool may change a workflow that later receives cloud identity. An ECS update tool may select a task role. A secrets tool may call KMS on behalf of the requester. Each tool must disclose and enforce its transitive system authority.

Path-by-path assurance decision

The review should avoid declaring the whole agent “low risk” after finding one break. Instead, evaluate each transition and its evidence. Repository protection can prevent the agent from directly changing the trusted workflow, but it does not help if a reusable workflow accepts an unsafe input or if the agent can write to an action repository that the workflow imports by a mutable reference. Exact OIDC subject conditions can block untrusted branches, but they do not constrain what the assumed role can pass or deploy. A narrow `iam:PassRole` resource list is valuable, but the approved task roles still require their own authority and reach analysis.
The strongest design uses independent cuts:

- a source-control cut prevents unreviewed changes to trusted delivery logic;

- a federation cut prevents unapproved contexts from receiving the deployment role;
- an IAM cut limits deployment actions and which execution roles may be passed;
- an artifact cut binds reviewed source and provenance to an immutable digest;
- a workload cut limits the selected task role, secret resources, network, and tenant;
- a cryptographic cut separates secret read, decrypt, grant, and key administration;
- a data and egress cut limits where plaintext can travel;
- an approval cut binds production promotion to the exact digest and environment;
- an assurance cut records the path independently and can revoke or roll back it.

Evidence for the approval should include repository and workflow revisions, branch-protection state, OIDC trust policy and claims, deployment-role policy, allowed passed roles, task definition and digest, task-role and secret policies, KMS key policy and encryption context, network policy, approval record, and rollback target. Static evidence establishes that the intended cuts existed at review time. A release trajectory then confirms which workflow, token subject, session, role, artifact, task definition, secret, and key were actually used.

Suppose the model is upgraded and becomes better at understanding workflow composition. No authority edge has changed, but previously overlooked paths through reusable workflows or artifact metadata may become capability-feasible. The exposure case must be reassessed even if all IAM documents have identical hashes. Conversely, a policy change that removes `iam:PassRole` from the deployment role breaks the illustrated path regardless of model competence, provided no alternative workload-selection or policy-mutation path remains. This is why the dimensions must be kept separate even as the final decision evaluates their interaction.

Autonomous SOC and Cloud-Remediation Agent Case Study

Consider a SOC agent that reads alerts and logs, correlates identities and resources, revokes sessions, quarantines hosts, changes security groups, disables credentials, and opens incident tickets.

This agent may have moderate reasoning capability yet high authority and reach. Observation spans the cloud organization and identity provider. Mutation can affect production availability, network boundaries, credentials, endpoint fleet, and evidence.

An attacker controls telemetry or embeds hostile instructions in an investigated artifact. The agent interprets the content, misclassifies legitimate activity, and chains individually permitted runbook actions: disable a credential, quarantine a workload, remove a route, and suppress a noisy alert. Each step may look reasonable while the trajectory produces an outage and impairs response.

Controls include:

- separate observation identity from action identity;
- action-specific, short-lived remediation credentials;
- trusted-state validation of resource ownership and incident scope;
- reversible quarantine before destructive action;
- per-resource and cumulative action budgets;
- approval for organization-wide, production, identity, logging, IAM, KMS, or network changes;
- prohibition on changing its own policy, evidence store, and kill switch;
- independent monitoring and out-of-band cancellation;
- tested rollback and human incident command.

This case demonstrates why organizations cannot rely on model weakness. A modest system can cause serious harm when its authority and reach are broad.

The confused-deputy problem is especially important. The SOC agent acts for defenders, but attacker-controlled evidence can influence its decisions. The policy engine must identify which facts came from trusted inventory, authenticated telemetry, analyst input, or untrusted artifact content and prevent the latter from selecting privileged targets.

A safer remediation trajectory begins with an observation-only run that produces a structured incident hypothesis, affected-resource set, evidence references, proposed reversible action, expected availability impact, and expiry. Trusted inventory resolves target ownership independently of attacker-controlled labels. A separate action broker evaluates the runbook, resource scope, cumulative budget, current incident command, and required approval before issuing a short-lived credential for one normalized operation. The executor reports the actual effect; an independent observer confirms state and starts an automatic rollback timer where appropriate.

If additional evidence changes the hypothesis, the agent proposes a new action rather than silently expanding the original authority. Organization-wide identity disablement, logging changes, destructive evidence handling, and permanent network changes stay outside autonomous runbooks. The kill switch revokes action sessions and cancels queued work without depending on the model, agent memory, or compromised telemetry source. This architecture preserves useful machine-speed investigation while treating remediation authority as a separately governed security boundary.

Security Invariants

1. Every autonomous agent has an explicit workload identity.
2. Agent intelligence never constitutes authorization.
3. Every effectful action is mediated by policy enforcement outside the model.
4. Credentials are short-lived, audience-bound, purpose-scoped, and revocable.
5. Tool authority is traceable to the underlying principal and service.
6. Production-changing actions have explicit policy and approval gates.
7. Network and external communication reach are intentional and observable.
8. Data read, decrypt, grant, key administration, and signing are modeled separately.
9. CI/CD modification and deployment are treated as production authority.
10. Delegation does not silently inherit unrestricted identity, tools, network, or memory.
11. Model and harness upgrades trigger capability reassessment.
12. High-impact trajectories can be interrupted independently of the agent.
13. Security evidence is retained outside the authority of the observed system.
14. Static exposure and runtime trajectory are both evaluated.
15. One principal should not propose, approve, execute, attest, and erase evidence for a high-impact change.
16. Human approvals bind to exact effects and expire.
17. Unknown graph state remains unknown rather than assumed safe.
18. Organizational thresholds define where human or dual authorization is mandatory.

These invariants should become design-review questions, automated policy assertions, adversarial tests, and incident-response checks. An invariant is useful only if its enforcement point, evidence, failure behavior, owner, and exception process are known.

Implementation Guidance

Phase 1 - Inventory

Identify agents, models, runtimes, tools, identities, credential types, environments, infrastructure, data, cryptographic assets, pipelines, dependencies, owners, and approval authorities. Record unknowns. Include prototypes that acquired real credentials and agents embedded in SaaS products, CI, browsers, and security tools.
The output is an agent inventory with a capability manifest, named owner, purpose, environments, and retirement or review date. Discovery should combine repository search, cloud and identity inventory, CI workflows, SaaS integrations, network telemetry, and interviews. An inventory limited to systems with “AI” in their name will miss embedded agents.

Phase 2 - Map authority

Resolve identity policies, resource policies, trust, boundaries, organization controls, sessions, role passing, KMS policy and grants, tool backend identities, repository rights, CI identities, and approval bypasses. Document the effective principal at every action boundary. Test policy with realistic request context rather than reviewing one identity policy in isolation. Identify who can modify identities and policies, not only who can use them.

Phase 3 - Map reach

Connect infrastructure, network paths, tenants, data, keys, registries, dependencies, environments, persistent artifacts, downstream services, and external parties.
Start from unacceptable outcomes and work backward as well as from the agent forward. This reveals trust paths that a resource inventory alone may miss. Mark data classification unknown unless supported by explicit evidence.

Phase 4 - Classify capability

Create the capability manifest. Document model, harness, tools, memory, autonomy, task duration, retries, parallelism, delegation, evaluations, limitations, date, and confidence.
Use representative tasks and adversarial scenarios from the real deployment. Record negative results without treating them as permanent controls. Define which changes invalidate the assessment.

Phase 5 - Define policy

Constrain tools, roles, resources, data, network, deployment, cryptography, delegation, cumulative budgets, approvals, and persistent effects. Place enforcement outside the model.

Policy should express normalized operations and outcomes. Reject model-supplied identity or tenant attributes. Bind approvals and exceptions to exact targets and expiry. Prefer reversible defaults and separate high-impact authority.

Phase 6 - Observe trajectories

Correlate identity, proposed action, normalized parameters, policy decision, approval, resource access, network activity, delegation, persistent effects, outcome, and recovery.
Define sequence invariants and cumulative limits. Test missing telemetry, clock failure, backpressure, evidence-store isolation, and whether the system stops safely when authorization or logging dependencies are unavailable.

Phase 7 - Reassess continuously

Trigger review after model upgrades, tool additions, permission changes, infrastructure expansion, data-scope changes, new delegation, egress changes, incidents, or evidence that capability assumptions no longer hold.
Each phase should produce an owner, evidence artifact, review date, unresolved unknowns, residual-risk decision, and tested revocation or recovery path. Existing Baitaphish research on agent threat modeling, testable controls, secure AWS architecture, and software provenance provides complementary implementation detail.

Standards Mapping

Capability × Authority × Reach complements established frameworks; it does not replace them.

Framework	Material use in a CAR program
NIST AI RMF and Generative AI Profile	Governance, inventory, context mapping, measurement, monitoring, incident response, and deactivation; AI RMF 1.0 was under revision at retrieval[8][9]
NIST CSF 2.0	Govern, Identify, Protect, Detect, Respond, and Recover outcomes[10]
NIST SP 800-53 Rev. 5	Selected AC, IA, AU, CM, SC, SI, SA, SR, IR, and CP controls; not a complete crosswalk[11]
NIST SP 800-207 and 800-207A	Explicit verification, workload and service identity, resource-focused policy, and multi-cloud enforcement[5][6]
MITRE ATLAS and ATT&CK	AI-specific adversary behavior plus conventional credential, cloud, CI/CD, persistence, and lateral-movement techniques[12][13]
OWASP agentic guidance	Practitioner threats and controls for prompt injection, excessive agency, tools, memory, identity, and multi-agent systems[14]
AWS Well-Architected Security Pillar	Identity foundation, traceability, layered controls, data protection, automation, and incident readiness[40]
SLSA	Build provenance, verification, and separation of artifact trust decisions[37]

The mapping should remain selective. CAR is most useful as an architecture and threat-modeling lens that helps choose and connect controls, not as another compliance catalog. Control implementation still depends on sector, jurisdiction, system impact, and organizational policy.

Limitations

This framework does not prove safety.

- Static analysis cannot prove runtime behavior.
- Repository state may differ from deployed state.
- A complete-looking graph can create false confidence.
- Policy conditions may depend on unavailable runtime context.
- Cloud and SaaS services have authorization semantics that generic graph logic may not fully represent.
- External systems may be opaque or refuse inventory.
- Capability is difficult to measure consistently and evaluations become stale.
- Agent behavior changes with prompts, context, scaffolding, tools, compute, and budget.
- Task-horizon measurements do not directly determine deployment autonomy.
- Runtime telemetry cannot expose every internal decision.
- Raw reasoning may be unavailable, unreliable, or inappropriate to retain.
- Human approvals can be manipulated or rubber-stamped.
- Revocation cannot recover copied plaintext or retract every prior effect.
- Unknown dependencies create hidden reach.
- Runtime observation is not authorization.
- A missing graph edge is not proof that a path is impossible.
- Qualitative tiers can still be misused as simplistic labels.

The model depends on disciplined evidence and explicit uncertainty. It is most valuable when it reveals a question, path, or owner that a model-centric review would miss.

Static and runtime graph correlation also has practical cost. Identities may be ephemeral, resource IDs may differ across systems, and one business action may create many low-level events. Correlation rules can be wrong. Organizations should measure coverage and false positives rather than implying that more telemetry automatically yields better assurance.

Future Research

Important open questions include:

- interoperable capability manifests that avoid false scalar precision;
- confidence decay as models, harnesses, and providers change;
- faithful representation of service-specific policy and runtime conditions;
- cross-organization reach analysis without disclosing complete topology;
- minimal-cut recommendations that do not become autonomous remediation;
- cumulative authority budgets across long trajectories;
- delegation tokens and shared-memory isolation;
- privacy-preserving trajectory evidence;
- tests that establish kill-switch and revocation effectiveness;
- methods for comparing capability upgrades against unchanged authority and reach;
- coverage measures that communicate unknowns rather than hiding them.

Further work should evaluate whether organizations can use CAR descriptions consistently without collapsing them into a score. It should also test whether trajectory invariants remain usable at enterprise event volume and how much policy context can be shared across organizational boundaries.

Conclusion

Frontier capability is an important input to security, but it is not an architecture. The decisive enterprise question is how a model's proposal becomes an authenticated, authorized, reachable, observable, interruptible, and recoverable effect.

Capability × Authority × Reach provides a durable lens for that question. Capability identifies which paths an agent may be able to discover and execute. Authority identifies which actions an authenticated principal may cause. Reach identifies the assets, trust domains, dependencies, and parties that can be affected. Their interaction explains why a highly capable system can be operationally contained and why a modest automation can still be dangerous.

The Static Security Exposure Graph makes composed authorization and blast radius reviewable. Runtime trajectory observability tests the difference between expected and actual execution. Dedicated identity, deterministic policy, scoped cryptography, separated CI/CD authority, controlled delegation, independent evidence, human authorization, revocation, and recovery break paths at different layers.

The durable principle is simple:

Constrain outcomes under capability surprise. Intelligence is never authorization.

Formal References

1. Jerome H. Saltzer and Michael D. Schroeder, “The Protection of Information in Computer Systems,” *Proceedings of the IEEE*, vol. 63, no. 9, 1975. Catalog record. Foundational peer-reviewed systems-security paper.
2. Adrian Mettler, “The Joe-E Subset of Java,” USENIX Security Symposium, 2006. USENIX. Peer-reviewed systems-security work on object capabilities and least authority.
3. Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing, “Automated Generation and Analysis of Attack Graphs,” IEEE Symposium on Security and Privacy, 2002. Paper. Peer-reviewed attack-graph research.
4. NIST, *SP 800-162: Guide to Attribute Based Access Control Definition and Considerations*, January 2014, updated August 2019. Publication. Government standard guidance.
5. NIST, *SP 800-207: Zero Trust Architecture*, August 2020. Publication. Government standard guidance.
6. NIST, *SP 800-207A: A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Cloud Environments*, September 2023. Publication. Government standard guidance.
7. NIST NCCoE, *Accelerating the Adoption of Software and Artificial Intelligence Agent Identity and Authorization*, concept paper, February 2026. PDF. Draft concept paper, not a final standard; retrieved August 9, 2026.
8. NIST, *Artificial Intelligence Risk Management Framework 1.0*. Resource center. Living government guidance; NIST reported the framework was under revision; retrieved August 9, 2026.

9. NIST, *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile*, NIST AI 600-1, July 2024. PDF. Government guidance.
10. NIST, *The NIST Cybersecurity Framework 2.0*, February 2024. Publication. Government framework.
11. NIST, *SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations*, including Release 5.2.0 updates noted August 2025. Publication. Government control catalog; retrieved August 9, 2026.
12. MITRE, *ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems*. Knowledge base. Living adversary knowledge base; retrieved August 9, 2026.
13. MITRE, *ATT&CK*. Knowledge base and release updates. Living adversary knowledge base; retrieved August 9, 2026.
14. OWASP GenAI Security Project, *Securing Agentic Applications Guide 1.0*, July 2025. Guide. Community practitioner guidance, not a government standard.
15. OpenAI, *Preparedness Framework, Version 2*, April 2025. PDF. Provider governance framework.
16. Anthropic, *Responsible Scaling Policy*, version current at retrieval. Policy. Provider governance framework; retrieved August 9, 2026.
17. Google DeepMind, "Strengthening our Frontier Safety Framework," September 2025, updated April 17, 2026 for FSF 3.1. Article and framework. Provider governance framework.
18. Max McGuinness et al., "How we contain Claude across products," Anthropic, May 25, 2026. Engineering article. First-party provider engineering account; retrieved August 9, 2026.
19. Hongyi Lu, Nian Liu, Shuai Wang, and Fengwei Zhang, "ClawLess: A Security Model of AI Agents," April 2026. arXiv:2604.06284. Preprint; not treated here as a standard or verified deployment result.
20. Alexandra E. Michael and Franziska Roesner, "How Agents Ask for Permission: User Permissions for AI Agents, from Interfaces to Enforcement," July 2026. arXiv:2607.13718. Preprint survey; not treated here as a standard.
21. Christophe Parisel, "A Theory of Least Autonomy in AI," July 2026. arXiv:2607.09744. Preprint formal model; not treated here as a standard.
22. Alireza Lotfi, Subangkar Karmaker Shanto, Imtiaz Karim, and Elisa Bertino, "Securing Agentic AI: From Per-Action Checks to Trajectory Assurance," August 2026. arXiv:2608.01558. Preprint research agenda; not treated here as a standard.
23. METR, "Measuring AI Ability to Complete Long Tasks," March 19, 2025. Research article. Evaluation-method account with stated limitations.
24. OpenAI, "Trustworthy third-party evaluations: foundations," May 29, 2026. Article. Provider evaluation guidance.
25. OpenAI, "OpenAI and Hugging Face partner to address security incident during model evaluation," July 21, 2026. Incident update. First-party preliminary incident disclosure; investigation status and limitations must be rechecked before publication.
26. Hugging Face, "Anatomy of a Frontier Lab Agent Intrusion: A Technical Timeline of the July 2026 Incident," July 27, 2026. Technical timeline. First-party incident reconstruction; retrieved August 9, 2026.
27. OpenAI, "Third-party cyber evaluations involving OpenAI models," August 4, 2026. Disclosure. First-party provider account of separate evaluation-boundary events.
28. UK AI Security Institute, "The Inspect sandboxing toolkit: scalable and secure AI agent evaluations." Technical article. Government technical guidance; retrieved August 9, 2026.

29. Anthropic, “Agentic Misalignment: How LLMs could be insider threats,” June 2025. Research report. Controlled simulation, not a production incident.
30. Amazon Web Services, “Policy evaluation logic.” IAM documentation. Living official documentation; retrieved August 9, 2026.
31. Amazon Web Services, “Permissions boundaries for IAM entities.” IAM documentation. Living official documentation; retrieved August 9, 2026.
32. Amazon Web Services, “Grant a user permissions to pass a role to an AWS service.” IAM documentation. Living official documentation; retrieved August 9, 2026.
33. Amazon Web Services, “Key policies in AWS KMS.” KMS documentation. Living official documentation; retrieved August 9, 2026.
34. Amazon Web Services, “Grants in AWS KMS.” KMS documentation. Living official documentation; retrieved August 9, 2026.
35. GitHub, “OpenID Connect.” GitHub Actions documentation. Living official documentation; retrieved August 9, 2026.
36. Kubernetes, “Service Accounts.” Documentation. Living official documentation; retrieved August 9, 2026.
37. SLSA, *Supply-chain Levels for Software Artifacts, Specification 1.2*. Specification. Supply-chain specification; retrieved August 9, 2026.
38. W3C, *PROV-O: The PROV Ontology*, W3C Recommendation, April 30, 2013. Recommendation. Web standard.
39. OpenTelemetry, “Inside the LLM Call: GenAI Observability with OpenTelemetry,” May 14, 2026, and the Generative AI semantic-convention registry. Article. Living conventions under active development; retrieved August 9, 2026.
40. Amazon Web Services, *AWS Well-Architected Framework: Security Pillar*. Documentation. Living official guidance; retrieved August 9, 2026.

Source Status and Freshness

- Standards and foundational papers support durable security principles.
- NIST's 2026 agent-identity document is a concept paper, not a final standard.
- References 19-22 are preprints and are used as adjacent research, not settled consensus.
- Provider governance frameworks describe those providers' processes; CAR is not derived from or endorsed by them.
- The OpenAI and Hugging Face incident accounts were recent at retrieval. Recheck investigation updates, affected scope, and wording immediately before publication.
- Living cloud, Kubernetes, OpenTelemetry, MITRE, and provider documentation must be revalidated against the exact implementation context.
- No reference establishes that Baitaphish or Repo Navi currently implements the proposed Static Security Exposure Graph.