Deterministic Segregation-of-Duties Evaluation for Microsoft Copilot Studio Agents

Arin Mallanna Tumbagi

Abstract

AI agents accumulate effective access from configured capabilities, human principals, delegated agents, and shared identities, so a user-only segregation-of-duties model is insufficient for their evaluation. We specify five risk contexts and a deterministic three-pass engine that emits evidence-bearing violations with lifecycle state. We also specify how such engines should be evaluated. To our knowledge, no prior model covers deterministic SoD conflicts across Copilot Studio agent capabilities, owner and invoker identity-governance (IGA) entitlements, invocation chains, and shared-identity cross-agent groups in a unified lifecycle engine.

1 Introduction

Segregation of duties (SoD) constrains combinations of authority that permit one principal to complete a sensitive process without an independent control. Role-based access control formalizes static and dynamic SoD over users, roles, permissions, and sessions [1, 2]. An agent platform adds several authority sources that are not represented by a user's assigned roles alone. A Copilot Studio agent can possess tools, knowledge sources, connections, and credentials; invoke another agent; execute with end-user credentials or maker-provided credentials; and participate in a shared environment governed through connector and data policies [3, 4, 5, 6].

This report defines effective access for this setting and documents a prototype evaluator. The engine converts agent configuration and IGA entitlement data into normalized capability evidence. It evaluates that evidence under five contexts: agent alone, agent with owner, invocation chain, cross-agent group, and agent with invoker [7, 8]. A result is not merely a Boolean value. It identifies the matched rule, functions, contributing records, scope, severity, and lifecycle state [7, 9, 10].

The contribution is deliberately bounded. The system detects violations after the fact; it is not a runtime authorization mechanism. The public repository uses static agent snapshots and synthetic owner entitlements, and it lacks the identity integrations needed for complete invoker evaluation [11, 12]. The report consequently separates the intended model from the behavior established by the inspected implementation.

2 Problem

Let A be a set of agents and P a set of human principals. Each agent $a \in A$ has a normalized capability set $C(a)$. A capability record may originate from a tool, knowledge source, connection, credential, or invocation edge. Each principal $p \in P$ has an entitlement set $H(p)$ obtained from an identity-governance source. The owner mapping is $o : A \to P$. An invocation graph is $G = (A, D)$, where $(a, b) \in D$ means that a declares an invocation of b. A grouping function $\gamma_k : A \to V_k$ maps an agent to the value of attribute k, such as connection, owner, or credential [7, 8].

A conventional user-SoD check evaluates combinations within $H(p)$. That check does not detect a conflict whose evidence is divided between $H(o(a))$ and $C(a)$, between $C(a)$ and $C(b)$ for a delegated child, or across agents that act through the same connection. The relevant principal is therefore an evaluation context rather than a single directory object.

Consider a finance agent that reads a shared drive and sends email. Its own configuration can satisfy a rule for reading sensitive content and transmitting it externally. If its owner also holds an accounts-payable approval entitlement while the agent initiates payments, the owner–agent union can satisfy a separate initiate-and-approve rule. The public project post uses this case to distinguish an agent-only path from an owner-composite path [10]. The evaluator records the exact grants and capabilities that support each result, which permits review without asking a language model to decide whether the rule matched [10, 9].

The model makes two exclusions. First, a matched result is evidence of a prohibited capability combination, not evidence that the corresponding transaction occurred. Second, the prototype does not issue a runtime deny decision. Prevention, remediation authorization, and runtime interception remain outside its demonstrated boundary [7, 9].

3 Model

3.1 Effective-access contexts

For a focal agent a, owner $o(a)$, invoker u, graph G, and grouping attribute k, define

$$E_A(a) = C(a), \tag{1}$$

$$E_O(a) = C(a) \cup H(o(a)), \tag{2}$$

$$E_I(a, u) = C(a) \cup H(u), \tag{3}$$

$$E_C(a, G) = \bigcup_{x \in R_G(a)} C(x), \tag{4}$$

$$E_X(a, k) = \bigcup_{x : \gamma_k(x) = \gamma_k(a)} C(x). \tag{5}$$

$R_G(a)$ is the intended set of agents reachable from a, including a. The current code substitutes the parent and its directly declared children for $R_G(a)$; it does not compute a transitive closure [8, 12]. Effective access is represented as evidence-bearing records rather than as an untyped set of action strings. A record retains its originating agent or principal and the field that matched the rule [7, 8].

3.2 Rule semantics

A rule r contains an ordered set of functions F_r. Each function f contains an ordered sequence of conditions $(q_1, \ldots, q_m)$. A condition selects a normalized field and applies one of 'EQUALS', 'NOT_EQUALS', 'CONTAINS', 'NOT_CONTAINS', 'IN', 'NOT_IN', 'REGEX', or 'EXISTS'. The implementation combines condition results from left to right using the preceding condition's 'nextOperator'. It does not apply an independent operator-precedence grammar. An empty function is false [13].

Let $Q(q, E)$ be the Boolean result of condition q over context E, and let $\diamond_i$ be the configured connector following q_i. The function value is

$$M(f, E) = (((Q(q_1, E) \diamond_1 Q(q_2, E)) \diamond_2 Q(q_3, E)) \cdots). \tag{6}$$

The rule value before scope checking is

$$B(r, E) = \bigwedge_{f \in F_r} M(f, E). \tag{7}$$

The evaluator also collects scope sets from matching evidence. If at least two functions provide nonempty scope sets, their intersection must be nonempty. If fewer than two scope sets are available, the implementation accepts the overlap check because overlap cannot be determined [8, 13]. Thus

$$\text{Match}(r, E) = B(r, E) \wedge \text{ScopeOK}(r, E). \tag{8}$$

Operation matching uses normalized action prefixes and aliases from a cached taxonomy. This admits connector-specific operation names but does not establish complete semantic equivalence between all connector actions [8, 12].

3.3 Risk-type specification

`AGENT_ONLY`. The evaluator selects $E_A(a)$. All supporting evidence must come from the focal agent. For example, a SharePoint knowledge source supplies a sensitive-read capability and an HTTP tool supplies an external-send capability without confirmation. A rule requiring both capabilities matches the single agent [7].

`OWNER_COMPOSITE`. The evaluator selects $E_O(a)$. Evidence can be partitioned between the agent and its owner. For example, $H(o(a))$ contains an SAP accounts-payable approval entitlement while $C(a)$ contains payment initiation. The union matches an initiate-and-approve rule even though neither side matches it alone [7, 8].

`CHAIN`. The intended evaluator selects $E_C(a, G)$. A parent that reads invoice records and a child that approves payments can jointly match a read-and-approve rule. The implementation creates a virtual agent by concatenating the parent capabilities with each directly named child's capabilities, evaluates the union, and attaches the result to the parent. The example is therefore supported for one invocation edge; deeper reachability is not implemented [7, 8].

`CROSS_AGENT`. The evaluator selects $E_X(a, k)$. It groups agents by the rule's configured attribute, constructs one virtual union per group, and associates a match with each member. For example, two agents with the same connection identifier can divide finance-record read and payment approval capabilities across their configurations. A 'SHARED_CONNECTION' rule observes both records in the group union. Shared connection, owner, and credential grouping are implemented; shared-invoker grouping returns no members [7, 8].

`INVOKER_COMPOSITE`. The intended evaluator selects $E_I(a, u)$. For example, $H(u)$ can contain purchase-order creation while the invoked agent can approve purchase orders on behalf of the user. The public code passes an empty invoker-entitlement set, so this example specifies the intended context rather than a demonstrated end-to-end path. Invoker identity and entitlement acquisition are recorded as dependencies on external directory and HR-system integration [8, 11, 12].

4 Architecture

The prototype uses Spring Boot 3.5 and Java 21, is built with Gradle 8.5, stores data in MySQL 8, and applies schema migrations through Flyway. It uses direct JDBC rather than an object-relational mapper. The package structure separates boundary controllers, control services, persistence data-access objects, entity records, and utilities [7, 9].

An evaluation request causes the service to load the job, rules, agent snapshots, entitlement data, and action-taxonomy mappings. It then performs three ordered passes [8, 9]:

1. The per-agent pass resolves owner entitlements and evaluates three rule families: 'AGENT_ONLY', 'OWNER_COMPOSITE', and 'INVOKER_COMPOSITE'. The last family currently receives no invoker entitlements.
2. The cross-agent pass groups the loaded agents for each 'CROSS_AGENT' rule. Groups with at least two members are converted to virtual agents by concatenating tools, knowledge sources, connections, credentials, and edges.
3. The chain pass converts each parent with declared edges and its loaded direct children into a virtual agent and evaluates 'CHAIN' rules.

For each match, the service computes a deterministic fingerprint and persists a violation plus supporting evidence. The documented lifecycle is 'OPEN' on first detection, confirmation when present again, 'CLOSED' when absent from a subsequent run, and 'REOPENED' when a closed fingerprint recurs. A separate history store appends history events [7, 9, 8]. These transitions are defined for sequential runs against a fixed snapshot: each evaluation reads one self-consistent database state, and the job is not re-run mid-evaluation.

Behavior under concurrent execution is outside the model and intentionally unspecified.

The prototype consumes static Copilot Studio export JSON for two agents and synthetic entitlement data for four owners. The dependency register lists live agent inventory, Microsoft Graph group membership, invoker identity and entitlements, connector-operation metadata, Power Platform administration data, and complete chain discovery as unavailable or external inputs [11, 12]. These inputs define the boundary between the evaluation kernel and production data acquisition.

5 Evaluation

This report presents no latency measurements, and that omission is deliberate. The available observations were recorded without a published hardware configuration, warm-up policy, repetition count, or dataset, so reporting them as results would imply a rigor the record does not support. What follows instead is how an engine of this kind should be evaluated, so that future measurements are comparable.

A reproducible evaluation publishes the input snapshots, the rule corpus, the software and hardware configuration, the warm-up procedure, the repetition count, and the latency distribution, reported separately for the per-agent, cross-agent, and chain passes. Correctness evaluation uses fixtures: positive and negative cases for every condition operator, mixed Boolean expressions, scope intersection and non-intersection, group formation at boundary sizes, direct chains, transitive chains, cycles, missing nodes, lifecycle transitions across runs, and absent identity data. A labeled corpus additionally measures false positives and false negatives; without one, detection quality is unknown rather than good.

6 Related Work

RBAC models define users, roles, permissions, sessions, and constraints. Static and dynamic separation of duty are established constraints in that literature [1]. Simon and Zurko analyze how role-based environments distribute permissions so that a sensitive operation requires multiple subjects [2]. The older user-SoD implementation used here as an engineering comparator likewise evaluates human entitlement assignments against risk definitions [14]. These sources establish prior work for SoD over human identity and authorization structures.

Copilot Studio documents agent identities, tools, knowledge sources, connectors, child agents, and credential modes. Tools may execute with end-user credentials or maker-provided credentials; child agents retain their own instructions, tools, knowledge, and orchestration [3, 4]. Microsoft also documents environment governance, data policies, connector controls, sharing controls, and automated security scans. The scan identifies configuration conditions that include missing authentication, maker-provided credentials, and broad sharing [5, 6, 15]. These mechanisms govern agent configuration and authentication. They are not described as a business-SoD evaluation across all five contexts specified here.

The NIST Generative AI Profile provides cross-sector risk-management guidance for generative-AI systems [16]. AgentSpec defines declarative runtime constraints and an enforcement architecture for LLM agents [17]. These works establish prior approaches for organizational AI risk management and generic runtime agent policy. The present scope is narrower in platform target and broader in identity composition: it combines static agent configuration, IGA entitlements, delegation, shared identities, deterministic evidence, and violation lifecycle state. The absence statement in the abstract applies only to that complete conjunction and only to the bounded sources reviewed here.

No individual constituent is claimed as new. Human SoD, role constraints, agent authentication, connector governance, configuration scanning, invocation, declarative policy, and lifecycle tracking all have antecedents. The research question is whether those constituents form a sound and complete effective-access model for business SoD when an agent can act through several identity and delegation contexts.

7 Limitations and Future Work

Model completeness. The five risk types have no completeness proof. A formal capability-and-scope lattice is required, followed by either a proof that every relevant authority composition maps to one of the five contexts or counterexamples that extend the taxonomy [10].

Graph semantics. The documented chain concept implies reachability, while the implementation unions only direct children. Future work must define transitive closure, cycle handling, duplicate paths, missing nodes, and evaluation over partially loaded graphs. A chain-related rule condition in the current implementation also checks for a chain-related name and a nonempty edge list without comparing parent and child owner identifiers [8, 13, 12].

Identity integration and drift. Invoker-composite evaluation and shared-invoker grouping are incomplete. Live retrieval must define nested group expansion, application roles, delegated permissions, identity freshness, failure behavior, and reconciliation with HR-system records. Static JSON and synthetic owner entitlements also leave configuration and entitlement drift unmeasured [11, 12].

Scope and taxonomy. Scope parsing and operation classification are pattern-based. The repository records possible scope false negatives, while the evaluator accepts scope overlap when fewer than two functions provide scope sets. A specification must choose fail-open or fail-closed behavior for missing and malformed scope data and version the connector-operation taxonomy [13, 12].

Lifecycle correctness. Lifecycle transitions are specified for sequential snapshot runs; concurrent-run semantics, uniqueness constraints, and history durability remain open and require a MySQL integration test. Database constraints or transactional invariants should define uniqueness, legal state transitions, history durability, and concurrent-run behavior. The current record does not verify these properties [8, 9].

Empirical evaluation. A future study should report latency distributions, scaling by agents, rules, functions, group sizes, and graph density, separating data-loading cost from evaluation and persistence. A labeled corpus is required to measure false positives and false negatives. This report specifies that protocol; it presents no measurements.

Literature coverage and enforcement. The related-work review is bounded. A systematic search is required before publication. Runtime enforcement, remediation authorization, and conflicts between business policy and platform policy are also outside the demonstrated implementation and require separate designs [7, 9].

References

[1] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. "Role-Based Access Control Models". In: *Computer* 29.2 (1996), pp. 38–47. DOI: 10.1109/2.485845. URL: https://csrc.nist.gov/csrc/media/projects/role-based-access-control/documents/sandhu96.pdf.

[2] Richard T. Simon and Mary Ellen Zurko. "Separation of Duty in Role-Based Environments". In: *Proceedings of the 10th Computer Security Foundations Workshop*. 1997. DOI: 10.1109/CSFW.1997.596811. URL: https://doi.org/10.1109/CSFW.1997.596811.

[3] Microsoft. *Add Tools to Custom Agents*. Microsoft Learn. Accessed 2026-09-05. URL: https://learn.microsoft.com/en-us/microsoft-copilot-studio/add-tools-custom-agent.

[4] Microsoft. *Configure User Authentication for Tools*. Microsoft Learn. Accessed 2026-09-05. URL: https://learn.microsoft.com/en-us/microsoft-copilot-studio/configure-enduser-authentication.

[5] Microsoft. *Security and Governance in Microsoft Copilot Studio*. Microsoft Learn. Accessed 2026-09-05. URL: https://learn.microsoft.com/en-us/microsoft-copilot-studio/security-and-governance.

[6] Microsoft. *Configure Data Policies for Agents*. Microsoft Learn. Accessed 2026-09-05. URL: https://learn.microsoft.com/en-ca/microsoft-copilot-studio/admin-data-loss-prevention.

[7] Arin Mallanna. *Agent Risk and Segregation-of-Duties Evaluation*. GitHub repository, commit 3e6615b13b6367d599ef058b8c0ff768ac7920c5. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/agent-risk-SOD-Evaluation/blob/3e6615b13b6367d599ef058b8c0ff768ac7920c5/README.md.

[8] Arin Mallanna. *EvaluationService.java*. Source code, commit 3e6615b13b6367d599ef058b8c0ff768ac7920c5. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/agent-risk-SOD-Evaluation/blob/3e6615b13b6367d599ef058b8c0ff768ac7920c5/src/main/java/com/saviynt/detectiveagent/control/EvaluationService.java.

[9] Arin Mallanna. *Agent SoD System Design*. Design document, commit 3e6615b13b6367d599ef058b8c0ff768ac7920c5. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/agent-risk-SOD-Evaluation/blob/3e6615b13b6367d599ef058b8c0ff768ac7920c5/docs/SYSTEM_DESIGN.md.

[10] Arin Mallanna. *Segregation of Duties for AI Agents*. Public technical post. Source at commit d8f3d69e662e835278fb458de2ccf5f0e9e847ee; accessed 2026-09-05. 2026. URL: https://github.com/Arin016/Arin016.github.io/blob/d8f3d69e662e835278fb458de2ccf5f0e9e847ee/src/content/blog/sod-for-agents.md.

[11] Arin Mallanna. *Agent SoD Data Dependencies*. Project record, commit 3e6615b13b6367d599ef058b8c0ff768ac7920c5. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/agent-risk-SOD-Evaluation/blob/3e6615b13b6367d599ef058b8c0ff768ac7920c5/DATA_DEPENDENCIES.md.

[12] Arin Mallanna. *Agent SoD Version-One Tradeoffs*. Project record, commit 3e6615b13b6367d599ef058b8c0ff768ac7920c5. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/agent-risk-SOD-Evaluation/blob/3e6615b13b6367d599ef058b8c0ff768ac7920c5/TRADEOFFS_V1.md.

[13] Arin Mallanna. *ConditionEvaluator.java*. Source code, commit 3e6615b13b6367d599ef058b8c0ff768ac7920c5. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/agent-risk-SOD-Evaluation/blob/3e6615b13b6367d599ef058b8c0ff768ac7920c5/src/main/java/com/saviynt/detectiveagent/control/ConditionEvaluator.java.

[14] Arin Mallanna. *Risk and Segregation-of-Duties Evaluation*. GitHub repository, commit 2647272de05b2d8140497a64aa9465df20dad036. Accessed 2026-09-05. 2026. URL: https://github.com/Arin016/risk-SOD-Evaluation/blob/2647272de05b2d8140497a64aa9465df20dad036/README.md.

[15] Microsoft. *Automatic Security Scan for Agents*. Microsoft Learn. Accessed 2026-09-05. URL: https://learn.microsoft.com/en-us/microsoft-copilot-studio/security-scan.

[16] National Institute of Standards and Technology. *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile*. Tech. rep. NIST AI 600-1. National Institute of Standards and Technology, 2024. DOI: 10.6028/NIST.AI.600-1. URL: https://doi.org/10.6028/NIST.AI.600-1.

[17] Zehan Wang, Christopher M. Poskitt, and Jun Sun. *AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents*. 2025. arXiv: 2503.18666 [cs.SE]. URL: https://arxiv.org/abs/2503.18666.