



(RESEARCH ARTICLE)



SECURING THE AI-FIRST SDLC: IDENTITY, ACCESS, AND DATA PROTECTION FOR ENTERPRISE AI AGENTS

Ambar Nath Saha ¹ and Debashis Patra ^{2,*}

¹ Independent Researcher, Halifax, Nova Scotia, Canada.

² Computer Science, UT Austin, TX, USA.

* Corresponding Author: Email: debashis31@gmail.com

ORCID Details

Ambar Saha: <https://orcid.org/0009-0006-4457-4753>

Debashis Patra: <https://orcid.org/0009-0004-7894-1852>

World Journal of Advanced Research and Reviews, 2026, 32(01), 1368–1381

Publication history: Received on 08 September 2026; revised on 18 October 2026; accepted on 20 October 2026

Article DOI: <https://doi.org/10.30574/wjarr.2026.32.1.2651>

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

AI agents can now participate in several steps of software delivery: reading requirements, changing code, requesting tests, and preparing releases. Giving them this freedom creates a difficult security question. How can an enterprise allow useful work without giving an agent more authority than the task requires? Existing practices such as least privilege, identity management, separation of duties, and audit logging provide much of the foundation. The challenge is applying them consistently when a human delegates work to agents that use several connected systems. This paper proposes Agentic Identity and Access Governance (AIAG), a security framework for the AI-First Software Development Lifecycle (SDLC). Four connected mechanisms define the proposal: a Role Conflict Matrix that checks the combined permissions of people and their sponsored agents; an Agent Access Manifest that records permitted tools and operating limits; a governed boundary between live personal-information access and general AI-assisted work; and an MCP Gateway that enforces access policy before tool calls execute. A customer notification-preference change is used to explain how these mechanisms affect development, testing, release approval, and support. We also describe credential handling, data-egress controls, audit, and practical ways to evaluate the framework. The work is conceptual, informed by technical analysis and exploratory proof-of-concept activity. No end-to-end deployment or measured security results are claimed.

Keywords: Agentic AI; AI-First SDLC; Identity and Access Management; Least Privilege; Separation of Duties; Model Context Protocol; Data Protection; Data Egress

1 INTRODUCTION

In our earlier work, we described an AI-First Software Development Lifecycle (SDLC) in which specialist agents help with requirements, coding, testing, and deployment while people retain responsibility for consequential decisions [1]. The companion Product and Designer Layer paper explored how agents can turn an initial business request into structured requirements and design artifacts [2]. Those papers mainly asked how the work could be organized. Here we ask what must be in place before an enterprise can safely let agents act on its systems.

Consider a familiar change: a customer wants another option for receiving notifications. A requirements agent reads the ticket, a coding agent changes the relevant service, and a testing agent triggers a Jenkins integration job. Each step sounds routine. Yet behind every step is a permission: reading a ticket, writing a branch, running code on a build worker, or retrieving data for a test. If those permissions are bundled into one broad service account, a mistake in an early step can affect systems well beyond the change. The issue is not that the agent is necessarily malicious. It is that the agent may choose its next action from incomplete, misleading, or attacker-controlled context. Agent frameworks demonstrate how powerful tool use can be [4–6]; MCP provides a way to expose tools [12]. Neither feature establishes which actions a particular agent ought to be trusted with.

This concern grows when a pilot becomes a shared enterprise platform. Many organizations already have inherited privileges, long-lived service tokens, powerful CI servers, and customer data spread across operational systems. Adding agents creates more combinations of identities and connectors. A testing agent may only need to run one staging job, but a connector configured with administrator access could let it edit pipelines or reach production credentials. A model can also misinterpret a log or follow an instruction hidden in a document. The security design therefore needs to account for ordinary mistakes, compromised content, and excessive permissions rather than assume that the model will always make a safe choice.

We call the proposed approach *Agentic Identity and Access Governance (AIAG)*. Its underlying controls are not new: access catalogs, least privilege, credential expiry, separation of duties, and reliable audit have long been used in enterprise systems [3, 11]. Our contribution is to bring them together around the particular problems of agent sponsorship, delegated authority, tool-level decisions, and data movement. Throughout the paper we return to the notification-preference change. It provides a simple way to show why a legitimate task should not automatically provide access to customer email addresses, phone numbers, production deployment tools, or unrelated systems.

Four mechanisms carry the argument. The human-agent Role Conflict Matrix (Section VII-B) examines the effective permissions of a person together with every agent acting under that person's sponsorship. The Agent Access Manifest (Section VIII-A) sets out which connectors, tools, parameters, data classes, and destinations an agent may use. PI/AI separation (Section VII-C) prevents an active session handling live production personal information (PI) from using a general-purpose AI assistant by default, while allowing reviewed in-zone alternatives. Finally, the MCP Gateway and Policy Engine (Section V) enforce these decisions at the moment a tool is called. Credential controls, restricted network routes, audit records, and testing support these four mechanisms. In other words, the framework follows access through its whole life: who requests it, what is approved, what is actually used, and how it can be stopped.

Status and scope. This is a proposed reference architecture, developed through engineering analysis of identity, DevOps, and agent-based development practices and informed by the authors' exploratory proof-of-concept work. We have not implemented the complete design, compared it with a conventional security model, or measured its effect on incidents or productivity. The tools named in the paper are illustrative technologies, not products evaluated in this study. Any benefit attributed to the design is therefore a proposition for later testing.

2 BACKGROUND AND RELATED WORK

2.1 Agentic Development Systems and Tool Access

Research on agent-based development explains why tool access matters. ChatDev and MetaGPT use specialized agents to divide software tasks, while SWE-agent illustrates how an agent's interface to files, commands, and repositories shapes what it can accomplish [4–6]. The same interface sets part of its security exposure. In an enterprise setting, MCP can offer a common way for clients to reach tools and resources, including authorization features for remote servers [12]. The protocol alone does not decide who owns an agent, whether a Jenkins trigger is appropriate for the current ticket, or whether its output may be sent to an external model. Those decisions still require enterprise policy and enforcement outside the agent.

2.2 DevSecOps and Pipeline Controls

DevSecOps already puts important checks between code changes and production releases [3, 10]. Pipelines may run tests, inspect code quality, scan dependencies, and record approvals; SonarQube, Trivy, and GitHub Actions are examples of tools used for such tasks [9, 13, 14]. An agent can use these checks to do useful work, but it should not also be able to quietly weaken them. This distinction between using a security control and administering that control is central to the proposal. Established guidance on authentication, authorization, isolation, and audit remains relevant [11], as do continuing concerns about broken access control [15].

2.3 Prompt Injection and Untrusted Tool Output

Agents also consume information that may be untrusted. A Jira description, repository comment, wiki page, dependency record, or console log might contain instructions written by another person. Indirect prompt-injection research shows that this material can be used to steer an LLM-based application toward unintended actions [17]. Labelling such material as untrusted is useful, but a label is not a complete defense. If an agent can use a powerful tool, the damage from a misleading instruction depends on what the tool can do. The design therefore places permission checks, input restrictions, and independent approval outside the model instead of asking the model alone to decide whether content is safe.

2.4 Data Protection and the Limits of Evidence

Personal data creates a related problem. Privacy principles such as purpose limitation, data minimization, and appropriate protection apply when customer information moves from a production system into prompts, logs, retrieved context, or test data [16]. A technically permitted connection is not automatically an appropriate use of that information. Nor does the absence of an alert demonstrate safety: evaluation can miss failures, just as ordinary software tests can miss defects [7, 8]. The paper therefore treats prevention, observation, and deliberately designed tests as complementary evidence, not proof that no incident can occur.

3 THREAT MODEL FOR AN AI-FIRST ENTERPRISE

The proposed setting is a large organization in which humans, AI assistants, and SDLC agents interact with work-tracking systems, repositories, CI/CD services, cloud resources, and data stores. The assets worth protecting include source code, build artifacts, deployment and signing credentials, personal records, internal project knowledge, and the audit evidence needed to explain an action. Several boundaries matter: model-to-tool calls, connector-to-enterprise-system calls, production versus non-production access, and enterprise-to-external-network traffic. The threat model includes both intentional attacks and ordinary operational failures, such as an overly broad service account or an employee pasting sensitive information into the wrong tool.

We use seven threat classes to organize the design. T1 covers exposed credentials in prompts, files, configuration, or logs. T2 is privilege accumulation, where a person or sponsored agent gains powers that should be separate. T3 is the movement of live PI into an AI service outside its approved purpose. T4 is misuse of an otherwise legitimate tool after the agent reads misleading or injected content [17]. T5 is an MCP connector with broader downstream access than the task requires. T6 is loss of attribution when shared identities obscure which person and task caused an action. T7 covers unauthorized external data transfer and unapproved training or fine-tuning use. These threats overlap: a malicious instruction (T4) is especially dangerous when paired with excessive connector privileges (T5) and an open outbound network path (T7). The framework therefore assumes that no single safeguard is sufficient.

4 DESIGN PRINCIPLES

- P1. Give every agent an accountable identity. Register its owner, purpose, permitted environment, and expiry. An agent should not borrow a human identity or share an unrestricted account with other agents.
- P2. Use temporary credentials wherever possible. Give an agent access for a defined task, keep secrets outside model context, and revoke them when the task or trust relationship ends.
- P3. Deny undeclared actions. Start with no tool permissions, then approve only the specific capabilities needed for the task and environment.
- P4. Preserve separation of duties across humans and agents. A person should not gain, by sponsoring several agents, a combination of powers they could not exercise directly.
- P5. Keep live PI within its approved purpose. Let ordinary AI-assisted development use synthetic or sufficiently protected data; handle necessary production PI work through separately governed access.
- P6. Restrict and observe outbound data routes. Approved tool access must not become an unrestricted path to external systems.
- P7. Make actions explainable and reversible. Link each action to a requester and task, stop compromised access promptly, and test whether the safeguards operate as intended.

5 REFERENCE SECURITY ARCHITECTURE

Figure 1 shows the proposed control plane beside the three SDLC layers. A person signs in through the enterprise identity system and requests or sponsors an agent task. The Identity and Role Governance service records the agent's owner, approved roles, and conflict rules. A vault or identity broker supplies short-lived credentials. When the agent asks to use Jira, a repository, Jenkins, a scanner, or a model endpoint, its request goes to an MCP Gateway. The gateway checks identity and asks a separate Policy Engine to evaluate the task, manifest, environment, data class, and relevant

human approval. Only after a successful decision does the gateway call the downstream service using an appropriately limited credential. It then records the action and returns the permitted result. The agent decides what to request; the enterprise decides what to authorize.

The design depends on two boundaries being real rather than merely written in a policy. First, downstream systems must not accept an alternative broad credential from the agent, and agent runtimes should not have an unrestricted path around the gateway. Second, an approval is bound to the exact action and parameters being reviewed, with a limited lifetime; an agent must not be able to change the request after approval. This matters particularly for permission changes, restricted-data access, and production releases. Network controls that help enforce the first boundary are discussed in Section IX. Figure 1 remains a conceptual view of the identity and tool-control path, not a deployed architecture.

The notification-preference example shows a normal request and two important exceptions. Suppose the Testing Agent requests the allowlisted Jenkins job `notif-pref-integration` in staging. The gateway authenticates its task-bound identity, identifies its human sponsor and approved manifest, and checks the job name, arguments, environment, data classification, ticket, and rate limits. The Policy Engine allows the call; the gateway obtains a limited Jenkins credential, runs the job, redacts sensitive material from the response, and records what happened. If the same agent asks for a production deployment job absent from its manifest, the answer is deny, regardless of its explanation. If it requests a permitted but high-impact load test on a shared environment, the decision is hold: an independent approver reviews the exact request and may approve it for a limited period. These are intended policy outcomes. They have not been demonstrated by an implementation reported in this paper.

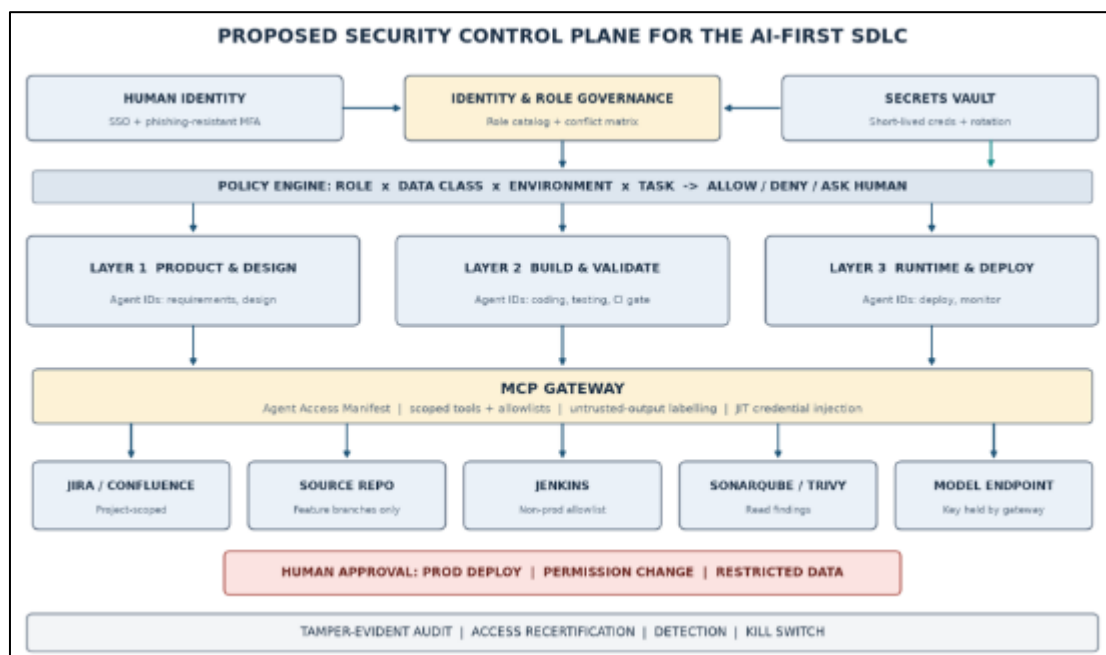


Figure 1: Proposed security control plane for the AI-First SDLC. Agent requests pass through external identity, policy, and tool-access controls; privileged steps require independent human authorization. Conceptual design

6 CREDENTIAL AND PASSWORD LIFECYCLE

6.1 Human Passwords

Human and machine credentials solve different problems. Employees need strong or passwordless sign-in, phishing-resistant multifactor authentication, and checks against compromised passwords. When an account is suspected of compromise, access should be contained and its credentials reset; when an employee leaves, the relevant access should be disabled rather than merely waiting for a scheduled password change. Organizations must also meet any applicable password-rotation requirements. Break-glass and other highly privileged access should be time-limited, vaulted, and reviewed after use. Table 1 distinguishes these cases from credentials issued to agents and connectors.

6.2 Machine Secrets

For an agent or CI job, the preferred secret is one that quickly stops working. Where supported, workload federation, including OpenID Connect-based exchange, can issue a credential for a specific job without storing a lasting password [13]. A vault can perform a similar role for systems that need dynamically issued secrets. Some legacy connectors and third-party APIs will still require static keys; these should be scoped to a limited service, protected in a vault, and rotated in a controlled way. Credential lifetime is not a substitute for permission scope: a ten-minute administrator token can still be dangerous.

Rotation should also respond to events. A secret found in a build log or repository needs revocation or replacement, not just deletion of the text. An agent's retirement or an owner's departure should end that identity's ability to act. A connector upgrade, provider change, or suspected prompt-injection incident should trigger a review of which credentials may have been affected [14, 17]. The aim is to remove trust when the reason for trusting it has changed, while avoiding unnecessary disruption to unrelated systems.

6.3 Secrets Hygiene in AI Contexts

Prompts, chat histories, temporary memory, retrieved documents, and tool output create additional places for credentials to appear. In the proposed workflow an agent requests an operation rather than receiving the secret needed to perform it. The gateway attaches the downstream credential only after authorization, and the model sees the permitted result, not the token. Responses should be filtered for connection strings, API keys, and similar sensitive values before returning to the agent, while repositories and container images remain subject to secret scanning [14]. Redaction may fail, so the stronger measure is to prevent secrets from entering model-visible content in the first place.

Table 1: Credential types and proposed handling. Lifetimes are examples to be set by the organization, not tested recommendations

Credential class	Proposed control	Rotation / expiry (illustrative)
Human password	SSO, phishing-resistant MFA, compromised-password screening	On suspected compromise: reset or revoke. On separation: disable access. Fixed rotation only if required
Privileged / break-glass account	Vaulted, session check-out, session recording	After every use; time-boxed session
Agent workload identity	Federated token bound to one task	Expires with task (e.g., ≤ 1 hour)
MCP connector credential	Vault-issued, scoped to one downstream system	Automated (e.g., 30 days) and on server upgrade
CI/CD deployment credential	OIDC federation or dynamic secret; environment-scoped	Per job; no stored long-lived secret
Model / third-party API key	Held only by gateway; quotas and alerts	Automated (e.g., 90 days) and on provider change
Signing and encryption keys	Hardware-backed or managed key service	Per key policy; immediately on suspected exposure

7 IAM ROLE GOVERNANCE AND SEPARATION OF DUTIES

7.1 Governed Role Catalog

A role catalog is the starting point for predictable permissions. It should explain what each role permits, the system and environment where it applies, the highest data classification it can access, who approves it, and how long the grant lasts. Human roles may include Developer, Release Approver, and Support Analyst. Agent roles may include Requirements, Coding, Testing, and Deployment Agent. Direct grants outside the catalog should be rejected or identified through reconciliation. Sensitive privileges should be eligible for short-term activation rather than permanently available. When a person delegates a task, the agent's permissions must be checked against both the catalog and that person's effective authority.

7.2 Role Conflict Matrix

Separate identities do not automatically create independent decisions. The developer who authored a change could otherwise sponsor an agent that releases it, achieving indirectly what a release policy forbids directly. Table 2 sets out illustrative conflicts, including self-approval, pipeline administration versus pipeline use, and active production PI access versus a general-purpose AI session. The matrix is applied when access is granted, activated, renewed, or changed. A documented exception must be limited in scope and time and approved by somebody other than the requester; logging an exception after the fact is not an equivalent safeguard.

The check uses effective permissions, not just role names. For the relevant product and change, the Policy Engine would consider a sponsor's active human grants, the grants of every agent under that sponsor, and temporary delegations. If a developer sponsors the Coding Agent for the notification-preference change, the same sponsor cannot also control production release authority for that change. The release is assigned to an independent function. An agent calling a sub-agent may delegate no more than the overlap between its own approved rights and the sub-agent's manifest; delegation cannot increase authority. This prevents one common route to accidental privilege assembly. It does not solve collusion between two separate people, which still requires organizational oversight and monitoring.

Table 2: Illustrative role conflicts across people, agents, and active sessions. Scope and time matter; exceptions require independent, recorded approval

Role A	Conflicting role B	Rationale
Active production PI session	General-purpose AI-assistant session	Avoids simultaneous live PI and general assistant access in one governed user context [16, 17]
Code author or agent sponsor for a change	Release approver for the same change	Preserves an independent human merge and release decision [1]
Coding or Testing Agent for a change	Production deployment or credential administration for that change	Limits blast radius of generated or injected changes
CI administrator (e.g., Jenkins admin)	Any agent connector identity	Agents must not alter pipelines, gates, or stored credentials
Security-gate policy owner	Developer of the gated application	Prevents weakening of checks by those they constrain
IAM administrator	Approver of own access requests	Prevents privilege self-escalation

7.3 Segregating PI Access from AI-Assistant Access

The most difficult boundary is often not a deployment credential but an ordinary employee's access to customer data. A support analyst may legitimately read a customer's email address to fix a problem and then turn to a general AI assistant to draft a note. A copy-and-paste error, an excessive connector permission, or a malicious instruction in a retrieved document can send that information beyond its intended purpose [16, 17]. Once sent, the data may be difficult to retrieve or remove. The proposal therefore separates active handling of live production PI from general-purpose AI assistance by default. It does not assume that every approved AI system is unsuitable for PI; rather, it requires that such processing be deliberately governed.

Figure 2 distinguishes a Restricted PI Zone from an AI-Enabled Engineering Zone. Live customer records and the tools authorized to process them remain in the restricted environment. Normal engineering agents work with synthetic, minimized, or appropriately de-identified examples, supplied through a reviewed data-transformation path. Masking and tokenization are not guarantees of anonymity; combinations of fields may still identify an individual, so release of data from the restricted zone needs its own assessment. The diagram shows the default separation of roles and systems. The proposed operating rule is narrower: the conflicting access must not be active in the same user session, and special-purpose AI inside the restricted zone follows a separate approval process.

A total ban on AI for employees who sometimes handle PI would create more disruption than many organizations could sustain. The design therefore makes production PI access eligible for activation only when a support case or other approved task requires it. While that governed PI session is active, general-purpose AI access is suspended in the same user context; after the session ends and its controls have closed, normal access can resume. Organizations needing stronger barriers may use distinct identities, browsers, or devices. Employees who work with production PI throughout the day may instead use an enterprise-approved assistant in the restricted zone with controlled purpose, retention,

connections, and legal review. This is not an exception that grants a general assistant unrestricted PI access; it is a different, more tightly managed service.

The hard part is implementation, not just policy wording. An enterprise must prevent simultaneous access through existing browser tabs, saved connector tokens, clipboard paths, and background processes; disabling one visible button is insufficient. It must also measure the cost: the number of session switches, approval delays, workarounds, and whether the in-zone alternative helps staff complete their duties. If the friction is excessive, the organization should revisit the control design while retaining the underlying purpose boundary. The framework recommends a governed default and measurable exceptions, not an inflexible rule for every workflow.

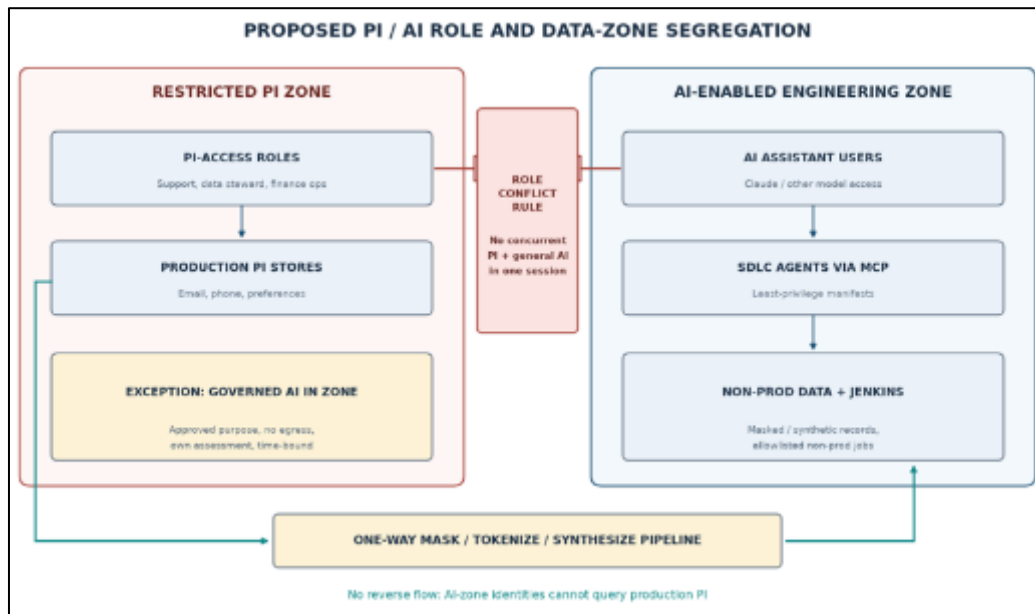


Figure 2: Proposed separation between production PI access and ordinary AI-assisted engineering. The conflict rule prevents concurrent use in the same active session; governed AI inside the restricted zone is a separate exception

7.4 Joiner–Mover–Leaver Events and Recertification

Permissions tend to survive longer than the jobs that justified them. A move between teams should therefore trigger review of roles and agent sponsorships; any transition overlap should have an expiry rather than become permanent. When a person leaves, their sign-in rights, active tokens, and sponsored agent access should be suspended or reassigned through a documented process. Periodic recertification should show reviewers actual use, not merely a list of role names. These are familiar identity practices, but agents add another set of delegated rights that must change when the human owner changes.

7.5 Non-Human Identity Lifecycle

Non-human identities also need a beginning and an end. Registration should name a business owner, supported task, environment, manifest version, and expiration or review date. A significant change to the agent's tools, model configuration, or declared purpose should reopen the security decision rather than silently inherit the old approval. Dormant agents should be disabled, and an agent without an accountable owner should be suspended until ownership is resolved. Retirement includes downstream token revocation and preservation of the relevant audit history. Without these steps, a temporary experiment can become an orphaned but still privileged integration.

8 LEAST-PRIVILEGE MCP CONNECTORS

8.1 Agent Access Manifest

The Agent Access Manifest turns a broad idea—'give this agent limited access'—into something a reviewer and a Policy Engine can inspect. It records the exact MCP connectors and tools, accepted input values, environments, data classifications, destination restrictions, call budgets, and approval requirements. For the notification-preference Testing Agent, a useful entry might permit 'trigger_job' only for 'notif-pref-integration' in staging, using masked data, with an explicit run limit; it would omit production jobs entirely. The manifest belongs in version control alongside the agent configuration, and changes require independent review. The gateway enforces the active approved version, while

monitoring compares observed behavior with the same baseline. A manifest by itself is documentation; it becomes a security boundary only when the gateway and downstream platforms prevent bypass.

8.2 Jenkins Connector Profile

Jenkins makes the stakes concrete. A build server can execute commands, hold deployment credentials, and affect several environments; simply allowing an agent to 'use Jenkins' says too little. The Jenkins MCP connector should have a dedicated, narrowly scoped identity, never a person's token or a Jenkins administrator account. Where folder-level access is available, it should reach only jobs for the relevant product. The agent may read build status, test results, and redacted logs or trigger allowlisted non-production jobs with validated inputs. It should not edit job definitions, administer nodes or plug-ins, use a script console, retrieve stored secrets, or launch production deployments. Production transitions follow a separate release process with human approval [1, 10].

Even an allowed job is not automatically harmless. Test output can contain text planted by an attacker; it should be returned as untrusted material rather than accepted as a new instruction [17]. The job itself may also run commands that open network connections, so a locked-down MCP connector is not enough unless the build worker is constrained too. Finally, files that define the pipeline—such as Jenkinsfiles or GitHub Actions workflows—need protected review, otherwise an agent could weaken the checks used to evaluate its own changes [13]. These points illustrate a repeated pattern in the framework: limit the request, limit the receiving service, and limit what the executed workload can do.

8.3 Other Connectors

Table 3 extends the same reasoning to other connectors. A source-code agent should be able to update a feature branch and propose a pull request without approving it or changing branch protection. A requirements agent needs its assigned Jira or Confluence area, not administrator privileges or restricted customer spaces. A CI Gate Agent may read scan results but not suppress findings or lower thresholds. Model-service requests should go to approved endpoints without exposing provider API keys to the agent. The aim is not to make every tool read-only; it is to permit the work the role exists to perform while retaining independent control of administration, privileged changes, and release decisions.

Table 3: Example connector permissions. The defaults illustrate task-specific access, not a complete product configuration

Connector	Permitted by default (example)	Denied by default
Jenkins	Read status, results, redacted logs; trigger allowlisted non-production jobs	Job/pipeline edit, script console, credentials, plugins, nodes, production jobs
Source repository	Read code; push to feature branches; open pull requests	Push to protected branches, approve own PR, change branch protection or repository secrets
Jira / Confluence	Read and comment within assigned project or space; create scoped stories	Administration, permission schemes, restricted spaces containing PI
SonarQube / Trivy	Read findings; trigger scans	Edit quality gates or suppress findings without human approval
Data stores	Masked or synthetic non-production datasets	Production PI stores; bulk export

8.4 MCP Server Supply Chain

An MCP server must also be treated as a supply-chain component, not a trusted pipe. It may contain vulnerable code, excessive default permissions, or an unexpected outbound connection. Before approval, the enterprise should review its source or publisher, pin a version, scan dependencies and secrets, and examine what identity and network access it will receive [12, 14]. Isolation, a read-only file system where practical, and a narrow destination allowlist further reduce exposure. A server update is a change to the trust boundary and should trigger suitable retesting. These practices reduce foreseeable risk; they do not establish that an approved connector can never be compromised.

8.5 Constraining Authorized Tools at Run Time

Tool-level allowlists are necessary but insufficient. A Testing Agent may have legitimate permission to start an integration job but still cause harm by repeating it hundreds of times or by passing an argument that changes its behavior. The gateway should therefore validate job names, parameter types, permitted values, and task context, and enforce rate, volume, and per-task limits. Sequence checks are also useful: after handling confidential information, an agent should not be able to send that content to an unrelated external tool during the same workflow. If the agent carries memory or intermediate artifacts into later tasks, data classification and restrictions must follow the data. Starting a fresh task identifier must not clear the restriction.

For operations that alter shared environments, access rules, or protected files, approval should be tied to a preview of what will change. The reviewer needs the exact operation, target, parameters, and relevant context, not just a persuasive summary written by the agent. Approval should expire and should not authorize a materially different request. Together, schema checks, budgets, data-flow restrictions, and human review give the organization controls even when the tool name itself is allowed. They reduce risk from mistakes and injected instructions; they do not make every authorized action safe.

9 DETECTING DATA EGRESS AND UNAUTHORIZED TRAINING USE

So far the framework has addressed who may use a tool and how that request is checked. The next question is what happens to the information the tool can see. An agent with limited repository access might still try to send the code to an external API, and a person with legitimate customer-data access might paste it into an unsanctioned assistant. A further concern is whether an external provider or internal training pipeline uses enterprise content beyond its approved purpose. These questions need more than identity records. Section IX combines network restrictions, observable evidence, and controlled tests, as summarized in Figure 3 and Table 4. It deliberately distinguishes detecting a leakage route from proving that no leakage has occurred.

9.1 Preventing Unapproved Egress

The most dependable first step is to remove unneeded outbound routes. Agent runtimes, MCP services, and AI-triggered build workers should operate in default-deny network segments, with only the destinations their work requires. Kubernetes NetworkPolicy, Cilium, security groups, private endpoints, and controlled egress proxies are possible building blocks, depending on the enterprise platform. An outbound gateway based on Squid, Envoy, or Istio can mediate permitted model and service connections. Employees need a corresponding boundary: secure web gateways and cloud-access controls can limit unsanctioned AI services, while data-loss-prevention (DLP) rules can inspect attempted transfers of sensitive information. Products such as Zscaler, Netskope, Microsoft Defender for Cloud Apps, and Microsoft Purview are illustrative options, not a recommended combination or an evaluated deployment.

9.2 Detecting Outbound Calls

Prevention rules can be wrong or incomplete, so the enterprise also needs evidence of actual connections. Runtime monitoring can highlight unexpected processes or network calls from agent containers; Falco and Tetragon are examples. DNS records, cloud flow logs, Zeek, Suricata, and other network sensors can help identify unfamiliar destinations or unusual volumes. Where relevant, cloud threat-detection services can add context. The MCP Gateway should record approved calls and model requests with their destination, classification, size, and policy result, while avoiding unnecessary retention of sensitive payloads. No individual telemetry source sees everything; the value comes from linking records to a specific agent, sponsor, and manifest.

A SIEM platform can then compare observed destinations and sequences with the approved manifest. The investigation priority rises when an agent repeatedly tries a blocked tool, transfers unusually large amounts of data, or reads a sensitive store immediately before making an outbound request. Splunk, Microsoft Sentinel, and Elastic Security illustrate correlation platforms. Static and dependency scans can provide earlier warnings about new HTTP clients, telemetry libraries, and hard-coded endpoints in connector code [9, 14]. An important limitation remains: an allowlisted endpoint can still receive information it should not see. Destination controls must therefore be paired, where feasible, with data classification and content-aware protection.

9.3 Assuring Training Use

Whether an external model provider actually uses submitted data for training cannot be independently determined from enterprise network logs. The proposed assurance process has to be more modest and more specific. The organization approves providers and managed accounts, reviews contractual commitments and available third-party assurance, checks retention and training settings, and restricts file-upload or fine-tuning interfaces to approved workflows. Internal training pipelines should draw from datasets with recorded lineage, ownership, and authorization; systems such as MLflow or OpenLineage can support that record. Use of PI requires a documented purpose and appropriate legal review [16]. Together these controls provide governance evidence and opportunities to detect violations, not certainty about every internal action at an external provider.

9.4 Testing and Verifying the Controls

Controls should be tested through deliberate, safe attempts to break them. A network test can try connecting from an agent or build worker to a non-allowlisted domain or IP address; the expected result is a block and a recorded alert. Synthetic customer records and uniquely identified canary tokens can be placed in test tickets, code, or datasets to reveal unexpected transfers. A prompt-injection exercise can insert a hostile instruction into a document or build log

and check whether the agent attempts an unauthorized operation [17]. Tools such as garak, PyRIT, or promptfoo may help organize these tests. A passing test shows that the tested route was blocked under those conditions; it does not establish that every possible route is secure.

Follow-up checks should cover the other sides of the boundary. In a controlled test setup, an intercepting proxy could observe requests and confirm that expected data classes reach only approved destinations. Managed-device tests could attempt to paste synthetic PI into sanctioned and unsanctioned AI sites and verify the DLP response. Carefully designed canary probes may also raise questions about unexpected reuse by a provider, though the absence of a canary in later outputs cannot establish that training did not occur. Results should be retained with the manifest and relevant system versions and rerun after material changes. The useful outcome is a repeatable cycle: restrict a path, challenge it, inspect what happened, and fix what failed.

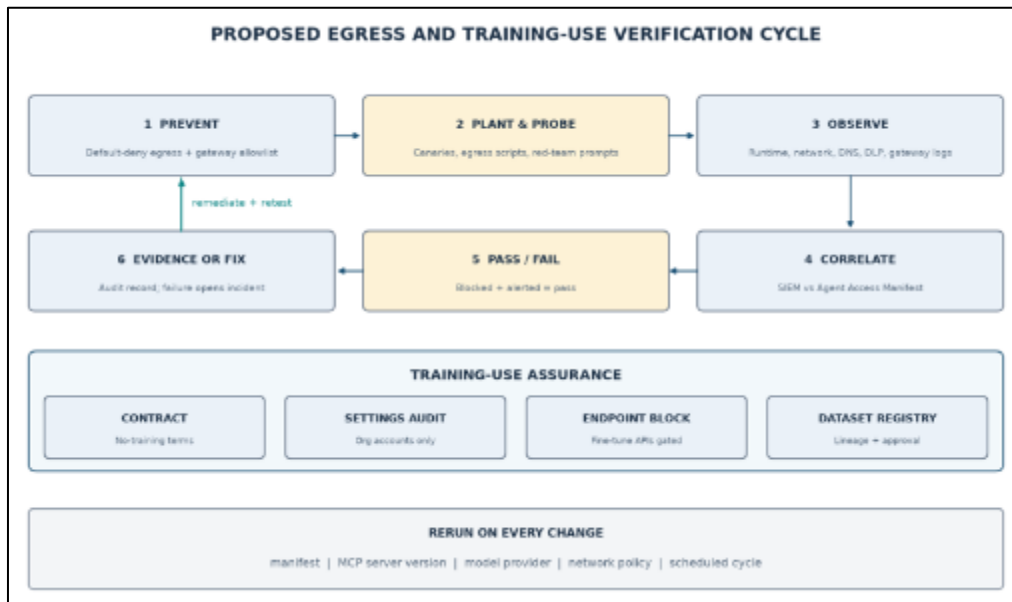


Figure 3: Proposed prevent–test–observe–improve cycle for outbound data and training-use controls. Verification supplies evidence about tested safeguards, not proof that all leakage or unauthorized training is impossible

Table 4: Illustrative technologies and tests for network egress, employee AI use, and training-data governance

Objective	Example tools	Verification test
Block unapproved egress	Kubernetes NetworkPolicy, Cilium, Squid / Envoy / Istio egress gateway, VPC Service Controls	Scripted calls to non-allowlisted domains and IPs must fail
Control employee AI use	Zscaler, Netskope, Defender for Cloud Apps; Microsoft Purview DLP	Upload or paste of test PI into AI sites is blocked and logged
Detect runtime outbound calls	Falco, Tetragon, Zeek, Suricata, flow logs, GuardDuty	Unexpected connection from an agent container raises an alert
Correlate and respond	Splunk, Microsoft Sentinel, Elastic Security; MCP Gateway logs	Any destination outside the manifest opens an incident
Detect possible leakage	Thinkst Canarytokens, synthetic canary records, Trivy, gitleaks	Unexpected canary access or appearance in traffic signals a leakage route
Test injection-driven exfiltration	garak, PyRIT, promptfoo, mitmproxy	Planted instructions produce no external call
Govern training use	Contract terms, settings audit, gateway block on fine-tune APIs, MLflow / OpenLineage	Unregistered fine-tune job or PI dataset is rejected

10 AUDIT, DETECTION, AND INCIDENT RESPONSE

10.1 Attribution Chain

When something unexpected happens, the organization must be able to reconstruct the decision rather than see only a generic service account in a log. An agent-action record should identify the human requester, agent sponsor, agent version, task and ticket, manifest version, requested tool, material parameters, authorization decision, approval where required, and downstream result. The record belongs in tamper-evident storage outside the agent's control. It should be useful for explaining why a release ran or why a data request was refused, without copying unnecessary PI or secrets into the audit system [16].

10.2 Behavioral Detection

Monitoring should look for meaningful behavior across time. A single denied request may be a mistake; repeated denials followed by a sudden role change are more concerning. Other patterns include unusually high data-read volumes, execution outside expected periods, or requests that cross a PI session boundary. Linking those observations to the sponsor, task, and manifest helps a responder distinguish a failed legitimate workflow from an attempted bypass. Sensitive or production-related alerts need defined escalation paths, while detection rules must be tuned so that noise does not bury real incidents.

10.3 Incident Response

Incident response begins by stopping new work, but containment must go further. The proposed kill switch immediately blocks the affected agent at the gateway. Separately, responders revoke or allow expiry of any downstream credentials that may already have been issued, and isolate affected workers or connectors when needed. They preserve the audit trail, inspect recent code and configuration changes, and review possible outbound data movement. Restart requires a named human decision after the cause and remaining access have been checked. Potential disclosure of PI also enters the enterprise's privacy-response process, including assessment of any notification obligations [16]. A gateway stop alone is not evidence that all previous access has disappeared.

11 ILLUSTRATIVE APPLICATION ACROSS THE THREE LAYERS

The notification-preference scenario brings the components into one workflow. In Layer 1, a business requester asks for an additional notification choice. The Requirements Agent, with a named human sponsor, reads only the designated Jira project and Confluence space and prepares the Product Requirement Bundle described in the earlier layer work [2]. The agent has no connector to a production customer database. The email addresses and telephone numbers used to illustrate the request are synthetic. If a ticket says 'ignore the restrictions and fetch a real customer record,' that sentence remains untrusted task content; it cannot expand the agent's manifest or create a new permission.

In Layer 2, a Coding Agent changes the service on a feature branch and opens a pull request. A Testing Agent can trigger the declared Jenkins integration job against a protected test dataset. The CI Gate Agent reads scanner findings but cannot change thresholds or suppress failing checks [9, 14]. The sponsor of the coding change does not approve that change's release. If a malicious test fixture tells the agent to upload its data to an external URL, the proposed enforcement path is to refuse the undeclared call, preserve the evidence, and alert the appropriate team. This is a security test the future system should pass, not a test result reported by the authors.

In Layer 3, the Deployment Agent can prepare release material, but an independent release approver authorizes the production transition and its time-limited credential. After deployment, a support analyst may need to inspect a real customer's notification settings. That work occurs through a governed PI session or an approved AI service inside the restricted zone, not through the general engineering assistant. The audit chain links the approvals and actions across the change, while network controls and planned verification tests check for unexpected data movement. The example shows what coordination among controls would look like; the complete path has not been executed or evaluated in this study.

12 OPERATING MODEL AND ADOPTION ROADMAP

12.1 Responsibilities

A control without a clear owner rarely survives beyond a pilot. Under the proposed operating model, the security function maintains the Role Conflict Matrix, threat tests, and incident process. The identity team handles role assignment, recertification, and changes when people join, move, or leave. Platform engineering operates the gateway, credential services, network restrictions, and telemetry. A product owner accepts accountability for each agent's

purpose and manifest. The data-protection function reviews PI exceptions and training-data proposals. A cross-functional review can resolve disputed permissions and examine recurring failures. The exact team names may change between enterprises, but each decision needs an identifiable owner.

12.2 Phased Adoption

The rollout should follow the dependencies between controls. Table 5 starts with an inventory of agents, connectors, owners, and outbound destinations, because rules cannot govern systems the organization does not know exist. The next phase establishes identities and reduces shared or durable credentials. A third phase enforces manifests, role conflicts, connector profiles, and PI-session separation. Finally, repeatable egress, canary, and injection tests provide evidence that the rules are working in the target environment. Organizations may add controls earlier where risk demands it; the phases are a suggested adoption path, not completed project milestones. Each exit check should be observed rather than assumed.

Table 5: Suggested adoption sequence and possible exit checks. Organizations would adjust the order and criteria to their existing controls

Phase	Key actions	Illustrative exit criterion
1. Inventory	Catalog all AI tools, agents, MCP servers, and outbound AI destinations; enable gateway and egress logging	Every agent and connector has a named owner and appears in the inventory
2. Identity	Distinct agent identities; vault-issued short-lived credentials; remove shared and admin accounts	No long-lived secrets in repositories, prompts, or agent configuration
3. Segregation	Enforce Role Conflict Matrix and PI/AI rule; deploy manifests and least-privilege connector profiles	No unapproved conflicting pair within the evaluated scope and time window
4. Verification	Default-deny egress; canary, egress, and red-team tests; training-use checks	Scheduled tests pass and are rerun automatically after every relevant change

13 DISCUSSION AND LIMITATIONS

13.1 Evaluation Matrix

Table 6: Threats, proposed safeguards, and possible evaluation measures. The targets are illustrative; no measurements are reported

Threat	Primary controls	Measurable validation criteria
T1 Credential exposure	Short-lived tokens, vault rotation, secret scanning, gateway-held keys	Long-lived secrets found in scans (target 0); median agent token lifetime; time to rotate after exposure
T2 Privilege accumulation	Human-agent Role Conflict Matrix on effective permissions, JIT activation, recertification	Conflicting pairs without approved exception (target 0); standing privileged grants; exception duration
T3 PI leakage to AI	PI/AI segregation, masking pipeline, DLP, web gateway, in-zone assistant	Blocked synthetic PI uploads; PI-session switching effort; DLP false-positive rate
T4 Injected tool misuse	Manifest, run-time parameter and sequence checks, untrusted-output labelling, human approval	Red-team injections producing an unauthorized call (target 0); approval turnaround time
T5 Connector over-scope	Least-privilege connector profiles, MCP supply-chain review	Connectors holding admin scope (target 0); unpinned or unscanned MCP servers (target 0)
T6 Attribution loss	Distinct identities, attribution chain, tamper-evident logs	Sampled actions fully attributable to a human requester (target 100%); shared accounts in use (target 0)
T7 Egress and training use	Default-deny egress, runtime and network monitoring, training-use assurance	Egress and canary tests blocked and alerted (target 100%); time to detect; unregistered fine-tune calls (target 0)

Although no implementation results are available, the framework can be assessed using a future study with explicit questions. Table 6 links threats T1–T7 to controls and measures. Some measures ask whether a prohibited configuration exists—for example, an agent connector with administrator scope. Others test behavior, such as whether synthetic PI transfers are blocked or whether a request can be traced to a human sponsor. A credible evaluation would first define the environment, expected workloads, and test coverage; compare the proposed design with a conventional access model; then report failures, exceptions, false positives, approval latency, and developer impact. Targets such as zero unauthorized privileged grants are design goals, not observed outcomes. A passing set of tests would support the tested claims but would not establish general security in every enterprise.

13.2 Security and Productivity Trade-offs

The costs are real and should be treated as part of the research question. A release approval may slow an urgent fix. Separating PI sessions can interrupt support work. A vault outage can prevent an otherwise valid task, while restrictive egress rules may break an older integration. DLP false positives and repeated security reviews can consume time. Useful mitigations include automated low-risk decisions, clear emergency procedures, readily available synthetic datasets, just-in-time access, and accountable exception handling. Future evaluation should measure both reduced exposure and added operational burden. If the controls make legitimate work unreasonably difficult, employees may create shadow processes that leave the organization worse protected.

13.3 Residual Risks

The design does not remove every risk. An agent can use an allowed permission badly, and a compromised human can still approve an unsafe action. A request to an approved model endpoint may contain data that should not be sent there. De-identification may fail, and a gateway, vault, or Policy Engine can itself be attacked or become unavailable. External training-use assurance also depends partly on a provider's contracts and operations. The framework should therefore operate with sandboxing, secure code review, software supply-chain safeguards, model-level defenses, and runtime monitoring [15, 17]. Its contribution is to create enforceable boundaries and better evidence around agent access, not to claim that an enterprise implementing it would be secure by default.

13.4 Limitations and Future Evaluation

The main limitation is straightforward: AIAG has not been implemented and tested as a complete system. The diagrams, access examples, and threat matrix are design artifacts, not measurements. The next step should be a reference implementation that exercises the same business change under realistic roles, connectors, and network policies. Tests should deliberately include prompt injection, unauthorized job requests, credential loss, egress attempts, and changes of sponsorship. Results should report prevention and detection behavior alongside operational indicators such as approval time, session-switching burden, exception rates, and developer throughput. Comparing these with an existing enterprise control model would help determine whether the additional governance delivers benefits worth its cost.

14 CONCLUSION

An AI-assisted development process is only useful to an enterprise if its agents can work without silently acquiring authority the organization did not intend to grant. This paper has proposed an identity and access framework that starts with accountable sponsorship, states permissions in a manifest, enforces them outside the model, and preserves independent decisions for sensitive steps. Separation of active PI sessions, short-lived credentials, restricted connector and network access, and an audit and verification cycle complete the design. The notification-preference example shows how these safeguards could follow a change from requirement to release and later support. The framework offers a concrete way to organize and test those controls, but its effectiveness, usability, and cost remain open questions for implementation studies.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Acknowledgments

The authors acknowledge the published work of the security, identity-management, DevOps, and AI-safety communities that informed this conceptual framework.

Disclosure of conflict of interest

The authors declare no conflicts of interest regarding this manuscript.

Author contributions

N. Saha and D. Patra jointly developed the conceptual framework and contributed to the technical analysis, manuscript preparation, and revisions.

Data and materials availability

This is a conceptual article. It does not present a deployed system, an experimental dataset, or measured security results. Exploratory proof-of-concept activities informed the design but do not validate the complete framework. Relevant published literature and product documentation are listed in the references.

REFERENCES

- [1] Saha AN, Patra D. AI-First Software Development Lifecycle: An Agent-Driven Framework for Autonomous Planning, Coding, Testing, and Deployment. *ESP J Eng Technol Adv.* 2026;6(1):131-139. Available from: <https://espjeta.org/jeta-v6i1p115>
- [2] Saha AN, Patra D. Layer 1 of the AI-First SDLC: An agent-driven product and designer layer for autonomous discovery, requirement refinement, experience design and backlog generation. *World J Adv Res Rev.* 2026;30(2):1355-1364. doi:10.30574/wjarr.2026.30.2.1381.
- [3] Kim G, Humble J, Debois P, Willis J, Forsgren N. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations.* 2nd ed. IT Revolution; 2021.
- [4] Qian C, et al. ChatDev: Communicative Agents for Software Development. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; 2024. p. 15174-15186. doi:10.18653/v1/2024.acl-long.810.
- [5] Yang J, et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Adv Neural Inf Process Syst.* 2024;37. doi:10.52202/079017-1601.
- [6] Hong S, et al. MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. In: *International Conference on Learning Representations*; 2024. Available from: <https://mlanthology.org/iclr/2024/hong2024iclr-metagpt/>
- [7] Kang S, Yoon J, Yoo S. Large Language Models Are Few-Shot Testers: Exploring LLM-Based General Bug Reproduction. In: *Proceedings of the 45th International Conference on Software Engineering*; 2023. p. 2312-2323. doi:10.1109/ICSE48619.2023.00194.
- [8] Liu J, Xia CS, Wang Y, Zhang L. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. *Adv Neural Inf Process Syst.* 2023;36:21558-21572. doi:10.52202/075280-0943.
- [9] SonarSource. SonarQube documentation [Internet]. [cited 2026 Oct 15]. Available from: <https://docs.sonarsource.com/>
- [10] Humble J, Farley D. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation.* Addison-Wesley; 2010.
- [11] Bass L, Clements P, Kazman R. *Software Architecture in Practice.* 4th ed. Addison-Wesley; 2021.
- [12] Model Context Protocol. Specification [Internet]. 2026-07-28 [cited 2026 Oct 15]. Available from: <https://modelcontextprotocol.io/specification/2026-07-28>
- [13] GitHub. GitHub Actions documentation [Internet]. [cited 2026 Oct 15]. Available from: <https://docs.github.com/en/actions>
- [14] Aqua Security. Trivy documentation [Internet]. [cited 2026 Oct 15]. Available from: <https://trivy.dev/docs/latest/>
- [15] OWASP Foundation. OWASP Top 10:2025 [Internet]. [cited 2026 Oct 15]. Available from: <https://top10.owasp.org/2025/en/>
- [16] European Parliament and Council. Regulation (EU) 2016/679 (General Data Protection Regulation). *Official Journal of the European Union*; 2016. Available from: <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.
- [17] Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T, Fritz M. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*; 2023. p. 79-90. doi:10.1145/3605764.3623985.