

(RESEARCH ARTICLE)



## ZERO TRUST FOR MCP-CONNECTED ENTERPRISE AI AGENTS

Adedamola Solanke <sup>1, 2, \*</sup>

<sup>1</sup> IT Touch Ltd., 101 East Park Boulevard, Suite 600, Plano, Texas 75074, United States of America.

<sup>2</sup> Dallas Baptist University, Dallas, Texas, United States of America.

\* Corresponding Author

ORCID Details

Adedamola Solanke: <https://orcid.org/0000-0003-3769-3827>

World Journal of Advanced Engineering Technology and Sciences, 2026, 21(01), 070–076

Publication history: Received on 17 August 2026; revised on 24 September 2026; accepted on 26 September 2026

Article DOI: <https://doi.org/10.30574/wjaets.2026.21.1.0467>

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

Enterprise AI agents now read tickets, documents, and web pages and then act on what they find through tool protocols such as the Model Context Protocol (MCP). Because an agent combines untrusted input with delegated privileges, it creates a path from attacker-influenced content to privileged action that earlier Zero Trust programs did not anticipate. This article examines how the principles of NIST SP 800-207 should be extended for agents. It compares five approaches that organizations use to give agents access to enterprise systems, from shared API keys to task-scoped delegated tokens enforced at a gateway, and evaluates them against five attack classes: indirect prompt injection, tool poisoning, confused-deputy misuse, token theft and replay, and excessive agency. An exposure analysis of a representative service-desk agent shows that task-scoped, audience-bound, sender-constrained tokens reduce exposure by about five orders of magnitude compared with a shared key, while gateway mediation constrains what a hijacked agent can do with the access that remains. The article closes with an adoption roadmap and measures of progress.

**Keywords:** Zero trust; AI agents; Model Context Protocol; Least privilege; OAuth token exchange; Prompt injection

### 1 INTRODUCTION

Organizations that adopted Zero Trust across their cloud estates did so on an assumption that rarely needed stating: every principal requesting access is either a person or a piece of software that behaves the way its code says it will. Earlier implementation guidance for multi-cloud environments rested on that assumption [1]. Identities were federated, permissions were narrowed, and every request was verified, but the workloads behind those requests were predictable. Enterprise AI agents remove that predictability. An agent built on a large language model receives a goal, reads tickets, documents, and web pages, chooses tools, and acts on what it finds. When those tools are a ticketing platform such as ServiceNow, a directory such as Microsoft Entra ID, a corporate mailbox, or a GitHub organization, the agent's choices become changes in production systems.

The Model Context Protocol (MCP), an open protocol released in late 2024, has become a common way to connect agents to such tools through a single client-server interface [Model Context Protocol specification, revision 2026-07-28, <https://modelcontextprotocol.io/specification/2026-07-28/basic/authorization>, Last accessed on 04/10/2026].

Standardization has accelerated adoption, and it has also concentrated risk. An agent that holds broad credentials while reading attacker-influenced content opens a direct path from untrusted input to privileged action, a path that traditional threat models never had to consider. Research on indirect prompt injection has shown that instructions planted in retrieved content can redirect an agent's actions without any compromise of the model, the host, or the network [2].

This article examines how the Zero Trust principles in NIST SP 800-207 [3] need to be extended for agents. Its central contribution is a comparison of the five approaches organizations use today to give agents access to enterprise systems, evaluated against the attack classes that matter most for MCP deployments and illustrated with an exposure analysis of a representative service-desk agent. It closes with an adoption roadmap and the measures that show whether a program is working.

## 2 WHY AGENTS STRAIN ZERO TRUST ASSUMPTIONS

NIST SP 800-207 moves defense from network perimeters to individual resources. A policy decision point evaluates each request using the identity of the requester and its context, and a policy enforcement point carries out that decision [3]. Three properties of agents complicate this arrangement. The first is that instructions and data travel in the same channel. A language model cannot reliably tell the instructions of its operator apart from text that arrives inside an email or a web page, and that weakness is the root of indirect prompt injection [2]. Benchmarks such as AgentDojo and InjecAgent have demonstrated that tool-using agents can be diverted in this way across realistic workspace, travel, and banking tasks [4, 5].

The second property is non-determinism. A conventional service account performs the same operations every time its code runs, so its permissions can be derived from the code itself. An agent selects its tools at run time, which means its permissions have to be derived from the task in front of it rather than from the program. The third property is delegated authority. An agent acts for a user, and the confused-deputy problem, first described for programs that held authority on behalf of others, applies directly whenever an agent or an MCP server exercises broader rights than the party that made the request [6].

## 3 THREAT MODEL FOR MCP-BASED AGENTS

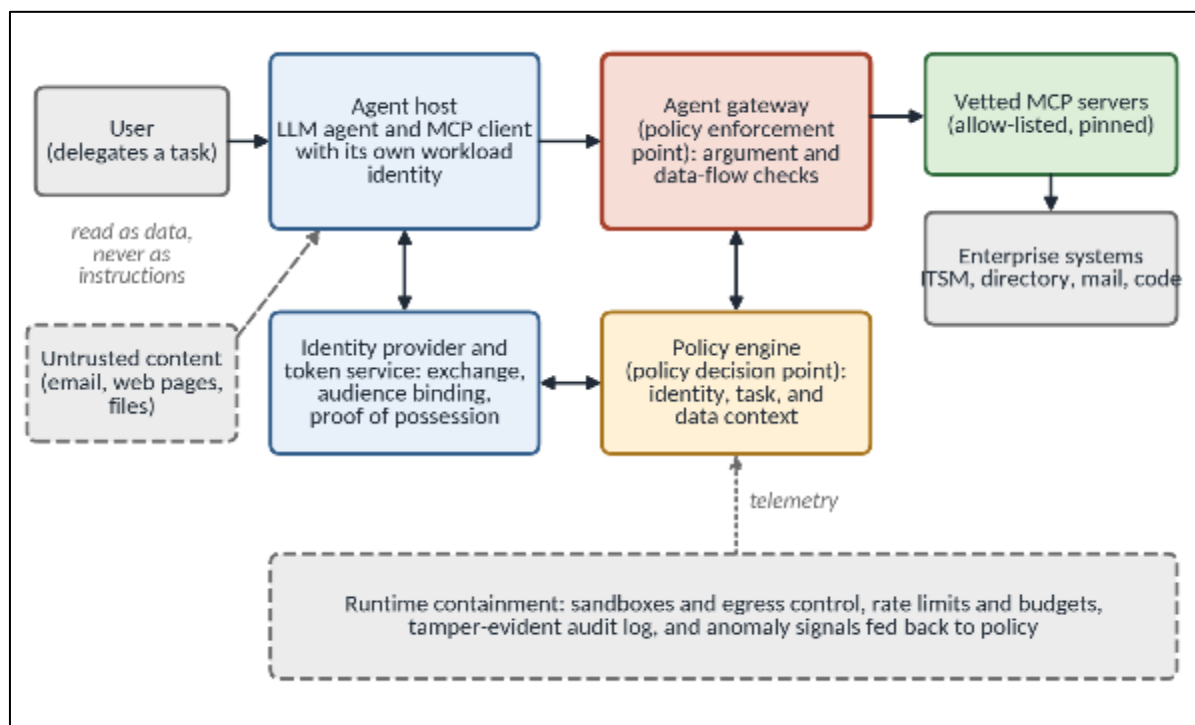


**Figure 1: Reference architecture. Every tool call passes through a gateway that acts as the policy enforcement point, a policy engine decides using identity, task, and data context, and untrusted content reaches the agent only as data.**

The deployment considered here has three parts: a host application that runs the agent, MCP clients inside that host, and MCP servers that expose tools, resources, and prompts. Remote MCP servers authorize requests with OAuth. Under the current revision of the specification, servers publish metadata that tells clients which authorization server to use,

clients name the specific server a token is meant for through resource indicators, and servers must reject any token that was not issued for them and must never pass a client's token through to an upstream service [7] [Model Context Protocol specification].

Five attack classes drive the analysis. Indirect prompt injection plants instructions in content the agent reads [2]. Tool poisoning hides instructions in a tool's description, and a variant known as a rug pull alters the description after the tool has been approved [8] [Invariant Labs, MCP security notification: Tool poisoning attacks, <https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks>, Last accessed on 04/10/2026]. Confused-deputy misuse occurs when a server forwards a user's token or applies its own broader authority to a request [6]. Token theft and replay reuse a stolen bearer token from somewhere else. Excessive agency gives an agent more tools or autonomy than its task requires. OWASP now ranks goal hijacking, tool misuse, and identity and privilege abuse among the leading risks for agentic applications [9]. Figure 1 shows the reference architecture used throughout the rest of the article.

#### 4 FIVE WAYS TO GRANT AN AGENT ACCESS

When security teams connect an agent to enterprise tools, they usually reach for one of five approaches. The approaches differ in who the agent appears to be, how much authority it holds, and how long that authority lasts.

**Table 1: Comparison of access approaches for enterprise AI agents.**

| Approach                         | Blast radius after a hijack                         | Theft and replay resistance                        | Confused-deputy resistance          | Attribution                      | Operating cost |
|----------------------------------|-----------------------------------------------------|----------------------------------------------------|-------------------------------------|----------------------------------|----------------|
| A1 Shared API key                | Every tool the key reaches                          | None; long-lived bearer secret                     | Poor                                | None                             | Very low       |
| A2 User token passthrough        | Everything the user can do                          | Low; replayable where audience is not enforced     | Poor; forbidden by MCP              | Agent masked as user             | Low            |
| A3 Agent identity with RBAC role | Union of all tasks the role covers                  | Medium; credentials live hours to a day            | Medium                              | Agent only                       | Medium         |
| A4 Task-scoped delegated tokens  | Tools of the current task only                      | High; audience-bound, proof of possession, minutes | High; actor claim records the chain | Agent and user                   | Medium to high |
| A5 A4 plus gateway mediation     | Current task, with arguments and data flows checked | High                                               | High                                | Agent, user, and policy decision | High           |

The first approach (A1) issues the agent a shared API key or service-account secret. It is the quickest to deploy, and for that reason it remains common in pilots. The same key often serves several agents, it is rotated quarterly if at all, and every server that trusts it accepts it. Actions taken with it cannot be attributed to a particular agent or a particular user. The second approach (A2) lets the agent reuse the user's own access token or session. With that token in hand, the agent inherits everything the user can do, often for hours at a time through refresh, and its actions become indistinguishable from the user's. Passing such tokens through to upstream services is explicitly forbidden by the MCP specification [Model Context Protocol specification], yet the pattern persists in internal integrations built before that rule was written.

The third approach (A3) gives each agent its own workload identity, for example a SPIFFE identity or a Microsoft Entra workload identity, and binds that identity to a role. Attribution improves considerably, because every action now carries the agent's name. The weakness lies in the role. It has to cover every task the agent might ever perform, so the agent holds the union of those permissions at all times, usually with credentials that live for hours or a full day.

The fourth approach (A4) issues task-scoped delegated tokens. When a task begins, a token service uses OAuth 2.0 token exchange to mint a token whose subject is the user and whose actor claim names the agent, so both parties remain visible in every log [10]. The token carries only the scopes the task needs, is bound by a resource indicator to a single MCP server [7], and is sender-constrained with DPoP so that a stolen copy cannot be replayed from another client [11]. Its lifetime is measured in minutes. The fifth approach (A5) adds mediation to A4. Every tool call passes through a

gateway that validates arguments against schema and policy, applies data-flow rules based on classification, checks pinned hashes of tool descriptions, holds high-impact or irreversible operations for human approval, and blocks egress that policy has not allowed. Architectural defenses that separate trusted control flow from untrusted data, such as CaMeL [12] and the plan-then-execute and dual-model patterns [13], fit naturally behind such a gateway. Table 1 summarizes how the five approaches compare against the criteria that matter in practice.

## 5 MEASURING EXPOSURE: A SERVICE-DESK EXAMPLE

A simple inventory makes the differences concrete. Consider a service-desk agent connected through six MCP servers to a ticketing platform, an identity directory, a mail system, a code repository, a cloud console, and a knowledge base. Together the servers expose 48 tools. Thirty of them only read data, 14 change state, and 4 are high-impact operations such as resetting a user's multifactor authentication or adding someone to a privileged group. The engineer on whose behalf the agent works holds about 160 operations across the same systems, 12 of them high-impact. A typical task, which involves triaging a ticket, confirming a user's group membership, resetting a password, and replying to the requester, needs four tools on three servers and none of the high-impact operations. The inventory is illustrative, and organizations can substitute their own counts.

Exposure is measured here as the number of operations a hijacked agent could invoke multiplied by the number of minutes its credential remains valid. The measure is deliberately simple. It treats every operation as equal and ignores data sensitivity, but it is easy to compute from an inventory and it makes orders of magnitude visible. Table 2 and Figure 2 show the results. A shared key valid for 90 days (A1) yields 6,220,800 operation-minutes. Token passthrough (A2) and a standing agent role (A3) yield 76,800 and 69,120 respectively, figures that look similar until the damage ceiling is considered: passthrough exposes 12 high-impact operations rather than 4, and it hides the agent behind the user's identity. Task-scoped tokens (A4) reduce exposure to 40 operation-minutes, about five orders of magnitude below the shared key, and take every high-impact operation out of reach for this task.

Gateway mediation (A5) does not lower the count any further, and that is precisely the point of including it. What it changes is what the four remaining operations can be made to do. An injected instruction asking the agent to reply to the requester with the contents of another user's ticket stays within the task's scopes, so token scoping alone cannot stop it. A data-flow rule enforced at the gateway can.

**Table 2: Exposure of the service-desk agent under each approach (illustrative inventory).**

| Approach                         | Reachable operations | Credential lifetime | Exposure (operation-minutes) | High-impact operations without approval | Servers reachable with one stolen credential |
|----------------------------------|----------------------|---------------------|------------------------------|-----------------------------------------|----------------------------------------------|
| A1 Shared API key                | 48                   | 90 days             | 6,220,800                    | 4                                       | 6                                            |
| A2 User token passthrough        | 160                  | 8 hours             | 76,800                       | 12                                      | 6 or more                                    |
| A3 Agent identity with RBAC role | 48                   | 24 hours            | 69,120                       | 4                                       | 6                                            |
| A4 Task-scoped delegated tokens  | 4                    | 10 minutes          | 40                           | 0                                       | 1                                            |
| A5 A4 plus gateway mediation     | 4                    | 10 minutes          | 40                           | 0                                       | 1                                            |

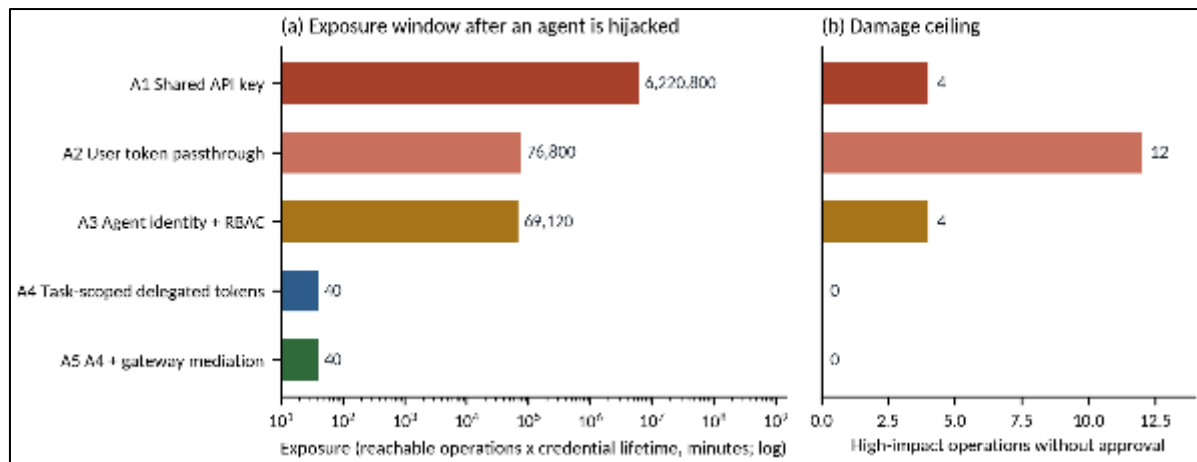


**Figure 2: Exposure of the illustrative service-desk agent. (a) Reachable operations multiplied by credential lifetime, on a logarithmic scale. (b) High-impact operations a hijacked agent could perform without human approval.**

## 6 MAPPING ATTACKS TO CONTROLS

Table 3 maps the five attack classes to the controls that contain them and to the first approach at which those controls become available. No single control covers every class. Indirect prompt injection in particular has no known complete defense, so the realistic goal is to limit what a hijacked agent can do rather than to detect every attempt [4, 13].

**Table 3: Attack classes and the controls that contain them.**

| Attack class                          | How it works                                                 | Contained from | Key controls                                                                                                             |
|---------------------------------------|--------------------------------------------------------------|----------------|--------------------------------------------------------------------------------------------------------------------------|
| Indirect prompt injection             | Instructions hidden in retrieved content redirect the agent  | A5             | Task-scoped tokens; data-flow rules at the gateway; separation of control flow from data; approval for sensitive actions |
| Tool poisoning and rug pulls          | Malicious or later-altered tool descriptions steer the model | A5             | Server allow-list; pinned and hashed tool descriptions; review on any change                                             |
| Confused deputy and token passthrough | A server applies broader authority than the request warrants | A4             | Token exchange with actor claims; audience-bound tokens; no passthrough                                                  |
| Token theft and replay                | A stolen bearer token is reused from elsewhere               | A4             | Proof-of-possession tokens; lifetimes measured in minutes                                                                |
| Excessive agency                      | The agent holds more tools or autonomy than its task needs   | A4             | Per-task tool sets; budgets; tiered human approval                                                                       |

The strongest controls in Table 3 share one characteristic: they are enforced outside the model. Scoped tokens, gateway checks, and sandboxes continue to hold after the model has been fooled, whereas filters that try to recognize malicious text do not. Tool poisoning and token replay, for their part, turn out to be largely supply-chain and credential-hygiene problems, so established practices such as pinning dependencies and binding tokens to their holders address them directly.

## 7 ADOPTION ROADMAP

Few organizations can move from shared keys to mediated, task-scoped access in a single step, and the sequence matters. The first phase is an inventory of every agent, every MCP server it reaches, and every credential it holds, including keys stored in vaults such as HashiCorp Vault or cloud secret managers. The second phase routes all agent tool traffic through a gateway and turns on per-call logging before any policy is enforced. That logging pays for itself,

because it shows which tools each task actually uses, and those observations become the basis for realistic least-privilege policies later.

The third phase gives every agent its own workload identity and replaces static keys with short-lived, audience-bound credentials. Token exchange and task-scoped policy arrive in the fourth phase, written in a policy engine such as Open Policy Agent so that decisions are versioned and reviewable, together with approval tiers for high-impact operations. The final phase adds containment: sandboxes for tool servers and code execution, egress control, budgets, and anomaly signals that feed back into policy, for example when a summarization agent suddenly calls a file-export tool.

Progress can be tracked with a small set of measures: the share of agent tool calls that pass through the gateway, the share of agents with distinct identities, the median lifetime of agent credentials, the number of standing high-impact permissions held by agents, the number of approvals requested per hundred tasks, and the time required to revoke an agent completely. Taken together, these figures show whether exposure is falling and whether the approval workload remains sustainable.

---

## 8 LIMITATIONS AND OPEN PROBLEMS

The approach has limits. No known control eliminates prompt injection, and the evidence to date supports defense in depth and architectural limits on blast radius rather than reliable detection [4, 13]. Delegation chains across multi-agent systems, where one agent hands a task to another, still lack a standard representation. Data-flow policy that security teams can write and review without specialist knowledge remains difficult. Human approval also carries its own risk, since approvers who are asked too often begin to approve by habit, so approval tiers have to be calibrated against measured request rates. The exposure measure used here is a coarse proxy that ignores data sensitivity and the likelihood of compromise. Finally, the MCP specification continues to evolve, and implementers should check authorization requirements against its current revision [Model Context Protocol specification].

---

## 9 CONCLUSION

Zero Trust remains the right frame for agentic systems, but its subjects now include non-deterministic software that reads untrusted text. Among the five access approaches compared here, only task-scoped, audience-bound, sender-constrained tokens reduce exposure by orders of magnitude, and only gateway mediation constrains what a hijacked agent can do with the access it still holds. Treating agents as distinct principals, issuing narrow and short-lived delegated authority, mediating every tool call, and assuming compromise give enterprises a coherent, standards-aligned path to deploying MCP-connected agents. The exposure analysis should be repeated with measured inventories and incident data as organizations accumulate them.

---

## STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

### *Disclosure of conflict of interest*

Adedamola Solanke declares that he has no conflict of interest.

---

## REFERENCES

- [1] Solanke AA. (2021). Zero trust security architectures for multi-cloud environments: Implementation strategies and measurable outcomes. *World Journal of Advanced Engineering Technology and Sciences*, 3(2), 122-134.
- [2] Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T and Fritz M. (2023). 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 (AISec)*, 79-90.
- [3] Rose S, Borchert O, Mitchell S and Connelly S. (2020). Zero trust architecture. NIST Special Publication 800-207, National Institute of Standards and Technology, Gaithersburg, MD, USA.
- [4] Debenedetti E, Zhang J, Balunovic M, Beurer-Kellner L, Fischer M and Tramer F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In: *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.
- [5] Zhan Q, Liang Z, Ying Z and Kang D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In: *Findings of the Association for Computational Linguistics: ACL 2024*.
- [6] Hardy N. (1988). The confused deputy (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4), 36-38.

- [7] Campbell B, Bradley J and Tschofenig H. (2020). Resource indicators for OAuth 2.0. RFC 8707, Internet Engineering Task Force.
- [8] Hou X, Zhao Y, Wang S and Wang H. (2025). Model Context Protocol (MCP): Landscape, security threats, and future research directions. arXiv preprint arXiv:2503.23278.
- [9] OWASP GenAI Security Project. (2025). OWASP Top 10 for agentic applications. Open Worldwide Application Security Project.
- [10] Jones M, Nadalin A, Campbell B, Bradley J and Mortimore C. (2020). OAuth 2.0 token exchange. RFC 8693, Internet Engineering Task Force.
- [11] Fett D, Campbell B, Bradley J, Lodderstedt T, Jones M and Waite D. (2023). OAuth 2.0 demonstrating proof of possession (DPoP). RFC 9449, Internet Engineering Task Force.
- [12] Debenedetti E, et al. (2025). Defeating prompt injections by design. arXiv preprint arXiv:2503.18813.
- [13] Beurer-Kellner L, et al. (2025). Design patterns for securing LLM agents against prompt injections. arXiv preprint arXiv:2506.08837.