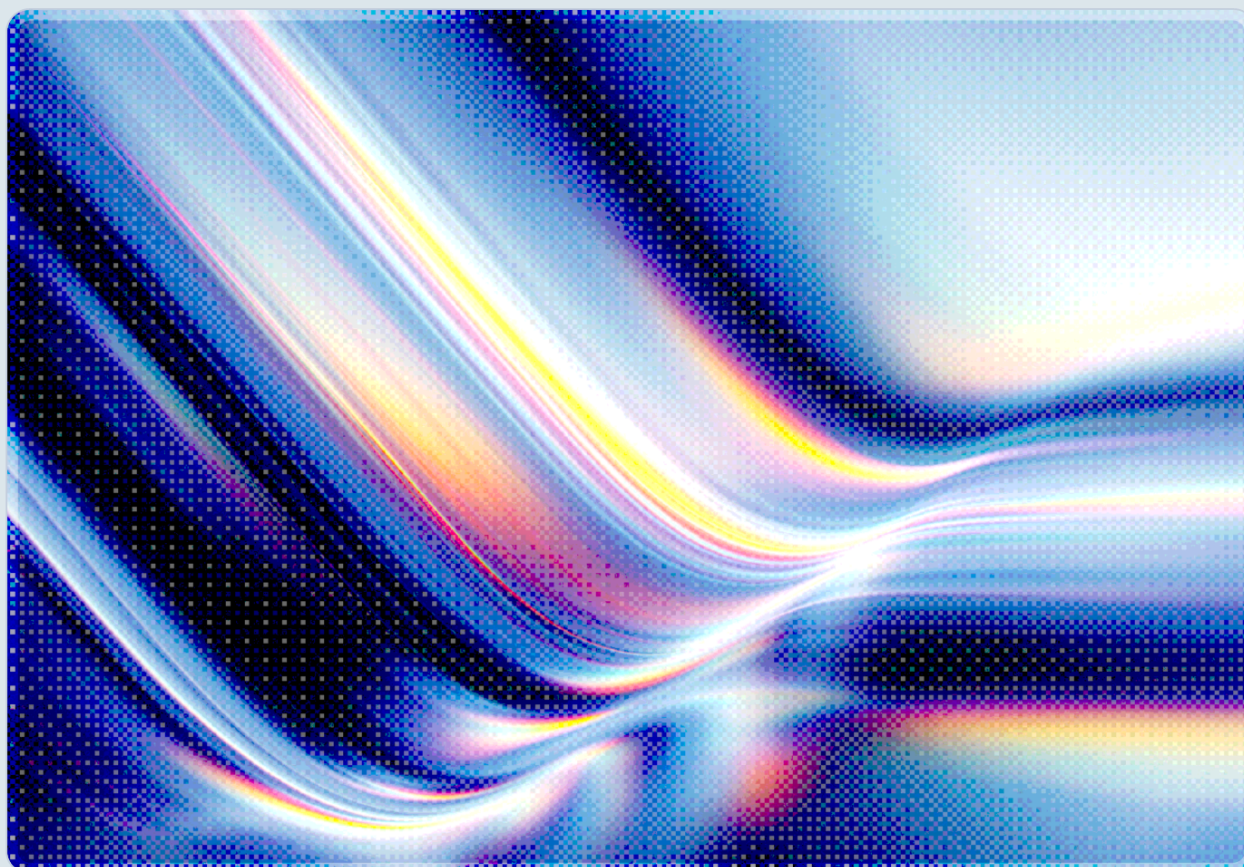


The MCP Python SDK OAuth Flaw and the Cost of Implicit Trust

2026-09-30

 AI-assisted Rapid Research



© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

- A high-severity flaw (CVSS 7.5) in the official Model Context Protocol (MCP) Python SDK allowed a malicious MCP server to redirect a connecting client's OAuth client secret, authorization code, and PKCE proof key to an attacker-controlled endpoint, enabling full account takeover of the downstream identity provider session [1][2][3].
- The root cause was a missing issuer-validation step on a legacy fallback discovery path: when a malicious server returned an HTTP 404 to the standard authorization-server discovery request, the SDK retrieved OAuth configuration directly from that untrusted server and accepted it without verifying the server's identity [2][3].
- The flaw affected SDK versions 1.9.1 through 1.29.1 and 2.0.0 through 2.1.1; it is fixed in 1.30.0 and 2.2.0, and organizations running MCP clients that authenticate against external identity providers should upgrade immediately and re-register stored OAuth clients with an explicit issuer binding [3][4].
- This vulnerability is consistent with a broader pattern CSA has observed in agentic AI infrastructure, in which protocols and SDKs treat a connected server's self-reported metadata as trustworthy by default [5] – an assumption that breaks down once agents connect to third-party or unvetted MCP servers.
- No public evidence of active exploitation has been reported as of this writing [1][2], but the disclosure adds to a growing body of MCP authentication and authorization findings [5][6] that argue for treating agent-to-service credential flows as a distinct, auditable identity problem rather than an extension of conventional web OAuth.

Background

The Model Context Protocol, introduced by Anthropic and now maintained as an open standard [6][7], allows AI agents and the applications that host them to connect to external tools, data sources, and services through a common interface. Because many of the services an agent needs to reach, such as SaaS platforms, cloud consoles, and internal line-of-business systems, are already protected by enterprise identity providers, the protocol's authorization specification builds on OAuth 2.1, requiring MCP clients to behave as OAuth clients and MCP servers to behave as either resource servers or intermediaries that direct the client to the correct authorization server [7]. This design choice appears

intended to let MCP inherit the maturity of an established, widely deployed standard rather than inventing a bespoke agent authentication scheme, based on the specification's stated rationale [7], and in principle it lets a single OAuth client implementation work across many different MCP servers and identity providers without custom integration work for each one.

That reuse comes with a structural risk that is specific to agentic architectures: the MCP server, not the identity provider, is often the first party to tell the client where to send its credentials. In conventional OAuth deployments, a developer hard-codes or centrally configures the authorization server endpoint for an application, so a compromised or malicious counterparty has no opportunity to redirect the credential exchange. In MCP's discovery model, the client instead asks the server it just connected to, which may be operated by an unknown third party, where the authorization server lives, and then follows that answer. This convenience feature suits an ecosystem where agents are expected to connect to many independent MCP servers with minimal manual configuration, but it also means that server-supplied metadata becomes part of the trust boundary unless the client independently verifies it.

The Python SDK is one of the two reference implementations maintained alongside the TypeScript SDK, and it was in this discovery and metadata-verification logic that Yuval Elbar of Cycode identified the flaw disclosed on September 28, 2026, and tracked under GitHub Security Advisory GHSA-qx49-fqc8-xw99 [1][2][3]. The finding requires little of the attacker: rather than crafting a sophisticated exploit, a malicious MCP server can trigger the vulnerable path simply by declining to answer a standard discovery request.

Security Analysis

The MCP Python SDK implements a two-step process for locating the identity provider that protects a given MCP server: it first requests authorization-server metadata from a well-known discovery endpoint, and if that endpoint responds normally, the SDK validates that the returned issuer matches the server it expects before proceeding. The flaw affected two separate paths that fall outside this well-behaved case. When a server responded with an HTTP 404, signaling that it did not support the modern discovery mechanism, the SDK fell back to requesting OAuth configuration directly from the MCP server itself, and it accepted that configuration without any issuer verification. A related gap existed in the unattended machine-to-machine providers, `ClientCredentialsOAuthProvider` and `PrivateKeyJWTAuthProvider`, which had no mechanism at all for binding stored client credentials to a specific, expected authorization server, so a server could substitute its own endpoint even outside the discovery fallback path [3][4].

A victim connects an MCP client to a malicious or compromised server; the server returns a 404 to the discovery request and then supplies its own OAuth metadata, pointing the authorization endpoint at the real identity provider, such as Google, Okta, or Azure AD, but pointing the token endpoint at infrastructure the attacker controls. The user is shown an authentic login and consent page at the real provider and approves it, since nothing in that step is falsified. The resulting authorization code is returned to the client as expected, but the client then bundles that code with its client secret and PKCE proof key and sends all three to the token endpoint named in the attacker-supplied configuration rather than the real provider's token endpoint. With those three values in hand, the attacker exchanges them directly with the legitimate identity provider for a valid access token carrying whatever scopes the original request had requested, effectively taking over the session with no further interaction from the victim [2]. Because the visible parts of the login experience are genuine, a user has no reliable way to detect the resulting attack chain, which is straightforward for an attacker to execute.

The affected version ranges and the identity-provider mechanisms the flaw touches are summarized below.

Component	Affected	Fixed	Notes
MCP Python SDK (1.x line)	1.9.1 – 1.29.1	1.30.0	No issuer validation on any discovery path
MCP Python SDK (2.x line)	2.0.0 – 2.1.1	2.2.0	Issuer checks present on modern path; missing on 404 fallback and 403 step-up paths
<code>ClientCredentialsOAuthProvider</code> / <code>PrivateKeyJWTAuthProvider</code>	All versions prior to fix	Requires explicit <code>issuer=</code> parameter	Machine-to-machine credentials previously had no server-binding mechanism

Component	Affected	Fixed	Notes
MCP-built servers, local stdio clients, custom token handlers	Not affected	N/A	Vulnerability is specific to HTTP-based OAuth clients using SDK defaults

The advisory carries a CVSS base score of 7.5 (High) for the unattended machine-to-machine providers, and 6.5 for the interactive `OAuthClientProvider` path described above, which requires a user to connect to and authenticate through a malicious or compromised server [3]. As of this writing, no CVE identifier has been publicly assigned to the advisory, and no confirmed instances of exploitation in the wild have been reported by the researchers or by Anthropic's MCP maintainers [1][2].

Beyond the immediate technical fix, this finding raises an architectural question rather than a purely technical one. MCP's discovery convenience appears to trade off against exactly the property this flaw exploited: reducing integration friction across a large, growing ecosystem of third-party servers also reduces the client's default skepticism of server-supplied metadata, asking a client to trust operational details supplied by the counterparty it is least certain about. This is a variant of a problem CSA has already flagged in adjacent MCP research, where the protocol's optional and permissive stance on authorization has repeatedly produced gaps that downstream implementations must close on their own rather than inherit by default [5]. Because agent-to-service authentication increasingly spans multiple identity providers, delegation chains, and machine-to-machine credential types, a single unvalidated assumption in a widely reused SDK can propagate the same weakness into every application built on top of it, which is precisely what happened here across the reported affected version ranges.

Recommendations

Immediate Actions

Organizations running MCP clients built on the affected Python SDK versions should upgrade to 1.30.0 or 2.2.0 without delay, particularly for any client that authenticates against external or third-party MCP servers rather than internally operated ones. Because the fix introduces mandatory issuer binding for the machine-to-machine providers, teams using `ClientCredentialsOAuthProvider` or `PrivateKeyJWTAuthProvider` should expect to add an explicit `issuer` parameter and

should clear and re-register any stored OAuth client registrations created under the vulnerable versions, since those registrations were never bound to a specific authorization server. Security teams should also review logs for any MCP client connections made to unfamiliar or newly added servers during the affected window, since a successful exploitation would appear in logs as an otherwise normal authorization code exchange.

Short-Term Mitigations

Beyond patching, organizations should inventory which internal applications embed the MCP Python SDK, since the vulnerability lived in a widely reused dependency rather than in any single product, and a dependency-level fix does not guarantee that every downstream consumer has rebuilt and redeployed against the patched version. Teams operating their own MCP servers should also confirm they publish standards-compliant authorization-server metadata rather than relying on client-side fallback behavior, since servers that behave correctly reduce the chance that a client falls into a vulnerable discovery path even against an unpatched SDK. Where feasible, restricting MCP clients to an allowlist of known, vetted servers for any workflow that involves OAuth-protected identity providers reduces the attack surface by limiting which servers can attempt the malicious-metadata path, while broader patching completes.

Strategic Considerations

The recurrence of authentication and authorization gaps across the MCP ecosystem suggests that agent-to-service credential flows warrant governance distinct from conventional application OAuth deployments. Because MCP servers are frequently third-party, rapidly added, and not subject to the same procurement or security review as traditional SaaS integrations, organizations should treat the authorization server metadata an agent receives from any server as untrusted input by default, verified independently rather than accepted as configuration. This argues for extending existing non-human identity and workload identity practices, such as short-lived, task-scoped credentials and explicit server-to-issuer binding, into agentic tooling rather than assuming that reusing OAuth libraries automatically confers OAuth's security properties. Governance, risk, and procurement functions evaluating MCP-based integrations should ask vendors directly which SDK version and discovery path their implementation uses, since the answer determines whether this specific class of credential-redirection risk applies to their deployment.

CSA Resource Alignment

This finding connects directly to three recent CSA AI Safety Initiative research artifacts. **AI Agent Identity Sprawl: The Enterprise Authorization Crisis** documents how enterprise AI agents accumulate credentials across multiple trust domains and protocols with limited audit visibility; the MCP Python SDK flaw is a concrete instance of the multi-protocol trust gap that report describes, since the vulnerability specifically exploited the ambiguity between an MCP server's self-reported configuration and the identity provider a client believes it is talking to. **Sandboxing Agentic AI: Least-Privilege Patterns for MCP and Coding Agents** prescribes identity and capability-scoping controls for MCP deployments and explicitly discusses OAuth 2.1 and workload identity federation as part of a reference architecture; the credential-binding gap in `ClientCredentialsOAuthProvider` and `PrivateKeyJWTAuthProvider` is the kind of unscoped, unbound machine-to-machine credential that report's recommendations target. **Non-Human Identity Management for Agentic AI: Extending IAM Beyond Humans** frames prompt injection and credential proximity as structural gaps between agent identity and conventional IAM discipline; the same reasoning applies here, since a client that trusts server-supplied metadata by default has effectively let an external party influence a credential exchange that IAM practice would normally treat as a hard-coded, out-of-band trust relationship. Collectively, these three artifacts and the AI Controls Matrix (AICM) v1.1 identity and access management domain provide the control language organizations can use to evaluate whether their own MCP integrations exhibit the same implicit-trust pattern this vulnerability exposed.

References

- [1] Ravie Lakshmanan. "[Official MCP Python SDK Flaw Can Let Malicious Servers Steal OAuth Credentials](#)." The Hacker News, September 29, 2026.
- [2] Yuval Elbar / Cocode Research. "[MCP SDK OAuth Flaw Enabled Account Takeover](#)." Cocode Blog, September 28, 2026.
- [3] Model Context Protocol Maintainers. "[OAuth client could send credentials to an authorization server other than the protected resource's \(GHSA-qx49-fqc8-xw99\)](#)." GitHub Security Advisories, September 28, 2026.
- [4] GBHackers Security Team. "[Anthropic MCP Python SDK Flaw Enables OAuth Credential Theft and Account Takeover](#)." GBHackers, September 2026.
- [5] Cloud Security Alliance AI Safety Initiative. "[MCP Security Crisis: Systemic Design Flaws in AI Agent Infrastructure](#)." CSA Labs, May 4, 2026.
- [6] Model Context Protocol. "[Authorization Specification](#)." Model Context Protocol Documentation, 2026.
- [7] Model Context Protocol. "[Understanding Authorization in MCP](#)." Model Context Protocol Documentation, 2026.