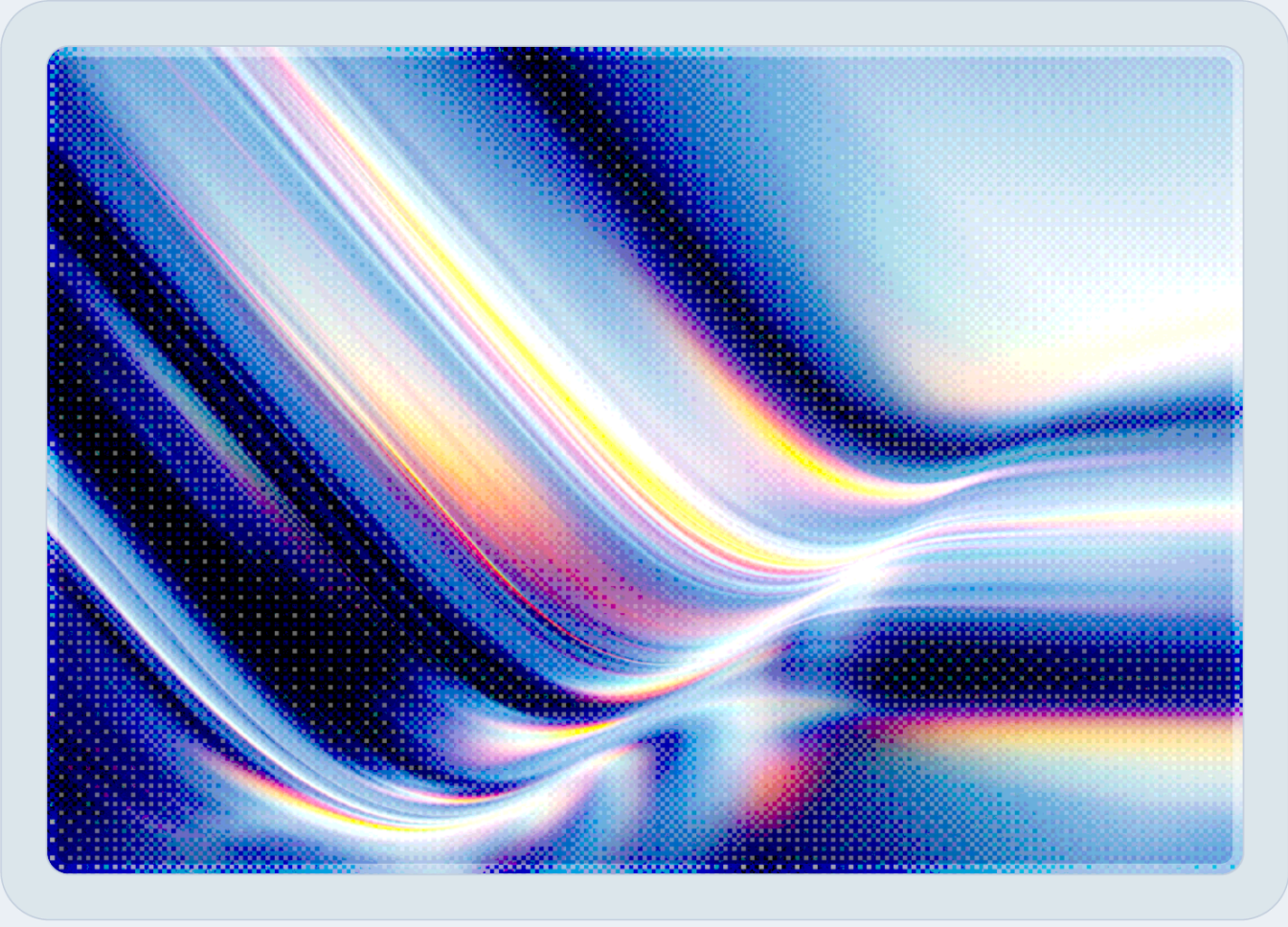
CSAI Foundation | Cloud Security Alliance

# MCP's Transport Trust Gap: Security Implications and Guidance

## Browser-Reachable Servers, Unbound OAuth Issuers, and Cross-Client Message Leakage

2026-11-09

AI-assisted Rapid Research



CSAI | CSA cloud security alliance®

**© 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

Between December 2025 and October 2026, the maintainers of the official Model Context Protocol (MCP) software development kits published a series of security advisories, most rated High severity, that can be read as expressions of one underlying weakness rather than a set of unrelated bugs. In each case, the SDK relied on an unverified signal to decide who it was talking to: a browser page was treated as a legitimate local client, an MCP server was allowed to choose which authorization server received a client's credentials, and a non-unique JSON-RPC request number was used to decide which client received a response. This note calls that pattern MCP's transport trust gap. The term is an analytical framing introduced by CSA for this note, not an industry-recognized vulnerability class.

The first expression of the gap is browser-reachable local servers. The TypeScript, Python, and Go SDKs all shipped without DNS rebinding protection enabled by default for localhost HTTP servers, allowing a malicious website to invoke tools on unauthenticated localhost HTTP servers that had not enabled rebinding protection (CVE-2025-66414, CVE-2025-66416, CVE-2026-34742) [1][2][3]. The Go SDK separately accepted cross-site `text/plain` POSTs that bypass CORS preflight (CVE-2026-33252) [4].

The second is unbound OAuth issuers. The Python and TypeScript OAuth clients let a malicious MCP server steer client secrets, authorization codes, PKCE verifiers, and refresh tokens to an attacker-chosen token endpoint (CVE-2026-115563, CVE-2026-104850), each rated CVSS 7.5 [6][7]. Upgrading alone does not close this gap, because operators must also configure an explicit expected issuer and clear or re-bind stored credentials [6][7].

The third is cross-client message leakage. Shared server or transport instances in the TypeScript SDK routed one client's responses and notifications to another, because routing keyed on JSON-RPC `id` alone and SDK clients typically number requests from zero (CVE-2026-25536, CVE-2026-111770) [12][13]. A related advisory found that experimental tasks were not tied to the session that created them (CVE-2026-111767, CVSS 8.6) [14].

The priority action for security teams is to inventory every MCP SDK dependency, upgrade to the fixed releases listed in this note, and treat the post-upgrade configuration steps (expected issuer, audience checks, per-session instances) as mandatory rather than optional.

# Background

MCP defines how AI agents and their host applications connect to external tools and data. The protocol supports two standard transports: stdio, where a client launches a server as a local subprocess, and Streamable HTTP, where every JSON-RPC message is an HTTP POST to a single endpoint [5]. The stdio transport inherits the operating system's process boundary as its trust boundary, and the advisories that address transport scope explicitly exclude stdio deployments from impact [1][2][6][7]. The HTTP transport has no such inherited boundary, so the SDK and the operator must establish one. That requirement is where the gap opened.

The specification has long stated what that boundary should look like. The current revision (2026-07-28) requires servers to validate the `Origin` header on all incoming connections to prevent DNS rebinding, recommends binding local servers to 127.0.0.1 rather than all interfaces, and recommends authentication for all connections [5]. On the authorization side, the same revision builds on OAuth 2.1, Protected Resource Metadata (RFC 9728), Resource Indicators (RFC 8707), and Authorization Server Issuer Identification (RFC 9207), and it requires servers to accept only tokens issued specifically for them as the intended audience [8]. Issuer identification entered the specification through SEP-2468, which was merged in May 2026 and assigned to the 2026-07-28 release [9]. Under the resulting text, clients must compare the `iss` value in an authorization response against the expected issuer when it is present, while authorization servers are recommended, but not yet required, to include it [8].

The difficulty is that a specification requirement does not protect anyone until a library enforces it, and the reference SDKs are widely used by MCP builders. The advisories show the reference implementations lagging the specification in three areas. Defaults left protections off, metadata supplied by an untrusted server was used to decide where secrets went, and server-side objects that developers reasonably assumed were safe to share were not isolated per client. Because the official SDKs are a common dependency across MCP clients, servers, gateways, and agent frameworks, a default chosen in one of them is likely to propagate to the many projects that depend on it, a concentration risk CSA has examined separately as agent protocol monoculture [21].

This note focuses on advisories in which the SDK trusted an unverified signal about a party's identity or position. The same TypeScript advisory listing includes other High-rated issues from October 2026, such as memory exhaustion and missing request body and batch size limits, but these are availability issues rather than trust issues and fall outside the scope of this analysis.

# Security Analysis

The three classes of defect described below differ in mechanism but share a common structure. In each, an HTTP-layer component made a security decision based on information that an adversary could supply or that did not uniquely identify the party concerned. The subsections that follow examine each class, the fixes released for it, and the work that remains after upgrading.

## Browser-Reachable Local Servers

Developers sometimes run MCP servers on localhost over HTTP, for example so that a desktop agent, an IDE, and a browser-based tool can share one server process. The implicit assumption is that "localhost" means "only my software can reach it." DNS rebinding breaks that assumption. An attacker-controlled DNS server first resolves the attacker's hostname to the attacker's web server with a very short TTL, then, when the browser re-resolves, answers with 127.0.0.1; the browser continues to treat subsequent requests as same-origin, and the attacker's script can now talk to the local MCP server. The TypeScript SDK advisory describes the result directly: an attacker could invoke tools or access resources on the user's behalf if an unauthenticated localhost server had not enabled rebinding protection [1]. Parallel advisories were published for the Python SDK the same day (fixed in 1.23.0) and for the Go SDK in March 2026 (fixed in 1.4.0), and all three are rated CVSS v4 7.6 and classified as CWE-1188, initialization of a resource with an insecure default [1][2][3].

A related Go SDK flaw shows that rebinding is not the only browser path. The Streamable HTTP handler accepted cross-site POST requests without checking `Origin` or requiring `Content-Type: application/json`, so a page could send MCP messages as `text/plain`, a CORS-safelisted type that skips preflight, and potentially trigger tool execution on servers without authorization configured [4]. That issue was fixed in Go SDK 1.4.1 [4]. The practical lesson is that a localhost MCP server holding real capabilities, such as file access, shell execution, or authenticated SaaS sessions, is an attack surface for every website the user visits unless the server validates the request's origin and requires authentication.

Browser vendors are narrowing this path from the other side. Chrome's Local Network Access feature, which introduced a permission prompt in Chrome 142, requires user permission before a public site can send requests to local network or loopback addresses [15]. This is a useful backstop, but it depends on browser version, enterprise policy, and the user declining a prompt, so it should not be treated as a substitute for server-side origin validation.

## Unbound OAuth Issuers

The second class concerns the client side of MCP authorization. In the MCP flow, a client that receives a 401 from a server fetches that server's Protected Resource Metadata, which names an authorization server; the client then fetches that authorization server's metadata and proceeds with sign-in or token refresh [8]. The trust problem is that the MCP server, which may be untrusted, is the party telling the client where to send its credentials.

The Python SDK advisory (GHSA-qx49-fqc8-xw99, CVE-2026-115563) states that the OAuth client did not validate the authorization server metadata `issuer` on every discovery path and did not bind stored or pre-provisioned client credentials to the authorization server they belonged to [6]. A malicious server could either name its own authorization server or publish no Protected Resource Metadata and serve authorization server metadata that impersonated the user's real issuer, receiving the `client_secret`, authorization code, and PKCE `code_verifier`, or a signed client assertion [6]. Affected releases were 1.9.1 through 1.29.x and 2.0.0a1 through 2.1.x, fixed in 1.30.0 and 2.2.0 [6]. Two days later the TypeScript SDK published the parallel advisory (GHSA-6qxp-vccf-f47h, CVE-2026-104850), under which a malicious server could obtain stored `refresh_token` and `client_secret` values without user interaction; affected releases were `@modelcontextprotocol/sdk` 1.12.0 through 1.30.1 and `@modelcontextprotocol/client` 2.0.0 through 2.1.0 [7].

Three aspects of these fixes deserve attention because they make upgrading alone insufficient. First, the fixes are partly opt-in. Python's `ClientCredentialsOAuthProvider` and `PrivateKeyJWTOAuthProvider` require an explicit `issuer=` argument, and on 1.30.0 omitting it emits a `DeprecationWarning` that Python hides by default [6]; TypeScript's bundled providers similarly require `expectedIssuer` [7]. Second, credentials stored before the upgrade remain unbound until operators add an issuer or clear them so clients re-register [6][7]. Third, the TypeScript advisory states plainly that new interactive sign-ins still go to the authorization server the MCP server names, and that direct calls to `refreshAuthorization()` and `exchangeAuthorization()` are not covered [7].

That third point reflects a structural limit rather than an SDK bug. The specification's RFC 9207 validation requires clients to record the expected issuer before redirecting and to compare the `iss` value in the authorization response against it [8][16]. This mitigates classic mix-up attacks, but the specification itself notes that the check "provides no protection if the expected issuer was obtained from an unvalidated source" [8]. When the MCP server is the source of the issuer, RFC 9207 confirms only that the response came from the authorization server the server chose. Organizations therefore need an independent statement of which issuers each MCP server is permitted to use, maintained outside the protocol exchange.

The same pattern appears on the server side and in transport plumbing. The TypeScript SDK's bearer authentication had no built-in audience check, so a token issued for another service by a shared authorization server was accepted (CVE-2026-111771, CVSS 6.8, Medium); the fix adds an `expectedResource` setting that is off by default [10]. Separately, the TypeScript HTTP client transports and OAuth functions followed redirects to any origin, re-sending custom headers, the `mcp-session-id` header, and on 307/308 redirects the request body, which could contain refresh tokens and client secrets (CVE-2026-111769, CVSS 6.5, Medium) [11].

## Cross-Client Message Leakage

The third class affects multi-client HTTP servers. In February 2026 the TypeScript SDK disclosed that reusing a single `StreamableHTTPServerTransport` across clients, or calling `connect()` on one `McpServer` with multiple transports, could send one client's responses, progress notifications, sampling requests, or elicitation prompts to another client (CVE-2026-25536, CVSS 7.1) [12]. The root cause was that request IDs mapped to HTTP streams, client SDKs typically begin numbering at zero, and colliding IDs overwrote each other's routing entries [12]. Version 1.26.0 converted the silent misrouting into explicit errors [12]. In October 2026 a further advisory reported the same class of defect in the 2.x `@modelcontextprotocol/server` package, affecting 2.0.0 through 2.2.0 and fixed in 2.3.0 (CVE-2026-111770, CVSS 7.7) [13]. That advisory also notes that the README examples for the `hono`, `express`, `fastify`, and `node` adapters showed the shared-instance pattern, so developers following the official documentation were led toward the unsafe design [13].

The confidentiality consequences are serious for any multi-tenant deployment, such as a hosted MCP gateway or a shared enterprise tool server, because tool results commonly contain records, documents, or credentials scoped to the requesting user. The patched 2.x SDK now refuses to be shared and rejects a second `connect()` with `ALREADY_CONNECTED`, which means applications built on the unsafe pattern will fail after upgrading and must be restructured to create a server instance per request or per session [13]. A closely related TypeScript advisory, the highest-scored in this set, found that experimental tasks were not tied to the session that created them (GHSA-22jm-h49p-29qw, CVE-2026-111767, CVSS 8.6), affecting `@modelcontextprotocol/sdk` 1.24.0 through 1.31.x and fixed in 1.32.0 [14]. It fits the same pattern of session ownership left unverified, and it suggests that per-client isolation inside the SDK warrants continued scrutiny.

The 2026-07-28 specification revision removes protocol-level sessions, the standalone GET stream, and server-initiated JSON-RPC requests over HTTP, requiring notifications sent within a request's response stream to relate to that originating request, while opt-in change notifications travel on a separate

`subscriptions/listen` stream [5]. This design may reduce the shared routing state that produced these bugs. However, servers that retain backward compatibility with 2025-era clients continue to implement the older session model [5], so the risk persists for as long as legacy revisions remain in service.

## Summary of Advisories

The table below consolidates the advisories discussed in this note, their severity, the releases that contain fixes, and the action that remains for operators after upgrading. The final column is the most important for remediation planning, because in most rows the upgrade alone does not complete the fix.

| Class | Advisory / CVE | CVSS | SDK | Fixed in | Residual action after upgrade |
|---|---|---|---|---|---|
| Browser-reachable | CVE-2025-66414 [1] | 7.6 (v4) | TypeScript | sdk 1.24.0 | Apply `hostHeaderValidation()` on custom Express setups |
| Browser-reachable | CVE-2025-66416 [2] | 7.6 (v4) | Python | mcp 1.23.0 | Configure `TransportSecuritySettings` on low-level servers |
| Browser-reachable | CVE-2026-34742 [3], CVE-2026-33252 [4] | 7.6 (v4) / 7.1 | Go | 1.4.0 / 1.4.1 | Configure origin verification; add authorization |
| Unbound issuer | CVE-2026-115563 [6] | 7.5 | Python | mcp 1.30.0 / 2.2.0 | Pass `issuer=`; clear unbound registrations; rotate secrets |
| Unbound issuer | CVE-2026- | 7.5 | TypeScript | sdk 1.31.0 / | Pass `expectedIssuer`; re-bind stored credentials |

| Class | Advisory / CVE | CVSS | SDK | Fixed in | Residual action after upgrade |
|---|---|---|---|---|---|
| | 104850 [7] | | | client 2.2.0 | |
| Token audience | CVE-2026-111771 [10] | 6.8 | TypeScript | sdk 1.32.0 / server 2.3.0 / express 2.0.2 / server-legacy 2.3.1 | Enable `expectedResource` and return `AuthInfo.resource` |
| Redirect leakage | CVE-2026-111769 [11] | 6.5 | TypeScript | sdk 1.32.0 / client 2.3.0 | Avoid `redirectPolicy: 'follow'` |
| Cross-client leak | CVE-2026-25536 [12], CVE-2026-111770 [13] | 7.1 / 7.7 | TypeScript | sdk 1.26.0 / server 2.3.0 | Restructure to per-request or per-session instances |
| Session isolation | CVE-2026-111767 [14] | 8.6 | TypeScript | sdk 1.32.0 | Confirm experimental tasks are scoped to the creating session |

© 2026 Cloud Security Alliance. AI-assisted Rapid Research.

# Recommendations

The recommendations below are organized by time horizon. Immediate actions focus on applying available fixes together with the configuration they depend on; short-term mitigations address gaps that the SDK fixes leave open; and strategic considerations concern how organizations should plan MCP adoption given the pattern these advisories reveal.

## Immediate Actions

Security teams should begin by building an inventory of every application, agent framework, gateway, and internal tool that depends on an official MCP SDK, including transitive dependencies, and should upgrade each to at least the fixed versions shown in the table above. For TypeScript deployments, upgrading to `@modelcontextprotocol/sdk` 1.32.0, or to `@modelcontextprotocol/server` and `@modelcontextprotocol/client` 2.3.0 (with `@modelcontextprotocol/express` 2.0.2 and `@modelcontextprotocol/server-legacy` 2.3.1 where used), installs the fixes for every TypeScript advisory discussed here [10][11][13][14]; several of those fixes take effect only after the configuration changes described below. Python deployments should run `mcp` 1.30.0 or 2.2.0 [6], and Go deployments `go-sdk` 1.4.1 or later [4].

Because several fixes are opt-in, the upgrade must be accompanied by configuration changes. Every OAuth client provider that holds a client secret or signing key should be given an explicit expected issuer, stored OAuth registrations and tokens created before the upgrade should be cleared or re-bound, and any client that may have connected to an untrusted MCP server should have its client secret or signing key rotated and its tokens revoked, as both advisories recommend [6][7]. Python operators should surface `DeprecationWarning` output in testing so that a missing `issuer=` argument is not silently ignored [6]. Finally, any localhost HTTP MCP server should be confirmed to bind to 127.0.0.1, validate `Origin`, and require authentication, or should be migrated to stdio where its use case permits [5].

## Short-Term Mitigations

Over the following weeks, organizations should close the gaps that SDK upgrades leave open. MCP servers acting as OAuth resource servers should enforce token audience validation, either through the new `expectedResource` setting or by rejecting tokens whose `aud` claim does not name the server, which works on all SDK versions for JWT access tokens; deployments using opaque tokens require introspection to obtain the audience [10]. Multi-client HTTP servers should be reviewed for shared `McpServer` or

transport instances and refactored to a factory, per-request, or per-session pattern before upgrading, since the patched SDK will reject the shared pattern at runtime [13]. HTTP clients that cannot yet upgrade should use a custom `fetch` that refuses redirects [11].

The issuer problem calls for a control outside the SDK. Organizations should maintain an allowlist that maps each approved MCP server to the authorization servers it is permitted to name, and should complete interactive sign-ins only for servers on that list, which is consistent with the TypeScript advisory's guidance to complete sign-ins only for trusted servers [7]. Where MCP traffic passes through a gateway, the gateway is a practical enforcement point for this mapping and a natural place to log every Protected Resource Metadata response that names an unexpected issuer.

### Strategic Considerations

The pattern across these advisories suggests that MCP's HTTP transport is still maturing toward the security properties that the specification describes. Organizations adopting MCP at scale should assume additional advisories in this space, track the official SDK security advisory pages as a standing intelligence source, and prefer deployment patterns, such as stdio for local tools and authenticated, audience-bound HTTP for remote tools, whose security does not depend on every default being correct.

Planning should also account for the protocol's direction of travel. The 2026-07-28 revision's removal of protocol-level sessions and its RFC 9207 requirements address parts of this gap at the specification level [5][8], and the authorization specification signals that a future revision is expected to upgrade authorization server inclusion of `iss` from a recommendation to a requirement [8]. Migration plans that retire legacy session-based clients will reduce the routing complexity associated with cross-client leakage. More broadly, the issuer problem indicates that agent-to-service credential flows warrant treatment as a distinct identity governance concern, with explicit, externally maintained trust relationships between MCP servers and identity providers rather than trust inferred from metadata the server itself supplies.

## CSA Resource Alignment

CSA's research note *The MCP Python SDK OAuth Flaw and the Cost of Implicit Trust* [17] analyzed the Python issuer vulnerability shortly after disclosure and concluded that protocols and SDKs were treating a connected server's self-reported metadata as trustworthy by default. The present note extends that finding in two directions. It shows that the same flaw appeared in the TypeScript SDK days later [7], and it identifies the same implicit-trust assumption in transport-level defects that have nothing to do with OAuth, namely origin handling and per-client message routing. The earlier note's recommendation to treat agent credential flows as an auditable identity problem is reinforced by the residual gap that SDK fixes leave for new interactive sign-ins.

*Identity Confusion by Design: The Grafana MCP SSRF* [18] traced a chained Grafana MCP server compromise to the conflation of `Mcp-Session-Id` values, which exist for conversation-state tracking, with authentication credentials, and argued that the specification anticipated but did not prevent that confusion. The advisories in this note show the same category of error in the reference SDKs themselves: transport-level identifiers such as JSON-RPC request IDs, origins, and server-supplied issuer metadata were used as if they established who a party was. That parallel suggests transport identity confusion belongs in MCP threat models generally rather than in vendor-specific risk registers.

*Agent Protocol Monoculture: MCP's Shared Systemic Risk* [21] examined how dependence on a single protocol and its reference implementations concentrates risk across the agent ecosystem. The advisories discussed here are a concrete instance of that concern, since an insecure default in one official SDK reaches every client, server, and gateway built on it.

*Sandboxing Agentic AI: Least-Privilege Patterns for MCP and Coding Agents* [19] recommends placing a least-privilege boundary between agents and their tools and scoping credentials narrowly. That guidance limits the impact of every class described here: a browser-reachable server with minimal capabilities offers little to a rebinding attacker, narrowly scoped and audience-bound tokens reduce the value of a leaked credential, and per-tenant isolation reduces exposure from cross-client leakage.

At the control framework level, the CSA AI Controls Matrix (AICM) v1.1 [20] provides a control structure that maps to these recommendations. Its Identity and Access Management domain applies to issuer allowlisting, audience validation, and credential rotation; its Threat and Vulnerability Management domain applies to SDK dependency inventory and advisory tracking; and its Application and Interface Security domain applies to origin validation, redirect handling, and per-client session isolation in MCP server code.

# References


[1] Model Context Protocol. "DNS Rebinding Protection Disabled by Default in Model Context Protocol TypeScript SDK for Servers Running on Localhost (GHSA-w48q-cv73-mx4w, CVE-2025-66414)." GitHub Security Advisory, December 2, 2025.

[2] Model Context Protocol. "DNS Rebinding Protection Disabled by Default in Model Context Protocol Python SDK for Servers Running on Localhost (GHSA-9h52-p55h-vw2f, CVE-2025-66416)." GitHub Security Advisory, December 2, 2025.

[3] Model Context Protocol. "DNS Rebinding Protection Disabled by Default in Model Context Protocol Go SDK for Servers Running on Localhost (GHSA-xw59-hvm2-8pj6, CVE-2026-34742)." GitHub Security Advisory, March 30, 2026.

[4] Model Context Protocol. "Cross-Site Tool Execution for HTTP Servers without Authorization in github.com/modelcontextprotocol/go-sdk (GHSA-89xv-2j6f-qhc8, CVE-2026-33252)." GitHub Security Advisory, March 18, 2026.

[5] Model Context Protocol. "Streamable HTTP, Specification Revision 2026-07-28." modelcontextprotocol.io, July 2026.

[6] Model Context Protocol. "OAuth client could send credentials to an authorization server chosen by the MCP server (GHSA-qx49-fqc8-xw99, CVE-2026-115563)." GitHub Security Advisory, September 28, 2026.

[7] Model Context Protocol. "OAuth client could send credentials to an authorization server chosen by the MCP server (GHSA-6qxp-vccf-f47h, CVE-2026-104850)." GitHub Security Advisory, September 30, 2026.

[8] Model Context Protocol. "Authorization, Specification Revision 2026-07-28." modelcontextprotocol.io, July 2026.

[9] Model Context Protocol. "SEP-2468: Recommend Issuer (iss) Parameter in MCP Auth Responses." GitHub, merged May 17, 2026.

[10] Model Context Protocol. "Server bearer authentication accepted tokens issued for another service (GHSA-rvq5-wwqv-78pq, CVE-2026-111771)." GitHub Security Advisory, October 5, 2026.

[11] Model Context Protocol. "HTTP client transports and OAuth requests followed cross-origin redirects and re-sent custom headers and request bodies (GHSA-6prh-2h8m-c8cw, CVE-2026-111769)." GitHub Security Advisory, October 2, 2026.

[12] Model Context Protocol. "Sharing server/transport instances can leak cross-client response data (GHSA-345p-7cg4-v4c7, CVE-2026-25536)." GitHub Security Advisory, February 4, 2026.

[13] Model Context Protocol. "Shared transport or server instances delivered one HTTP client's messages to another (GHSA-jm63-cq47-j5jp, CVE-2026-111770)." GitHub Security Advisory, October 15, 2026.

[14] Model Context Protocol. "Experimental tasks were not tied to the session that created them (GHSA-22jm-h49p-29qw, CVE-2026-111767)." GitHub Security Advisory, October 2, 2026.

[15] Chris Thompson, Google Chrome. "New permission prompt for Local Network Access." Chrome for Developers, June 9, 2025 (updated September 29, 2025).

[16] K. Meyer zu Selhausen, D. Fett. "RFC 9207: OAuth 2.0 Authorization Server Issuer Identification." IETF, March 2022.

[17] Cloud Security Alliance. "The MCP Python SDK OAuth Flaw and the Cost of Implicit Trust." CSA AI Safety Initiative, September 2026.

[18] Cloud Security Alliance. "Identity Confusion by Design: The Grafana MCP SSRF." CSA AI Safety Initiative, September 2026.

[19] Cloud Security Alliance. "Sandboxing Agentic AI: Least-Privilege Patterns for MCP and Coding Agents." CSA AI Safety Initiative, May 30, 2026.

[20] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, accessed November 9, 2026.

[21] Cloud Security Alliance. "Agent Protocol Monoculture: MCP's Shared Systemic Risk." CSA AI Safety Initiative, accessed November 9, 2026.

© 2026 Cloud Security Alliance. AI-assisted Rapid Research. 