Model Context Protocol (MCP) Security

Workstream 4: Secure Design Patterns for Agentic Systems

Version 7ec1306f2f55563f6eeef9d36a6bb2b531491ceb

8 January 2026

Contents

Model Context Protocol (MCP) Security

Approved by the CoSAI Project Governing Board on 8 January 2026.

Abstract

Since its emergence a year ago, MCP has rapidly established itself as the protocol for transmitting structured context between AI agents and services. Given the growing importance and attack surface of MCP and agentic systems, it is imperative that deployment specific security threats are identified and improvements are made to address the challenges and ambiguities inherent in MCP implementations. Our primary goal is to share actionable security guidance for today's MCP implementations while identifying areas where the protocol and ecosystem may need to evolve to address emerging threats. We introduce short and medium-term security implications related to MCP through the introduction of twelve core threat categories and almost forty threats. Our taxonomy distinguishes between traditional security threats amplified by AI and MCP, and novel attack vectors. For each threat and category, we propose mitigations, defenses, and best practices for using MCP across multiple deployment scenarios including enterprise use cases. Multiple critical CVEs have been reported and incidents such as data leakage have already occurred across MCP/agentic deployments. Several examples are mentioned with links in section 2.1.

Scope

This paper focuses on the security aspects of MCP implementations, covering:

- Security analysis of the June (2025-06-18) and latest revised specification MCP transport and protocol layers
- Threat modeling strategies for MCP-based agentic systems[1]
- Supply chain security considerations for MCP servers and tools
- Identity and access management challenges in agentic architectures consuming MCP endpoints
- Best practices for secure MCP deployment and operation
- Recommendations for protocol enhancements to address identified security gaps

We are collaborating with Anthropic and the MCP maintainer community to ensure our recommendations are practical and implementable. This work also coordinates with CoSAI's Software Supply Chain Security workstream to ensure comprehensive coverage of agentic system security concerns.

Anti-Scope

This paper does not address:

- Content safety, misinformation, or AI ethics concerns unrelated to security
- General AI model safety or alignment issues beyond their security implications

[1]Agentic System Definition - An agentic system is an AI-powered solution that autonomously handles one or more tasks within a business workflow, replacing human decision-making nodes with automated processes that can range from simple single-task agents to complex networks of interconnected AI agents working together. The scope and sophistication of an agentic system directly correlates with both its potential economic value and operational risk, as organizations can choose to automate anything from individual yes/no decisions to entire business functions depending on their risk tolerance and automation goals.

- Detailed implementation of specific security tools (we provide guidance, not implementation). Follow on papers will provide reference implementations and recommendations for specific mitigation controls.
- Non-security aspects of MCP performance, scalability, or functional capabilities
- Legal or regulatory compliance requirements (though our recommendations may support compliance efforts). For these matters, see the assets being created from our Workstream 3: AI Risk Governance.

Target Audience

Primary audience:

- **Security professionals and developers securing, creating or connecting to MCP servers**. This includes developers, security practitioners, enterprise architects, and organizations who are implementing, deploying, or integrating MCP servers and clients into their agentic AI systems.
- (longer term) MCP maintainers, the broader MCP ecosystem including contributors to MCP implementations and tooling, and the core protocol developers at Anthropic and beyond

Status: Draft

1. MCP Overview

MCP is an open standard developed by Anthropic and a growing open source community that provides a structured framework for connecting large language models and AI agents to external tools, data sources, and services. MCP addresses a fundamental challenge in agentic AI systems: how to enable models to access and interact with real-world resources dynamically while maintaining security and reliability through standardization. MCP simplifies AI application integration with databases, APIs, file systems, web services, and other external resources reducing the need for custom integrations for each tool or service.

MCP operates through a client-server architecture where AI applications (hosts) use MCP clients to establish connections with MCP servers that expose specific capabilities such as tools, resources, and prompts. The protocol defines standardized methods for:

- discovering available capabilities including tools or services
- invoking parametrized services
- accessing data resources

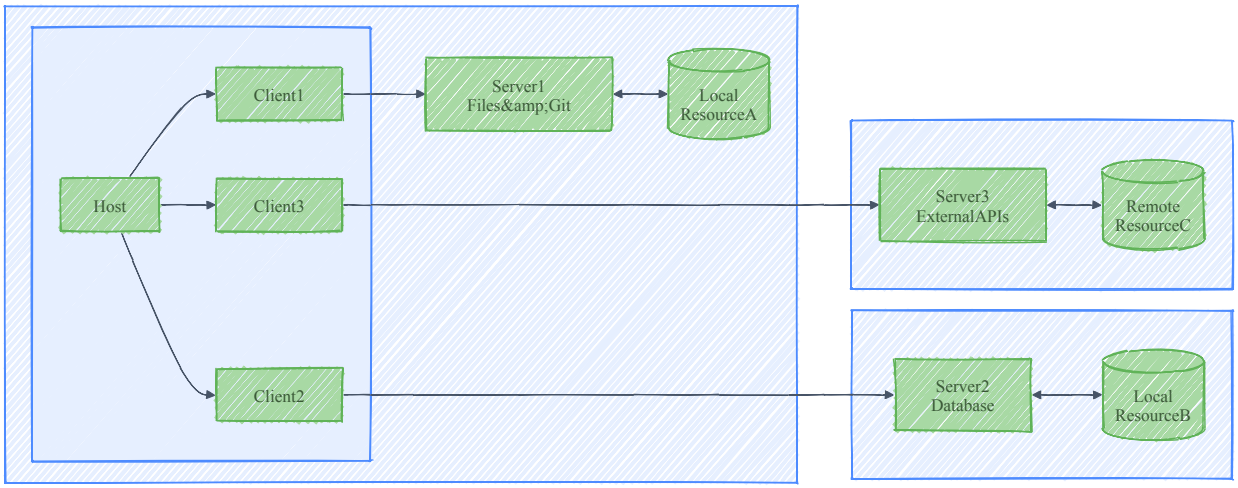
MCP supports multiple transport mechanisms including standard I/O (stdio) for local processes and Streamable HTTP for networked communications, enabling flexible deployment scenarios from local development environments to distributed cloud architectures.[2]

1.1 MCP Architecture

MCP follows a client-server architecture where host applications (such as AI assistants, IDEs, or workflow automation tools) use MCP clients to connect to local or remote MCP servers. Each client-server connection operates as a dedicated, stateful session that begins with initialization and capability negotiation. Communication is built on JSON-RPC, which defines the message format and protocol semantics including lifecycle management and core primitives (tools, resources, and prompts). The transport layer handles the delivery of these JSON-RPC messages be-

[2]Server-Sent Events (SSE) over HTTP has been deprecated in the 2025-06-18 revision of the MCP.

tween clients and servers, supporting stdio for local processes and Streamable HTTP for remote servers.



1.1.1 MCP Deployment Patterns

MCP servers can be deployed across diverse environments with varying trust relationships and security implications. The security posture of an MCP deployment depends on several intersecting factors: where the server code originates (first-party, open source, third-party), where it executes (local machine, internal infrastructure, external cloud), and what resources it can access (local files, enterprise systems, external services).

Each deployment pattern creates distinct trust boundaries that fundamentally shape the threat model. Local deployments using stdio transport provide process-level isolation but require careful management of file system access. Internal network deployments using Streamable HTTP must consider lateral movement risks and authentication requirements. External cloud deployments introduce internet-facing attack surfaces and multi-tenancy concerns.

1. **All-Local**: The MCP client and server are co-located leveraging stdio or http transports
2. **Single-Tenant Hybrid**: The MCP client connects to a single-tenant hosted MCP server over http. The MCP client may run locally or be hosted remotely.
3. **Multi-Tenant Cloud**: MCP clients from multiple tenants connect to a shared MCP server.

These deployment patterns each carry a distinct threat model based on the trust boundaries between client and server. Threat models differ by which components are trusted or untrusted, single- vs. multi-tenant setups, and local vs. remote deployments.A deeper analysis of the security implications of each is presented in the Appendix.

2. MCP Threat Model

2.1 Threat Landscape and Methodology

As with many newly adopted technologies there are numerous examples of significant security incidents across MCP deployments.

- **Asana AI incident** (May 2025): Tenant isolation flaw allowed cross-organization data contamination affecting up to 1,000 enterprises. (Read more)

- **WordPress Plugin vulnerability**: Over 100,000 sites affected by privilege escalation via MCP in AI Engine plugin, patched June 18, 2025 (Read more)
- **Supabase MCP Issue** Researchers demonstrated how prompt injection via support ticket data could cause AI tools like Cursor to expose private tables through a connected MCP server with direct database access, exploiting excessive tools and overprivilege. (Read more)

This section examines current threats through documented incidents and establishes a threat model addressing MCP's unique interdependency challenges.

2.2 Why MCP Requires a Different Approach

While there are numerous, high quality frameworks addressing AI risk (e.g. MITRE ATLAS, NIST AI RMF, MAESTRO, etc.) MCP introduces fundamentally new security considerations:

- Protocol-level authentication between AI clients and tool servers
- Dynamic capability negotiation that determines what tools AI can access
- Distributed trust relationships across multiple independent tool providers
- Session management complexities unique to long-lived AI conversations

Though existing frameworks are designed to assess complex multi-component systems, they assume components behave predictably according to predefined logic. MCP places an LLM, an agent whose behavior is shaped by natural language input, at the center of security-critical decisions, requiring a fundamentally different threat model.

3. MCP Threats

This framework organizes nearly forty threats to MCP deployments across twelve distinct categories, spanning the full technology stack—from foundational identity and access controls through AI-specific boundary failures to supply chain and operational visibility requirements. This model enables security teams to prioritize defenses based on the specific threats and attack surfaces relevant to their deployment. The taxonomy distinguishes between traditional security concerns amplified by AI mediation and novel attack vectors unique to LLM-tool interactions, providing clear guidance for implementing defense-in-depth strategies across the MCP ecosystem.

The first set of risk categories (T1–T2) covers foundational identity and access control risks critical to understanding the origins of a request and how it is being executed through the complex interactions of agents and tools. Next are threats related to input handling stemming from both traditional and AI-specific threats (T3–T4), protection of data and code confidentiality and integrity (T5–T6), and network and transport security (T7–T8). Lastly, MCP risks go beyond a protocol and reference implementation, and spans the MCP lifecycle, including how organizations use and manage MCP capabilities: managing trust relationships (T9), governing resources (T10), ensuring secure supply chains (T11), and maintaining visibility (T12).

3.1 MCP Specificity

The broad applicability and diverse deployment models and supported transports results in a large number of applicable threats. Threats are divided into three tiers:

- **Tier 1 - MCP Specific Threats (7 Threats)**: Novel risks and threats due to MCP's architecture and design decisions.
- **Tier 2 - MCP Contextualized Threats (8 Threats)**: known threats that manifest differently in MCP contexts or are amplified in MCP deployments

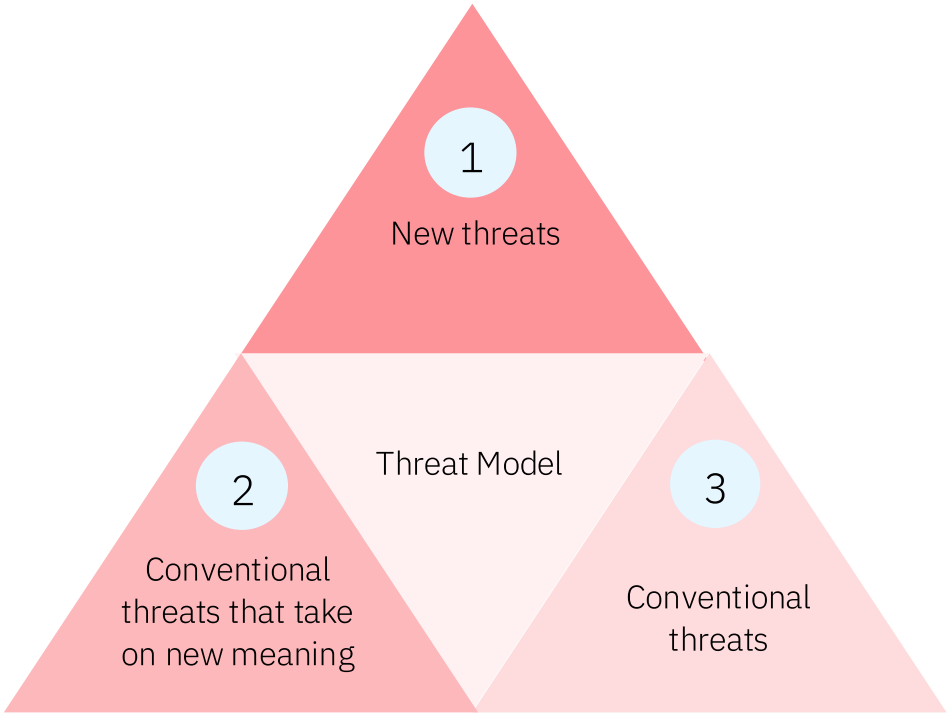


Figure 1: *Threats*

- **Tier 3 - Conventional Threats (19 Threats)**: security threats are are broadly applicable or derive from legacy, infrastructure, or transport implementation decisions

The table below organizes the threats by category and provides a mapping to controls and mitigations, discussed next.

Threat	Threat Category	MCP Specific	MCP Contextualized	Conventional Security	Control and Mitigation
MCP-T1	Improper Authentication and Identity Management	1. Identity Spoofing	8. Confused Deputy (OAuth Proxy)	16. Credential Theft/Token Theft 17. Replay Attacks/Session Hijacking 18. OAuth/Legacy Auth Weaknesses 19. Session Token Leakage	Agent Identity Secure Delegation (i.e. OAuth delegation)

Threat	Threat Category	MCP Specific	MCP Contextualized	Conventional Security	Control and Mitigation
MCP-T2	Missing or Improper Access Control		9. Insecure Human-in-the-Loop 10. Improper Multitenancy	8. Privilege Escalation 20. Excessive Permissions/Overexposure	Secure Delegation Access Control
MCP-T3	Input Validation/Sanitization Failures			21. Command Injection 22. File System Exposure/Path Traversal 23. Insufficient Integrity Checks	Data Sanitization Guardrails Sandboxing and Isolation (Roots support)
MCP-T4	Data/Control Boundary Distinction Failure	2. Tool Poisoning 3. Full Schema Poisoning 4. Resource Content Poisoning	11. Prompt Injection	21. Command Injection	Input Sanitization, Guardrails Context Isolation
MCP-T5	Inadequate Data Protection and Confidentiality Controls			24. Data Exfiltration & Corruption 22. File System Exposure/Path Traversal	Sandboxing and Isolation Access Control Guardrails
MCP-T6	Missing Integrity/Verification Controls	4. Resource Content Poisoning 5. Typosquatting/Confusion Attacks 6. Shadow MCP Servers		25. Supply Chain Compromise and Privileged host-base Attacks	Cryptographic Integrity Remote Attestation MCP server integrity

Threat	Threat Category	MCP Specific	MCP Contextualized	Conventional Security	Control and Mitigation
MCP-T7	Session and Transport Security Failures		12. Man-in-the-Middle (MITM)	26. Unrestricted Network Access 27. Protocol Security Gaps 28. Insecure Descriptor Handling 23. Insufficient Integrity Checks 29. CSRF Protection Missing 30. CORS/Origin Policy Bypass	Network and Transport Security
MCP-T8	Network Binding/Isolation Failures	6. Shadow MCP Servers	10. Improper Multitenancy	31. Malicious Command Execution 32. Dependency/Update Attack 26. Unrestricted Network Access	Network and Transport Security Sandboxing and Isolation
MCP-T9	Trust Boundary and Privilege Design Failures	7. Overreliance on the LLM	13. Consent/User Approval Fatigue		Secure tool design UX Design
MCP-T10	Resource Management/Rate Limiting Absence		14. Resource exhaustion and denial of wallet	33. Payload Limit/DoS	Network and Transport Security
MCP-T11	Supply Chain and Lifecycle Security Failures	6. Shadow MCP Servers		25. Supply Chain Compromise	Lifecycle Governance
MCP-T12	Insufficient Logging, Monitoring, and Auditability		15. Invisible Agent Activity	34. Lack of Observability	Logging Lifecycle Governance

3.1.1 MCP Specific

1. **Identity Spoofing** Weak or misconfigured authentication in MCP deployments could allow attackers to impersonate legitimate clients or the agents acting on their behalf, corrupting audit trails or gaining unauthorized access to server resources.
2. **Tool Poisoning** Malicious modification of tool metadata, configuration, or descriptors injected into clients via the tools/list method. This can cause AI agents or MCP components to invoke, trust, or execute compromised tools, potentially leading to data leaks or system compromise. As the MCP specification notes, 'descriptions of tool behavior such as annotations should be considered untrusted, unless obtained from a trusted server' (Key Principles), making tool poisoning a recognized risk when clients connect to unvetted servers.
3. **Full Schema Poisoning (FSP)** Attackers compromise entire tool schema definitions at the structural level, injecting hidden parameters, altered return types, or malicious default values that affect all subsequent tool invocations while maintaining apparent compatibility and evading detection by appearing legitimate to monitoring systems. Unlike Tool Poisoning (#2): Goes beyond poisoning individual tool metadata to compromise the entire structural definition and type system of tools.
4. **Resource Content Poisoning** Attackers embed hidden malicious instructions within data sources (databases, documents, API responses) that MCP servers retrieve and provide to LLMs, causing the poisoned content to execute as commands when processed, effectively achieving persistent prompt injection through trusted data channels rather than direct user input. Unlike Prompt Injection (#12): Malicious instructions are embedded in backend data sources, not user-provided prompts. Unlike Tool Poisoning (#2): Poisons the actual data/-content retrieved by tools, not the tool definitions themselves. This attack surface may be expanded with transitive or composed MCP server calls.
5. **Typosquatting/Confusion Attacks** Malicious actors create MCP servers or tools with names/descriptions similar to legitimate ones, tricking clients or AI agents into invoking harmful tools due to naming confusion or LLM hallucination. The MCP specification provides guidance on making tool origins and inputs visible to users and recommends human-in-the-loop approval for tool invocations (User Interaction Model), but consent fatigue—where users reflexively approve prompts without careful review—can significantly undermine these protections.
6. **Shadow MCP Servers** Unauthorized, unmonitored, or hidden MCP server instances create blind spots, increasing risk of undetected compromise and covert data exfiltration. These servers pose governance and compliance risks and may be malicious or easily compromised.

7. **Overreliance on the LLM** MCP server developers may implement overly permissive tools, assuming the LLM will invoke them correctly and safely. However, model-level controls (trained refusals, safety classifiers, etc.) are not ironclad—even capable models can be manipulated through prompt injection, make errors in judgment, or be replaced with weaker models that lack equivalent safeguards.

3.1.2 MCP Contextualized

8. **Privilege Escalation via Authentication and Authorization Bypass** Attackers exploit misconfigured roles, credentials, ACLs, trust relationships, or flawed delegation logic to gain elevated permissions and access unauthorized resources. In MCP deployments, this includes privilege escalation, as well as attacks that leverage the MCP server's intermediary role in multi-user token delegation. For example, confused deputy attacks can occur when an MCP server acting as an OAuth proxy fails to properly validate authorization context—allowing attackers to manipulate the server into using another user's credentials to perform privileged operations. See the official MCP guidance on preventing Confused Deputy attacks.
9. **Insecure Human-in-the-Loop** Missing or insufficient human-in-the-loop consent checks can allow an MCP server to take risky actions not authorized by the user.

10. **Improper Multitenancy** An attacker may exploit weak isolation between tenants or users, such as shared memory between processes, sessions, or secrets and credentials, to access or manipulate unauthorized data.
11. **Prompt Injection** LLMs have insufficient boundaries between input data and instructions. Attackers craft malicious inputs to manipulate LLMs or MCP components to perform unintended or harmful actions such as data exfiltration, privilege escalation, or unauthorized command execution. These malicious instructions can be sent *directly* to the LLM (e.g., via Sampling or when the MCP tool uses its own LLM) or *indirectly* by embedding instructions in prompts, resources, or tool metadata. This threat exists whenever untrusted input can reach the LLM's context window.
12. **Man-in-the-Middle (MITM)** Exploiting insecure network transport (lack of TLS, improper certificate validation, or missing mutual authentication) to intercept, modify, or reroute data between MCP components, enabling data theft or manipulation.
13. **Consent/User Approval Fatigue** Flooding users with excessive consent or permission prompts, causing habituation and leading to blind approval of potentially dangerous or malicious actions.
14. **Resource exhaustion and denial of wallet** Attackers trigger an excessive number of LLM, tool, or other API calls leading to unexpected costs or resource exhaustion and denial of service.
15. **Invisible Agent Activity** Agents or servers operate covertly, mimicking valid workflows but executing malicious or unauthorized actions without detection.

Conventional security threats to MCP, and definitions of the twelve threat categories, are discussed in the Appendix.

3.2 Controls and Mitigations

3.2.1 Agent Identity

All requests should be traceable across the entire execution chain: the end user or initiating agent, any intermediate MCP servers, and the tools or services that performed the resulting actions. Standards are emerging to define the identity of agents and servers. One of these is SPIFFE / SPIRE, which provides cryptographic workload identities that can be granted authorization to resources. The SPIFFE ID can be used in token exchange as the subject or actor depending on the flow.

Secure identity, authentication, and authorization across the agentic and MCP ecosystem is an extremely active area of research and development. We will provide a much deeper analysis of the problem space in a subsequent white paper.

3.2.2 Secure Delegation and Access Control

To mitigate against privilege escalation, MCP servers should operate with the minimum privileges necessary. OAuth provides a widely adopted framework for secure delegation, with extensions that support fine-grained scope control and secure token flows (see MCP Authorization).

- Leverage existing identity providers to provide user authentication using standards such as OIDC
- Register MCP server as clients with the IAM provider. If the registration cannot happen a priori, then use Dynamic Client Registration
- Do not passthrough the OAuth tokenss provided by the user
- Perform token exchange with the authorization server to provide full accountability (RFC8693)
- Reduce scopes for least privilege, such as removing write scopes when only read access is required (SEP-835 adds native support to define scopes in 2025-11-25 MCP specification)

- User short-liven tokens and support proof-of-possession (DPoP) to prevent replay attacks (RFC9449)
- Fine grained authorizations, through Rich Authorization Requests (RFC9396), limit requests to specific resources or tool parameters

All endpoint services should implement robust access control models, such as role-based access control (RBAC) or attribute-based access control and evaluate against claims made by the identity provider, such as role membership, job title, or work location. Additionally, robust policy languages including Open Policy Agent (OPA), Cedar, or OpenFGA provide robust, flexible, and secure protections.

3.2.3 Input and Data Sanitization and Filtering

A secure implementation of MCP requires strong data sanitization, input validation, and guardrails to protect against malicious or unsafe data inputs. Existing best practices for securing other RPC protocols should be applied.

All inputs should be strictly validated using allowlists at every trust boundary, with particular attention to sanitizing file paths through canonicalization, employing parameterized queries for database operations, and applying context-aware output encoding appropriate to each execution context (SQL, shell, HTML). Tool developers can include cryptographic checks, such as message authentication codes, digital signatures and encryption to ensure the end-to-end integrity and confidentiality of tools and resources.

LLM guardrails should treat all AI-generated content as untrusted input requiring the same rigorous validation as direct user input, deploying prompt injection detection systems that analyze patterns and structured formats (strict JSON schemas) to maintain clear boundaries between instructions and data. This includes all data returned from MCP servers including tool and resource definitions, resources, prompts, elicitation requests, and tool responses.

3.2.4 Cryptographic Integrity and Remote Attestation

Hardware Trusted Execution Environments (TEE) like Intel TDX, and AMD-SEV/SNP provide stronger isolation and can provide protection against runtime tampering of trusted servers, such as tool poisoning due to server compromise. Combined with remote attestation, TEEs can isolate MCP servers and clients from compromised hardware, malicious administrators of server infrastructure, and certain classes of co-tenancy threats. For containerized deployments, consider the use of confidential containers (CoCo) that run containers in TEEs, and use remote attestation to verify the trustworthiness of the TEEs and what is running in them.

Threat Category	How TEE + Remote Attestation provide Mitigation
MCP T5: Inadequate Data Protection and Confidentiality Controls	MCP Client and MCP Servers are isolated from other software running on the hardware, and also from the operator of the hardware, by running them in TEEs. Insider and privileged access to the TEEs is prevented there by minimizing the attack vectors for this threat. Good system design should include other controls for verifying the attestations, and delivery of secrets and sensitive data via secure channels to the MCP Clients and Servers running in attested TEEs. Compromised and/or Malicious co-tenant cannot access or tamper with Client and MCP Server code or data that is running inside TEEs. Additional controls would be necessary to ensure the data is protected in-transit and at-rest with tenant-specific keys, and RA-TLS, etc. With end to end data protection designs built on TEEs, Identity credentials, access tokens, keys and secrets can be protected from compromise and exposure. Compromised or Incorrect (or shadow) MCP Server launched in TEEs have different sets of measurements, and these will fail to attest, and MCP Client can refuse to interact with the MCP Servers. Credentials and access tokens will not be provided to the MCP Servers. TEEs however cannot mitigate against vulnerabilities in the running code, and should be complemented with runtime controls. (Seccomp, Apparmor, etc.)
MCP T9: Trust Boundary and Privilege Design Failures	Privileged software host-based attacks will not affect the MCP Client and the MCP Servers when they are running in TEEs. The host system, host OS, host firmware and the host Hypervisor are outside the Trust Boundary of the TEEs. However, sophisticated host-based attacks that include certain physical attacks on the hardware are not mitigated by TEEs.

Complement the use of TEEs with other sandboxing and isolation technologies.

3.2.5 Sandboxing and Isolation

Agents and MCP servers should be executed with least privilege. MCP servers that interact with the host environment (e.g. by accessing files, running commands, issuing network connections), or that execute LLM-generated code, should always run in a sandbox to mitigate against potential

safety and security threats.

LLM-generated code and commands may contain hallucinations, bugs, or vulnerabilities, and should not run with full user privileges. MCP servers are commonly deployed in containers for ease of use, but containers should not be relied upon as a strong security boundary. Consider additional sandboxing (gVisor, Kata Containers, SELinux sandboxes) for stronger isolation.

3.2.6 Cryptographic Verification of Resources

Organizations developing MCP servers must provide cryptographic signatures and software bill of materials (SBOMs) for all server code to verify provenance. Organizations deploying MCP clients and servers should obtain and verify the contents and cryptographic signatures prior to deployment, and have policies restricting the approved sources and signing keys. TLS should be used to protect all data in transit. Remote attestation can further verify that servers are running expected code in a trusted environment. When supported, end-to-end cryptographic signatures can prove the authenticity of resources returned by MCP servers.

3.2.7 Transport Layer Security

MCP is structured around distinct communication layers that facilitate robust interaction between systems. At the transport layer, MCP leverages two primary communication methods:

- **stdio Transport:** A direct, pipe-based stream communication channel, typically used for intra-process or tightly integrated inter-process communications. JSON-RPC messages flow directly via standard input/output streams. This transport is most commonly used for local servers.
- **HTTP Streaming Transport:** A generalized HTTP-based transport channel supporting bidirectional JSON-RPC communication via streamed request-response patterns. This transport is most commonly used for remote servers.

At the higher-level protocol layer, MCP employs JSON-RPC 2.0 to standardize the formatting and processing of commands and responses communicated across these transport channels. JSON-RPC ensures structured messaging, enabling interoperability and clarity of communication across diverse platforms.

However, these transport and protocol layers, when improperly secured or configured, can expose MCP clients and servers to multiple vulnerabilities. The following table summarizes critical missing security controls across MCP's layers and transports, along with specific exploits enabled by each gap:

RequiredSecurity Control	Protocol	Example Exploits
Payload Limits	All Transports	Large payload and recursive payload DoS
Client-Server Authentication/Authorization	HTTP-based Transports	Impersonation, pre-init commands, unauthorized RPC calls
Downstream Authentication/Authorization	All Transports	Impersonation, pre-init commands, unauthorized RPC calls
Mutual TLS Authentication	HTTP-based Transports	Impersonation attacks
TLS Encryption	HTTP-based Transports	Stream tampering, TLS downgrade
Cross-Origin (CORS)	HTTP-based Transports	Cross-origin data leaks
CSRF Protection	HTTP-based Transports	Forged POST requests

RequiredSecurity Control	Protocol	Example Exploits
Secure Descriptor Handling	stdio Transport	Hijacking via inherited descriptors
Integrity Checks	All Transports	Replay, spoofing, poisoned responses

Implementing the above controls across transport and protocol layers significantly reduces the attack surface of MCP deployments.

3.2.8 Secure Tool and UX Design

Tool and UX design represent a critical security control point in Model Context Protocol (MCP) deployments. While much attention is paid to model safety and prompt injection defenses, the tools that agents invoke are often the actual execution surface where security boundaries are crossed and sensitive operations are performed. Poor tool design can undermine even the most robust authentication and authorization controls by creating overly permissive capabilities or delegating security-critical decisions to the LLM itself.

Each tool should have a single, clearly defined purpose with explicit boundaries on what it can and cannot do. When possible, create use-case driven or purpose-built tools, avoiding excessively powerful tools, e.g., execute a prepared statement versus executing any SQL statement. Tool implementations should not rely on the LLM to perform security-critical operations, validate inputs, or enforce constraints.

Safe and secure execution should not rely solely on the human user, who may not understand the security implications of frequent security prompts and can easily become fatigued. Security-relevant messages and elicitations should be clear, indicating the implications of the request, and unambiguous what is being requested.

3.2.9 Human-in-the-loop

There is the possibility that a large language model, legit or poisoned, decides to execute a tool in a dangerous way. MCP hosts and clients, in general, allow users to disable the confirmation prompt. There are two approaches organizations considering this risk unacceptable may implement to reduce its probability and impact:

- enforce the use of MCP hosts and clients with a configuration that unprivileged users cannot change and that keeps the confirmation prompt enabled.
- use elicitation on the MCP server side to request the user confirmation of actions.

3.2.10 Logging

Implement at all layers (MCP host, client and server) the capability to store a log of what tools have been decided to use, with which parameters, and as a result of which prompt. Having a log of the decisions made is crucial in order to troubleshoot or perform forensics in case of a security event.

Leverage the use of centralization tools like MCP gateways or proxies, for example, between the MCP clients and the MCP servers, to centralize there key functionality (e.g. logging) and avoid the need to implement it on each component.

3.2.11 Lifecycle and Governance

Organizations must:

- implement mandatory code signing verification for all MCP servers before installation,
- use private package repositories with security scanning and approval workflows,
- deploy software composition analysis (SCA) tools to detect vulnerable dependencies,

- implement allow-lists of approved MCP servers with documented security reviews,
- Run MCP servers and clients in TEEs and use remote attestation to verify prior to interactions,
- use cryptographic hash verification for package integrity,
- and deploy binary authorization that prevents execution of unsigned or unverified code.

Supply chain security requires:

- implementing software bill of materials (SBOM) tracking for all MCP components,
- using dependency pinning with hash-based verification rather than version ranges,
- deploying automated vulnerability scanning for MCP servers and dependencies,
- implementing secure software development lifecycle (SSDLC) practices for internal MCP servers,
- using reproducible builds to verify package authenticity,
- and monitoring security advisories and CVE databases for known vulnerabilities in dependencies.

Lifecycle management demands:

- maintaining centralized inventory of all deployed MCP servers with metadata (version, owner, purpose),
- implementing automated discovery to detect shadow or unauthorized MCP deployments,
- deploying lifecycle policies that automatically deprecate or remove outdated servers,
- using configuration management tools (Ansible, Puppet, Chef) to maintain consistent deployments,
- implementing rollback capabilities for problematic updates,
- and deploying update management processes with testing and staged rollout.

Operational practices include:

- regular security reviews and re-certification of approved MCP servers,
- automated scanning for shadow deployments across the organization,
- decommissioning procedures that ensure complete removal of deprecated servers,
- and version tracking with forced upgrade policies for servers with known vulnerabilities.

And, lastly, proper observability should be implemented across the stack to provide sufficient visibility to ensure compliance and enable developer debugging and incident investigation. Immutable records of actions and authorizations, such as token exchange implemented by an IDP (identity provider), provides accountability pertaining to who requested an action and how it was authorized. All interactions with the agent, tools, prompts, and models should be logged. OpenTelemetry provides end-to-end linkability of actions and is being widely adopted and integrated into many agentic tools and MCP servers and provides a consistent set of APIs and schemas.

4. Conclusion

MCP adoption is accelerating, and security must keep pace. Our analysis reveals common vulnerabilities in deployments that lack adequate authentication, session management, and supply chain controls. Incidents in adjacent AI systems demonstrate these are active threats, not theoretical concerns.

Organizations deploying MCP-based systems must develop defense-in-depth strategies including zero-trust architectures, hardware-based isolation through trusted execution environments, rigorous supply chain vetting, and continuous monitoring. Securing MCP deployments requires coordinated effort across developers, organizations, and protocol maintainers—investment in security architecture now will pay dividends as agentic systems become more prevalent.

5. Contributors and Acknowledgements

Workstream Leads

- Sarah Novotny, (sarah.novotny@gmail.com)
- Ian Molloy, IBM (molloyim@us.ibm.com)
- Raghu Yeluri, Intel (raghuram.yeluri@intel.com)
- Alex Polyakov, Adversa AI (alex@adversa.ai)

Editors

- Daniel Rohrer, NVIDIA (drohrer@nvidia.com)
- Jenn Newton, Anthropic (jenn@anthropic.com)
- David LaBianca, Google (ddlb@google.com)

Contributors

- Shrey Bagga, Cisco (sbagga@cisco.com)
- Damian Bogel, Google (kele@google.com)
- Florencio Cano, Red Hat (fcanogab@redhat.com)
- John Cavanaugh, ProCap360 (johncavanaugh@procap360.com)
- Jason Clinton, Anthropic (j@anthropic.com)
- Andre Elizondo, Wiz (andre.elizondo@wiz.io)
- Riggs Goodman III, Amazon (goriggs@amazon.com)
- Hani Jamjoom, IBM (jamjoom@us.ibm.com)
- Chooi Low, Dell (Chooi.Low@dell.com)
- Victor Lu (victorjunlu@gmail.com)
- Michael Medeiros, Cisco (mimedeir@cisco.com)
- Grant Miller, IBM (millerg@us.ibm.com)
- David Pierce, PayPal (davpierce@paypal.com)
- Shriti Priya, IBM (shritip@ibm.com)
- Xiaokui Shu, IBM (xiaokui.shu@ibm.com)
- Bill Stout, ServiceNow (bill.stout@servicenow.com)
- Moin Syed, Anthropic (moin@anthropic.com)
- Josiah Hagen, TrendMicro (josiah_hagen@trendmicro.com)
- Jonathan Whitson (jonathan_whitson@dell.com)
- Marina Zeldin, Dell (marina.zeldin@dell.com)
- Giulio Zizzo, IBM (giulio.zizzo2@ibm.com)

Technical Steering Committee Co-Chairs

- Akila Srinivasan, Anthropic (akila@anthropic.com)
- J.R. Rao, IBM (jrrao@us.ibm.com)

6. Appendix

6.1 Deployment Pattern (DP) Security Considerations

The following section examines the common deployment patterns and their security implications in more detail.

6.1.1 Deployment Pattern 1: All-Local

MCP Client: localhost
MCP Server: localhost
Transport: stdio | http

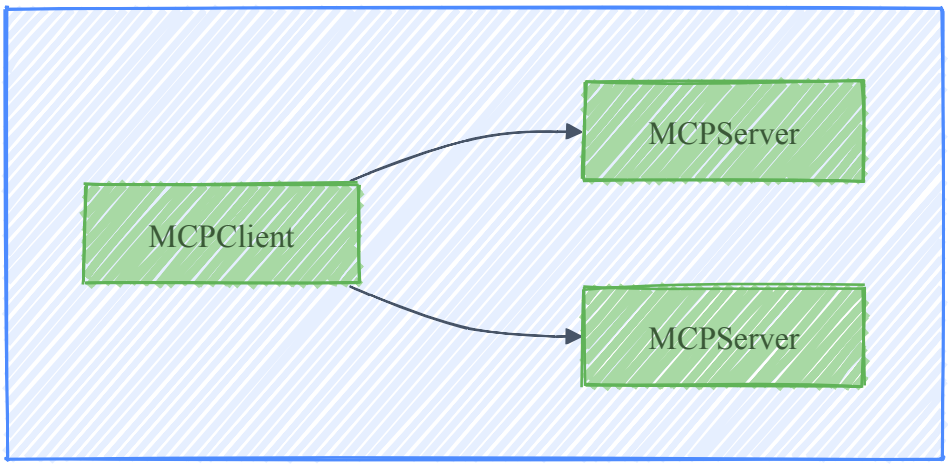


Figure 2: *DP1: All Local*

All-local deployment security is entirely dependent on the host system's posture and a general-purpose desktop or laptop with standard user software, internet connectivity, and physical access exposes the MCP server to the same attack vectors: malware execution, credential theft, supply chain compromise through installed packages, and physical device access.

With stdio transport, the MCP server runs as a subprocess of the host application, typically sharing the same user privileges and security context. This model trades network segmentation and centralized security controls for simplicity and direct access to local tools. It is well-suited for development workflows, trusted personal tools, and scenarios requiring direct local system access. However, for production deployments handling sensitive data or serving multiple users, the lack of isolation and dependence on host security may be insufficient depending on organizational risk tolerance and compliance requirements.

Security Implications:

- Exposes local system to potentially malicious or compromised servers
- Execution control: Inherits host's security posture
- Data access control: inherits host's data posture

Security Recommendations:

- Appropriate for development and personal use
- Use stdio to avoid DNS rebinding risks: stdio transport is strongly recommended for local MCP as this eliminates DNS rebinding risks that can occur with HTTP-based transports on local servers
- Use sandboxing to limit privilege escalation attacks: MCP servers locally requires a sandbox to prevent privilege escalation attacks

6.1.2 Deployment Pattern 2: Single-Tenant MCP Server

MCP Client: localhost
MCP Server: single-tenant remote host
Transport: http

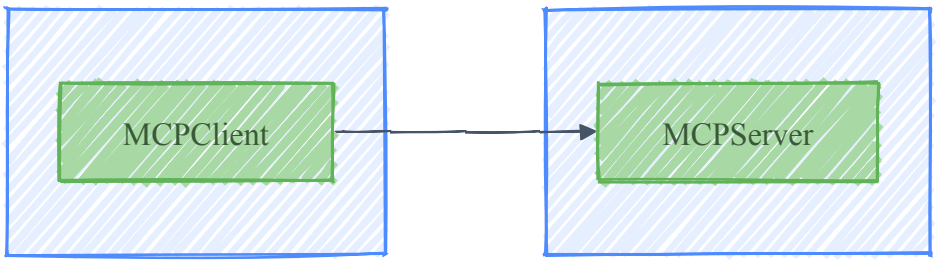


Figure 3: *DP2: Single-Tenant*

MCP deployment model where the client runs locally but connects to a single-tenant MCP server in the cloud. Provides users with local tools (file access, development utilities) and remote capabilities (API access, shared resources, centralized data).

Note: Authentication between client and server **is required** to establish the trust boundary.

Security Implications:

- Security of the remote MCP server depends on cloud infrastructure controls rather than local host posture
- Client security has similar implications toDeployment Pattern1 as it is running locally

Security Recommendations:

- Secure Credential Storage: clients should use secure credential storage (OS keychains, secret managers) to protect MCP server authentication tokens
- Authenticated and Encrypted: communication between local and cloud components must be authenticated and encrypted
- Secure Servers and Discovery: enterprise clients should enforce authenticated server discovery and maintain explicit allowlists (ex. via MDM)

6.1.3 Deployment Pattern 4: Multi-Tenant MCP Server

MCP Client: cloud-hosted or locally-hosted
MCP Server: PaaS or SaaS provided
Transport: http

An MCP deployment model where a service provider runs an MCP server and provides access to multiple tenants. The MCP server could serve its own tools, prompts, and resources or provide an MCP tool interface to an existing service, API, or application.

Security Implications:

- Requires robust tenant isolation, identity, and access control
- Improper isolation may lead to leakage of sensitive data or privilege escalation

Security Recommendations:

- MCP server developers must implement strong multi-tenancy controls (e.g., per-tenant encryption, RBAC).
- Select MCP servers hosted directly by the service provider (e.g., use GitHub's MCP server for GitHub access instead of a third-party server)
- Provide remote attestation for MCP servers when possible

6.2 Threat Details

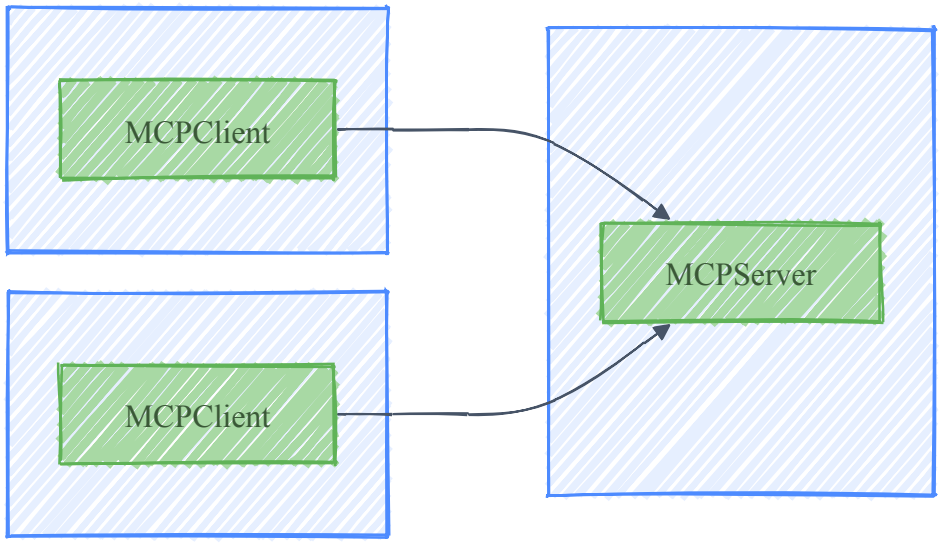


Figure 4: *DP3: Multi-Tenant Server*

MCP-T1: Improper Authentication and Identity Management

Technical Description: Absence of authentication mechanisms, insecure credential storage practices, and inadequate identity verification within MCP implementations. The protocol's optional authentication model combined with the common practice of storing multiple service credentials (OAuth tokens, API keys, database passwords) in centralized MCP servers creates high-value targets. MCP servers may accumulate credentials without cryptographic protection, secure storage standards, or credential rotation policies. See the official MCP documentation for additional guidance.

Architectural Impact: Improper agent identity and authentication leads to impersonation and replay attacks, preventing the MCP server and endpoints from correctly identifying the identity of the originating request, and a confused deputy. The insecure storage of authentication credentials across users and services fundamentally alters the security posture. Weak or absent authentication enables unauthorized server access, credential harvesting, token theft, and persistent access.

Vulnerability Examples: Credential exposure in configuration files, OAuth token theft, authentication bypass, lack of multi-factor authentication, insecure credential storage, static client ID vulnerabilities, authentication mechanism implementation flaws

MCP-T2: Missing or Improper Access Control

Technical Description: Absence of authorization mechanisms, improper enforcement of object-level permissions, and insufficient capability-based access control within the MCP specification. The protocol lacks native support for fine-grained authorization checks, role-based access control (RBAC), and privilege separation. MCP servers commonly request and receive overly broad permission scopes to maximize flexibility, while implementations fail to verify user permissions for individual objects, resources, or operations.

Architectural Impact: Enables unauthorized data access, privilege escalation, and lateral movement across connected services. The optional nature of authorization combined with developers' tendency to grant excessive permissions leads to inconsistent security postures

Vulnerability Examples: Broken Object Level Authorization (BOLA), privilege abuse, overbroad

permissions, insufficient authorization checks, context bleeding, cross-tenant data exposure, function-level access control failures, configuration file exposure.

MCP-T3: Input Validation/Sanitization Failures

Technical Description: Absence of input validation, sanitization, and parameterization across multiple injection contexts including command execution, database queries, file system operations, and serialization boundaries. This vulnerability class encompasses traditional injection flaws exacerbated by the false sense of security provided by AI mediation. Developers incorrectly assume that user input processed through an LLM is inherently safe, bypassing established secure coding practices, when in reality the LLM transforms but doesn't sanitize malicious payloads.

Architectural Impact: Enables command injection, SQL injection, LDAP injection, XML injection, path traversal, and deserialization attacks with potentially catastrophic consequences. Can be combined with other threats (MCP-T4) for increased impact.

Vulnerability Examples: Command injection, SQL injection, remote code execution (RCE), path traversal, LDAP injection, XML injection, deserialization vulnerabilities, unsafe file operations

MCP-T4: Input/Instruction Boundary Distinction Failure

Technical Description: This design limitation enables the entire class of prompt injection vulnerabilities, including direct injection, indirect injection through tool / schema descriptions, and context manipulation attacks. LLMs lack syntactic or semantic mechanisms to differentiate between control instructions and data payloads. This fundamental limitation stems from the continuous token stream processing model, where both trusted system prompts and untrusted user data are processed within the same computational context without clear demarcation. The absence of a control plane/data plane separation at the architectural level means any adversary-controllable input—including tool responses, schema descriptions, and resource content—can potentially alter the execution flow of the AI system.

Architectural Impact: Attackers can embed malicious instructions in seemingly benign content (emails, documents, API responses) that, when processed by the LLM, execute unauthorized actions. The blurring of boundaries between viewing content and executing commands creates attack vectors where reading a document can trigger data exfiltration, system compromise, or unauthorized API calls through MCP tools.

Vulnerability Examples: Prompt injection (direct and indirect), tool poisoning (TPA), full schema poisoning (FSP), advanced tool poisoning (ATPA), resource content poisoning, hidden prompt embedding (Unicode attacks), malicious message crafting

MCP-T5: Inadequate Data Protection and Confidentiality Controls

Technical Description: Insufficient encryption, inadequate secrets management, and absence of data protection standards for sensitive information in transit, at rest, and during processing. MCP implementations commonly expose personally identifiable information (PII), financial data, and intellectual property without proper encryption, data classification, or access segmentation.

Architectural Impact: Creates significant data exposure risks where attackers gaining partial access can perform correlation attacks across services to build comprehensive user profiles, enabling sophisticated spear-phishing, extortion, or identity theft. The concentration of access to disparate services in a single protocol layer violates the security principle of segregation, allowing data leakage through logging, error messages, debug output, and traffic inspection. Without encryption and proper secrets management, sensitive data remains vulnerable throughout its lifecycle.

Vulnerability Examples: Unencrypted credential storage, sensitive data exposure in logs, PII leakage

MCP-T6: Missing Integrity/Verification Controls

Technical Description: Absence of cryptographic integrity verification mechanisms for MCP servers, clients, tool definitions, message authenticity, configuration immutability, and behavioral attestation. The protocol lacks provisions for code signing, integrity verification, message authentication codes, reproducible builds, and tamper-evident logging, enabling undetected modification of critical system components. Permits tool behavior mutation, configuration tampering, message forgery, launch and execution of compromised/shadow MCP servers and clients and supply chain attacks.

Architectural Impact: Without integrity verification, malicious actors can modify tool definitions post-deployment, alter configuration files after approval, and inject malicious updates without detection

Vulnerability Examples: Rug Pull Attack, Tool Shadowing, Tool Name Spoofing, MCP Configuration Poisoning

MCP-T7: Session and Transport Security Failures

Technical Description: Systematic weaknesses in session lifecycle management and transport security including insecure session identifier transmission, absence of session binding mechanisms, lack of secure session storage standards, insufficient transport encryption enforcement, and inadequate session termination controls.

Architectural Impact: Enables session hijacking, replay attacks, session fixation, man-in-the-middle attacks, and cross-site request forgery.

Vulnerability Examples: Session management flaws, session hijacking, replay attacks, insufficient timeout policies, man-in-the-middle attacks, insecure transport protocols

MCP-T8: Network Binding/Isolation Failures

Technical Description: Architectural deficiencies in network isolation including improper network interface binding, absence of network segmentation requirements, vulnerability to DNS rebinding attacks, and lack of defense-in-depth network controls. MCP implementations commonly bind to all available interfaces (0.0.0.0) rather than localhost, exposing internal services to external networks. The protocol lacks specifications for network-level security boundaries, proper CORS policies, and protection against cross-origin attacks.

Architectural Impact: Exposes internal services to external networks, enables lateral movement within compromised environments, and permits DNS rebinding attacks that bypass same-origin policies.

Vulnerability Examples: Localhost bypass (NeighborJack), DNS rebinding attacks, improper network interface binding, insufficient network segmentation, cross-origin vulnerabilities, exposure of internal services, lateral movement exploitation

MCP-T9: Trust Boundary and Privilege Design Failures

Technical Description: Fundamental architectural flaw wherein the protocol assumes an optimistic trust model between the MCP client and server or an MCP server and third-party API or service.

Architectural Impact: Creates cascading security failures where compromise of a single component leads to complete system breach. Enables privilege escalation, lateral movement across service boundaries, and complex attack chains exploiting transitive trust relationships.

Vulnerability Examples: Overreliance on an LLM, confused deputy, cross-tenant exposure, overreliance on human-in-the-loop

MCP-T10: Resource Management/Rate Limiting Absence

Technical Description: Lack of resource consumption controls, quota management systems, and economic attack prevention mechanisms. The protocol provides no specifications for token limits, context size boundaries, API call quotas, computational resource allocation, or cost management, enabling resource exhaustion and economic denial-of-service attacks

Architectural Impact: Facilitates denial-of-service through token exhaustion, context window overflow, and API quota depletion. Enables economic attacks where minimal attacker investment causes disproportionate financial damage through excessive LLM API consumption. The absence of resource controls can lead to unexpected costs, high latency, and denial of service.

Vulnerability Examples: Denial of Wallet, Denial of Service , MCP Resource Exhaustion, MCP Recursive Task Exhaustion, Large Context Payload DoS

MCP-T11: Supply Chain and Lifecycle Security Failures

Technical Description: Absence of secure software supply chain practices for MCP server acquisition, installation, updates, and lifecycle management. The protocol lacks standardized mechanisms for verifying server provenance, validating package integrity before installation, or maintaining inventory of deployed MCP servers across an organization. MCP servers are commonly downloaded from third-party repositories without cryptographic verification, installed without security review, and remain operational indefinitely without lifecycle management. Organizations face risks from shadow MCP servers deployed without authorization, zombie servers that remain active after deprecation, and malicious packages masquerading as legitimate tools in public repositories.

Architectural Impact: Creates pre-deployment and operational security gaps distinct from runtime integrity issues. Attackers can distribute malicious MCP servers through popular repositories, compromise legitimate packages during distribution, or exploit the absence of inventory management to deploy unauthorized servers. Shadow deployments bypass security controls entirely, and the absence of centralized inventory prevents detection of unauthorized MCP infrastructure across the enterprise.

Vulnerability Examples: Malicious MCP package distribution, compromised package repositories, shadow server deployments, zombie/deprecated servers, typosquatting in package names, package substitution attacks, unsigned server distributions, unvetted community packages, lack of security review processes

MCP-T12: Insufficient Logging, Monitoring, and Auditability

Technical Description: Absence of standardized audit logging, comprehensive traceability mechanisms, and security monitoring capabilities within MCP implementations. Without robust logging of MCP server connections, tool invocations, authorization decisions, and data access patterns, organizations face significant compliance blind spots and inability to perform forensic analysis of security incidents.

Architectural Impact: Severely impairs incident detection, response capabilities, and post-incident forensics. Organizations cannot trace AI agent actions back to their source, establish accountability for security breaches, or identify patterns indicating compromise or abuse.Compliance frameworks requiring audit trails and accountability mechanisms cannot be satisfied, creating regulatory risks and limiting the ability to meet security certification requirements.

Vulnerability Examples: Insufficient audit logging, lack of security telemetry, inability to trace attack chains, missing forensic capabilities, inadequate anomaly detection, undetected shadow servers, compliance violations, absence of accountability mechanisms, blind spots in security visibility

6.3 MCP Threats and Vulnerabilities

6.3.1 Conventional Security

16. **Credential Theft /Token Theft** Attackers exploit insecure storage, handling, or transmission of secrets (OAuth tokens, API keys, credentials), enabling impersonation, unauthorized access, or privilege escalation.
17. **Replay Attacks/Session Hijacking** Attackers intercept, reuse, or hijack authentication tokens or session identifiers, impersonating legitimate users or agents and executing unauthorized actions.
18. **OAuth/Legacy Auth Weaknesses** Use of outdated, weak, or pass-through authentication and authorization (e.g., basic auth, static API keys) exposes systems to impersonation, privilege misuse, and poor accountability.
19. **Session Token Leakage** Exposure or insecure handling of session tokens across MCP components leads to unauthorized access, impersonation, or session hijacking.
20. **Excessive Permissions/Overexposure** AI agents, MCP servers, or tools are granted more privileges than necessary, increasing risk of abuse or compromise in case of attack or misconfiguration.
21. **Command Injection** Unvalidated or unsanitized user inputs, prompts, or tool arguments lead to execution of unauthorized system commands, resulting in data compromise or system takeover.
22. **File System Exposure/Path Traversal** Improper validation of file paths or tool arguments enables access to or exfiltration of files outside intended directories, exposing credentials and sensitive data.
23. **Insufficient Integrity Checks** Absence of signature or integrity validation on MCP messages and responses enables replay, spoofing, or delivery of poisoned results.
24. **Data Exfiltration & Corruption** Attackers leverage MCP components to steal or corrupt sensitive data, reroute messages, or manipulate outputs, often via compromised servers or poisoned tools.
25. **Supply Chain Compromise** Malicious or compromised MCP servers, dependencies, or packages are introduced into the environment, enabling attackers to execute arbitrary code, exfiltrate data, or persist within the infrastructure.
26. **Unrestricted Network Access** MCP servers or clients with open outbound or inbound network access can download malicious payloads, exfiltrate data, or connect to command-and-control infrastructure. Malicious or compromised MCP servers allow attackers to move laterally using stored credentials and exploiting poor network segmentation and isolation.
27. **Protocol Security Gaps** Weaknesses in MCP protocol/transport layers (e.g., missing payload limits, no TLS, unauthenticated requests) enable DoS, spoofing, or unauthorized command execution.
28. **Insecure Descriptor Handling** Improper management of transport descriptors (e.g., stdio) allows attackers to hijack or interfere with data streams and process communications.
29. **CSRF Protection Missing** Lack of Cross-Site Request Forgery (CSRF) controls on HTTP/SSE transports enables attackers to forge or replay unauthorized requests.
30. **CORS/Origin Policy Bypass** Missing or weak cross-origin policies allow unauthorized data leaks via cross-origin resource sharing (CORS) in browser-based or web transports.
31. **Malicious Command Execution** Compromised or rogue MCP servers execute arbitrary or malicious payloads (ransomware, data manipulation) triggered by crafted prompts or files.
32. **Dependency/Update Attack** Attackers compromise MCP dependencies or update channels (e.g., "rug pull" attacks), swapping benign code for malicious versions after trust is established. MCP servers may also introduce new capabilities (e.g., tools or prompts) that have not been vetted or approved for use.
33. **Payload Limit/DoS** Unrestricted payload sizes or recursion depth in protocols enable denial-of-service via resource exhaustion.
34. **Lack of Observability** Insufficient logging, monitoring, or attribution across MCP actions

hides malicious or unintended activity, hindering detection and response.

6.4 CoSAI Focus

CoSAI is an OASIS Open Project, bringing together an open ecosystem of AI and security experts from industry-leading organizations. The project is dedicated to sharing best practices for secure AI deployment and collaborating on AI security research and product development. The scope of CoSAI is specifically focused on the secure building, integration, deployment, and operation of AI systems, with an emphasis on mitigating security risks unique to AI technologies. Other aspects of Trustworthy AI are deemed important but beyond the scope of the project including, ethics, fairness, explainability, bias detection, safety, consumer privacy, misinformation, hallucinations, deep fakes, or content safety concerns like hateful or abusive content, malware, or phishing generation. By concentrating on developing robust measures, best practices, and guidelines to safeguard AI systems against unauthorized access, tampering, or misuse, CoSAI aims to contribute to the responsible development and deployment of resilient, secure AI technologies.

6.5 Guidelines on usage of more advanced AI systems (e.g. large language models (LLMs), multi-modal language models. etc) for drafting documents for OASIS CoSAI:

tl;dr: CoSAI contributions are actions performed by humans, who are responsible for the content of those contributions, based on their signed OASIS iCLA (and eCLA, if applicable). [Each contributor must confirm whether they are entitled to donate that material under the applicable open source license; OASIS and the CoSAI Project do not separately confirm that.] Each contributor is responsible for ensuring that all contributions comply with these AI use guidelines, including disclosure of any use of AI in contributions.

- Selection of AI systems: CoSAI recommends the use of reputable AI systems (lowering the risk of inadvertently incorporating infringing material).
- Model constraints: Currently, CoSAI or OASIS are not required to have a contract or financial agreement for using AI systems from specific vendors. However, CoSAI editors should consider employing varying tools to avoid potential fairness concerns among vendors.
- IP infringement: It is the responsibility of the individual who subscribes/prompts and receives a response from an AI system to confirm they have the right to repost and donate the content to OASIS under our rules.
- Transparency: CoSAI's goal will be to maintain transparency throughout the process by documenting substantial use of AI systems whenever possible (e.g., the prompts and the AI system used), and to ensure that all content, regardless of production by human or AI systems, was reviewed and edited by human experts. This helps build trust in the standards development process and ensures accountability.
- Human-edited content and quality control: CoSAI mandates human-reviewed or -edited results for any final outputs. A robust quality control process should be in place, involving careful review of the generated content for accuracy, relevance, and alignment with CoSAI's goals and principles. Human experts should scrutinize the output of AI systems to identify any errors, inconsistencies, or potential biases.
- Iterative refinement: The use of AI systems in drafting standards should be seen as an iterative process, with the generated content serving as a starting point for further refinement and improvement by human experts. Multiple rounds of review and editing may be necessary to ensure the final standards meet the required quality and reliability thresholds.

6.6 Copyright Notice

Copyright © OASIS Open 2025. All Rights Reserved. This document has been produced under the process and license terms stated in the OASIS Open Project rules: https://www.oasis-open.org/policies-guidelines/open-projects-process.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns. This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OASIS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS DOCUMENT OR ANY PART THEREOF. The name "OASIS" is a trademark of OASIS, the owner and developer of this document, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, documents, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark/ for above guidance.

This is a Non-Standards Track Work Product. The patent provisions of the OASIS IPR Policy do not apply.