Model Context Protocol (MCP): Security Design Considerations for AI-Driven Automation

Securing protocols that enable AI-driven analysis and automation.

Executive summary

Model Context Protocol (MCP) has become the de facto standard for enabling communication across a growing ecosystem of artificial intelligence (AI) driven services. Its purpose is to facilitate the automated sharing of data and execution of tasks across AI-driven ecosystems and workflows, including model evaluation, enrichment, pre-processing, and task automation. As adoption accelerates, MCP is now found in both experimental and production AI deployments across multiple industries, such as business, finance, legal, and software development. AutoGen Studio, Harvey AI, Agentverse, and Copilot are just a few examples of products deploying MCP that are used throughout these industries.

MCP's rapid proliferation has outpaced the development of its security model. Much like early web protocols, MCP was released with a flexible and underspecified design, allowing implementers freedom of design but also introducing ambiguity for safe usage. Critically, the protocol reverses a familiar interaction pattern: instead of clients requesting data from servers, MCP often expects servers to query and sometimes execute actions for the connected clients. This inversion creates new and largely not well-traced attack paths. These theoretical risks have now been made concrete: public labs and security researchers released vulnerable MCP server implementations to illustrate how easily insufficiently secured MCP can be exploited, providing examples that are not merely hypothetical but demonstrable. [1]

Some MCP implementations allow malicious actor-controlled inputs to reach execution environments without appropriate constraints, creating vulnerabilities generally considered high severity. One such class of vulnerabilities is Arbitrary Code Execution (ACE), which can easily arise in MCP environments where user provided logic or code is executed and typically tracked under multiple Common Weakness Enumeration (CWE) categories (CWE-77, CWE-78, CWE-94, CWE-95) for vulnerability management. These design tendencies suggest that secure-by-default behavior must

be enforced through implementation rigor, proper coding practices, clearer protocol specifications, and robust validation tools.

This report outlines observed security concerns and patterns from real-world deployments and offers practical recommendations for organizations adopting MCP in high-stakes or production environments. By doing so, it aims to reduce risk while supporting safe innovation in AI-augmented systems. While this guidance is based on the MCP specification, implementations, and currently known issues, it is designed to be extensible and remain relevant as the protocol, implementations, and operations continue to evolve.

Introduction

Model Context Protocol (MCP) is an application-level protocol that provides a simple and agreed upon messaging pattern and transport format currently used by many AI-enabled systems for managing interactions between services. Originally popularized as an open source framework by Anthropic in November 2024 [2], MCP has since gained traction in a wide array of AI orchestration and agent frameworks. Its purpose is to pass structured messages, usually containing a role, message content, and an optional tool or context, between components in a multi-step workflow.

To understand MCP's promise, and risks, consider a common real-world use case: travel planning. Imagine asking an AI assistant to plan an international trip. With a single prompt of a few words, it determines current location, local time, and the nearest international airport, and then retrieves visa requirements, recommends flights, and compiles a complete itinerary. Behind the scenes, this assistant relies on MCP to quickly and flexibly orchestrate these data gathering tasks across multiple tools and services.

Figure 1 illustrates how an agentic AI travel assistant might use MCP to automate each step of that process. In such a deployment, the User App/User Interface (UI) acts as the primary entry point for human interaction, collecting input, and displaying results. This input is passed to the large language model (LLM), which interprets the user's intent and formulates structured requests for execution. The MCP Client Library functions as the middleware layer, translating these LLM-generated requests into MCP protocol messages and managing the connection lifecycle. These messages are then routed to an MCP Server, which acts as the orchestrator, matching incoming requests to the

appropriate registered capabilities or tools. Finally, individual Tools (A, B, C) perform the requested tasks, ranging from data retrieval to computational processing, and return results up the chain, ultimately reaching the User App/UI.

While MCP simplifies the integration of such diverse capabilities into powerful agent workflows, the current protocol specification falls short on key security and privacy protections. As real-world adoption accelerates, especially for high context, sensitive tasks like querying electronic health records (EHRs), gaps in design, implementation, and operational posture are becoming more apparent. This report examines those risks and outlines gaps that must be addressed before MCP can be safely and confidently used in security-critical environments.

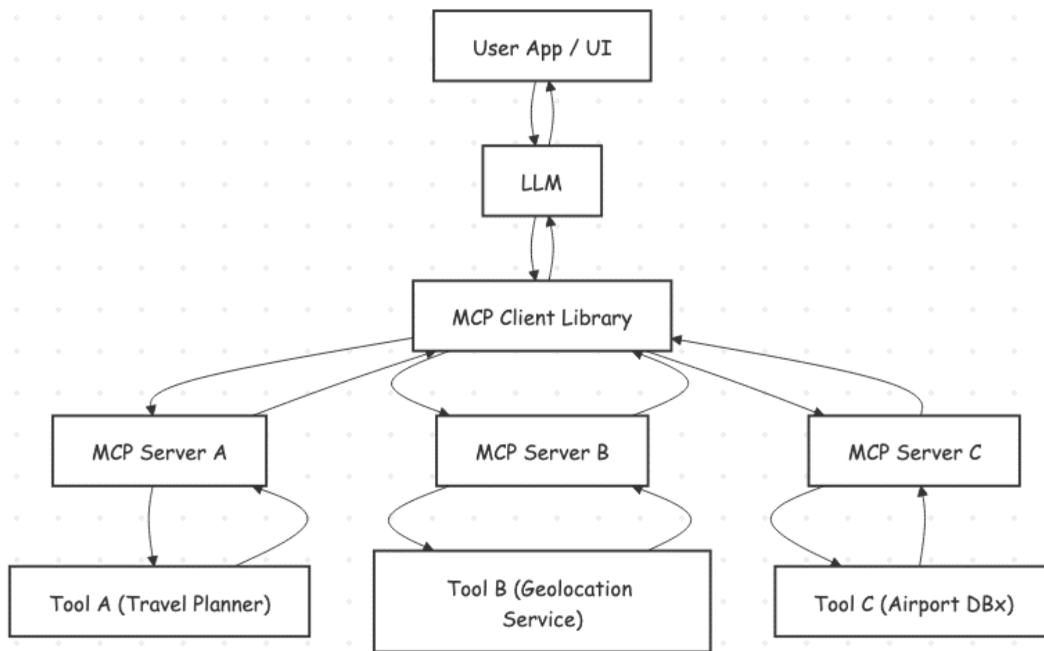

Figure 1: Diagram of how different MCP components interact

Security concerns in MCP design, implementation, and practices

Traditional cybersecurity principles, such as authentication, authorization, and input validation, remain as necessary protective measures in environments that deploy MCP. However, agentic AI systems, especially those in which MCP is featured, introduce novel and systemic risks that established cyber defense strategies do not adequately address. Refer to Careful Adoption of Agentic AI Services for more information about

securing agentic AI systems generally. [3] For MCP systems, the risks include dynamic tool invocation (where agents can autonomously call new tools at runtime), implicit trust relationships (where one agent's output is assumed valid by another without explicit verification), and context sharing (where long lived or overlapping context windows can leak, blend, or misalign across tasks).

These should not be viewed as isolated problems that can be patched at the interface or endpoint level. Rather, they require reworking of the entire lifecycle from the design of the protocol and agent behaviors, to runtime scheduling and coordination, to integration with external services and long-term operational monitoring. Securing MCP systems requires treating the agentic environment as a continuum, where misaligned assumptions or subtle inconsistencies at any stage can propagate and compound into exploitable conditions.

Access control

Associating a session to an identity is not defined by the protocol, but optionally achieved at the discretion of the implementor. MCP components—including clients, servers, proxies, and hosts—can be configured to access or process data without any required access control measures. Many implementations omit authentication entirely, and those that do include it often lack any role-based enforcement, such as distinguishing Create, Read, Update, and Delete (CRUD) permissions that are designed and implemented in database and data interfaces. This creates ambiguous and untraceable access to data and context.

Furthermore, multiple MCP servers, while not directly connected with each other, can be exposed to messages or information that is being shared freely from the MCP client. This practice increases the likelihood of data leakage or unverified task propagation.[1] MCP currently lacks support for exchanging Role Based Access Control (RBAC) permissions at instantiation, making it difficult to enforce or verify access boundaries between tasks or services.

Insecure context or data serialization

MCP enables the transmission of structured objects, such as context, configuration, and payloads, through serialization mechanisms that often lack strict validation or schema

[1] Unverified task propagation refers to situations where tasks are passed between MCP servers or agentic components without proper validation of their origin, scope, or intent. This kind of propagation can lead to overreach, leakage of sensitive context, or unintentional activation of unrelated downstream tools.

enforcement. Instead, these are left open ended for the implementors to achieve in their own way. Serialized content including comments or prompts may open a path for injection techniques because it can include executable code or embedded model calls. This is especially problematic if deserialization occurs in a permissive way without strong isolation between code and other text (e.g., data and comments). This problem echoes well-known serialization risks (e.g., OWASP A08/2017 [4]), with some newer implications for the application-level protocol that could embed content for natural language parsing.

Poor approval workflows

Although the MCP documentation provides some recommended security guidelines for implementers, the documentation states, "MCP itself cannot enforce these security principles at the protocol level." [5] While many implementations of MCP require end user consent to give access to an account (e.g., allowing the MCP GitHub server access to a user's GitHub account) in addition to requiring authorization of an action through positive affirmation (e.g., clicking "yes" or "allow" on a pop-up message), some implementations do not support codifying such approvals. [6] Even when workflow approval is implemented, a change in capability or data access for an MCP server that is already trusted or connected often can be made without approval. End users are often kept unaware of these changes, whether malicious or benevolent, and are not prompted for informed consent to accept the changes. This concern is closely related to access control issues. In an established MCP-connected AI system, a change in the behavior of an MCP server may go unnoticed as the server can perform continual small improvements in the background. As a result, a previously benign and approved AI service could later access sensitive resources on demand, without triggering any review.

Token or session security

MCP implementations frequently (and often only optionally) rely on OAuth-style bearer tokens or session IDs for access authorization [7], but as stated in their standard documentation:

- Authorization in MCP is optional (not every implementation uses it)

- MCP servers rely on bearer tokens, as defined in OAuth 2.1, without specifying protocol-level token lifecycle management (i.e., refresh, revocation, and reuse control).
- Expiration and rotation are recommended as best practices in some drafts, but the core MCP specification does not mandate any requirement for lifecycle management.

These gaps can lead to message replay or unauthorized reuse of valid sessions. In the event of session hijacking, a malicious actor can impersonate a legitimate client, inject malicious prompts, or interact with MCP servers without detection. Known weaknesses in token passthrough and lifecycle handling have been noted in the specification itself (MCP Security Best Practices [8], Lifecycle Guidelines [9]), but many details are left unspecified, allowing for insecure implementations. [10] As an example, idempotency—the ability to execute an operation multiple times without changing the result beyond the initial application—is not directly enforced by MCP. Instead, it is left to the underlying JavaScript Object Notation – Remote Procedure Call (JSON-RPC) and message queue specifications. This means the responsibility for ensuring idempotent behavior falls to the client protocol and often may not be implemented properly or overlooked entirely.

Misconfigurations and poor implementation

Given the availability of open source MCP frameworks and repositories, developers can rapidly deploy MCP servers without fully understanding the security implications. For example, a trivial implementation, such as one designed to provide weather related information for a location, might be deployed without restrictions and later repurposed to access or process sensitive data (e.g., user or asset information at the location). As another example, the same model interface and message structure may be reused across tasks, but MCP servers often lack task or data isolation, creating opportunities for inadvertent data exposure. MCP, in its current form, does not provide ways to isolate these services to support proper privacy and security.

Inconsistent behaviors

MCP implementations vary widely in how they interpret or respond to serialized context, messages, or server tasks. This inconsistency in behavior across deployments can introduce unpredictable and exploitable gaps, particularly when assumptions made by one component are violated by another. Even simple differences in context shaping or

timing of responses can lead to divergent outcomes, security blind spots, or denial of safeguards. For example, an agentic AI designed to assist with travel may interpret a user prompt differently depending on context history. One AI agent might link the request to an earlier email about vacation plans whereas another may treat it as a new query. This divergence, driven by probabilistic interpretation of prior context, can be exploited by a malicious actor who preconditions the agent to arrive at a specific or unsafe outcome. These non-deterministic behavior issues represent a relatively new challenge—often outside the scope of traditional information security practices—that will need to be considered to adopt newer AI technologies securely.

Poor or missing audit logs

While the MCP specification includes basic guidance for logging, it leaves the implementors to define and implement comprehensive audit requirements. As a result, many implementations either omit logging entirely or record only minimal operational metadata. Robust audit logging and alerting are essential in an environment as consequential as an MCP server. Logs should capture a traceable sequence of actions across sessions, include source information from hypertext transfer protocol (HTTP) headers where available, and provide native support for detecting protocol anomalies, such as repeated or malformed requests, authorization failures, or RBAC violations. Without these traceable audit logs, incident response and accountability become significantly more difficult, especially in multi-tenant environments.

Denial of service and fatigue-based techniques

MCP servers, particularly those serving agent-like functions, can be susceptible to prompt storms, malformed inputs, or recursive task requests that lead to denial-of-service (DoS) conditions. Some adversaries exploit "lethargy" by exhausting server resources through legitimate but overly complex tasks, causing delays and interruptions in the broader system workflow. These techniques can be difficult to distinguish from valid high load conditions, further complicating detection. While these are not unique to MCP supported agents, MCP provides an open door for such resource exhaustion techniques if not properly managed. [11]

Examples of real-world MCP security issues

As MCP implementations are emerging, some of the critical design and operational issues are being exposed in real-world deployments. The following examples of problematic MCP implementations reveal how unchecked assumptions, implicit trust boundaries, and integration shortcuts can lead to security problems. In many cases, malicious actors can manipulate legitimate functionality in unexpected or harmful ways. The lack of security further degrades trust in MCP. The following are examples of MCP-related security risks that have been identified and/or exploited in the real world.

Tool parameter injection in open MCP agents

Open source MCP agents were observed exposing sensitive MCP server data after unsanitized tool parameters, passed via malformed MCP messages, were executed using simple tools. The lack of contextual validation and logging allowed malicious actors to use legitimate interfaces to run arbitrary commands. [12]

Tool invocation path confusion

Some MCP orchestrators automatically resolve tool names from public registries or local modules. This allowed malicious actors to trick the system into loading actor-controlled code using naming collisions or manipulating configurations. “Naming collisions” refers to a class of vulnerabilities in software systems where similarly named tools from different sources (e.g., public registries vs. local modules) can be resolved unpredictably, allowing malicious actors to run malicious code via injected names or force a manipulated fallback behavior. Recent security research demonstrates that MCP clients often lack the ability to specify unique tool identifiers or strict resolution policies, leading to tool name collisions where malicious data or responses from external servers can override legitimate functionality and hijack execution flows. [13]

Unrestricted private/public repository access in GitHub-based MCP tools

With the GitHub MCP server, the user grants blanket access for the MCP tools to act on behalf of the user. As a result, these tools may obtain unrestricted read and write access at the repository level, across both private and public repositories, rather than narrowly scoped permissions tied to specific repositories or operations. This lack of granular authorization controls enabled malicious or compromised tools to read

sensitive content from private repositories and write or publish it into public repositories without the user's explicit awareness or intent. [14]

Exploitation via messaging platforms

WhatsApp with MCP extensions

There is a trusted third-party MCP agent for WhatsApp—a popular messaging application—to enhance chat capabilities. Researchers demonstrated that the introduction of a malicious MCP server within the agentic system coerced the MCP client to expose WhatsApp message data directly from the WhatsApp server without user notice or approval. This technique works because a) the MCP client was connected to both the malicious and trusted MCP servers and, b) the malicious MCP server's tool descriptions manipulated the MCP client's behavior through a maliciously designed MCP tool. Additionally, to better hide the attempt to leverage the technique, the malicious MCP server advertised a benign instruction at the time of installation and switched to a malicious instruction after the MCP server's second usage. [15]

Poisoning output for downstream automation

MCP agents have produced outputs that, if blindly trusted by downstream agents or systems, may be misinterpreted as executable prompts rather than passive content, enabling manipulation, prompt injection, or data exfiltration. This risk includes semantic manipulation of tool metadata, hidden instructions embedded in outputs, and cascading prompt injections that can influence how subsequent agents interpret and act on earlier results. Such vulnerabilities can arise in multi-agent workflows where one agent's output becomes another's input, allowing malicious content to propagate and lead to unauthorized behavior, data exfiltration, or control-flow hijacking across chained MCP processes. Recent research on semantic and tool poisoning in MCP highlights this class of threats as systemic rather than isolated, showing that compromised descriptors, tool outputs, or shared context can subvert intended operation across many agent pipelines. [16], [17], [18]

Remote code execution vulnerability

Execution of CVE-2025-49596 in an MCP inspector toolchain

A vulnerability (CVE-2025-49596) was disclosed in MCP-Inspector, a product used to test MCP servers under development. The inspector tool accepted unverified inputs,

allowing malicious actors to trigger remote code execution via crafted messages. This issue, fixed in version 0.14.1, underscores how well-known security weaknesses can resurface when AI toolchains overlook security hygiene. [19]

Recommendations

To securely adopt MCP, organizations must move beyond the suggestions mentioned in the protocol and adopt deliberate security controls that are beyond the scope of the document. While some of the following recommendations target developers building MCP client or server implementations, most can be integrated using standard software practices, such as reverse proxies, middleware firewalls, or application sandboxing and containment frameworks. It is essential that MCP operations are brought in line with established secure computing practices without stifling the flexibility and power that make it attractive in the first place.

Choose supported MCP projects when possible

The MCP project documentation has identified that many popular servers are no longer actively maintained. [20] Some of these archived servers interact with data and file systems, development tools, web and browser automation, productivity services, and specialized tools. The MCP GitHub page also maintains a list of reference implementations, helping to identify projects that provide reliable integrations. [21] If the organization has a code audit process, apply it to MCP server projects using the most stringent review profile, particularly when evaluating newer integrations. When possible, deploy and maintain MCP servers or clients locally, and leverage capabilities, such as the MCP Registry service [22], to enroll them and make them available for internal use.

Design for boundaries

Security begins with intentional separation of data and the relevant processing environments. It is important for organizations to clearly define trust boundaries between MCP components, including agents, plugins, models, and end users. These should be treated as residing in different trust zones, each with its own assumptions and controls. For example, data originating from a user facing plugin should not be blindly accepted by a privileged backend model.

Additionally, dynamic tool discovery [23], a hallmark of MCP flexibility, should be treated with caution unless it can be coupled with origin verification or authorization checks.

One practical strategy is to align tools and models with data classification zones. Publicly available tools can be grouped to handle public datasets (e.g., weather data) whereas access to tools that interact with sensitive or regulated information (e.g., national security information, controlled unclassified information, health records, financial information, etc.) should be explicitly controlled and segregated. When processing private data, always prefer a local instance of MCP server to reduce the risk of data leakage.

In order to protect the MCP environment, use a filtering outgoing proxy (e.g., Squid, tinyproxy) or an enterprise data loss prevention (DLP) solution, with specific resource URLs and access methods, for connections destined to external entities. Apart from code restrictions, this can further prevent any unintended data leakage or exfiltration from the MCP environment.

Validate parameters

Parameter validation is critical in a system that is as composable and extensible as MCP. The notion of parameter validation extends beyond simply validating input, by aiming to understand the context and configuration of the execution environment. For example, in an MCP-connected agentic mathematical interpreter LLM, a malicious actor may manipulate the ‘context’ parameter to trigger unintended file I/O behavior, although the mathematical input and output may itself seem valid. It is critical for security that every tool invocation or model execution request validate its inputs against well-defined schemas, expected ranges, and the intended context or configuration where the data will be processed. This includes checking for malformed inputs, missing fields, and excessive sizes—any of which can trigger unstable behavior or be leveraged in prompt injection or DoS techniques.

Parameter forwarding should be explicitly blocked or restricted when the source of the data is ambiguous or potentially user-supplied. Otherwise, inputs intended for one component may be misinterpreted or inappropriately reused by another, leading to cascading unintended outcomes, failures, or data leakage.

Constrain and sandbox tool execution

It is prudent to treat any tool execution triggered via MCP as a potentially high-risk action. To reduce exposure, constraints should be applied to limit access and prevent misuse. Whether a plugin is running a shell command, issuing a database query, or

invoking an external API, it should operate within strict resource and permission boundaries. At the operating system level, security frameworks, such as AppContainers (Windows), seccomp, AppArmor, or SELinux, should be used to isolate each tool's execution context. Tools should also be sandboxed to block lateral movement or privilege escalation if a compromise occurs. Equally important, MCP agent processes themselves should follow the principle of least privilege: if a server does not require access to sensitive file systems, model or data files, or internal networks, those access paths should be explicitly denied at runtime.

Sign and verify MCP messages

Message authenticity and confidentiality should never be assumed, especially when processing sensitive data or making security-critical decisions. MCP currently relies on transport layer encryption (e.g., TLS). The protocol itself cannot enforce or verify encryption and is unaware of message integrity as well. However, the standard can be extended with cryptographic signatures directly within the JSON payload, using capabilities that are readily available with modern libraries to protect sensitive messages. For session protection, time bound signatures can provide additional assurance when needed. MCP messages should include expiration timestamps and replay protection metadata to guard against delayed or duplicated messages, which is a known risk in distributed or event driven systems. The recommendations from OWASP Application Security Verification Standard (ASVS) V7 Session Management, are readily applicable here. In alignment to the well-known OWASP guidance in application security, MCP messages should cryptographically bind requests to time and context to prevent tampering, intentional replay techniques, and unintended re-execution errors.

Filter and monitor output pipelines and chained execution

Outputs from tools and models should never be treated as implicitly trusted, even if they originate from previously vetted components. Each output must be treated as untrusted input to the next phase of the pipeline and therefore undergo scrutiny before being processed or displayed. Filtering should include content length checks, disallowed keyword scanning, rate limiting, and application specific policy enforcement.

As LLMs and agents often generate text that may look benign, but carry hidden logic or prompt manipulation, output filtering should include detection of indirect prompt injection or toolchain pivot attempts. In multi-component MCP pipelines, this includes logging and

inspecting the output of each MCP tool before passing it to the next, to identify injected prompts or code elements that may alter downstream behavior. With regard to inspecting, MCP-aware security proxies remain limited and are still maturing, but may offer partial mitigations. However, given their early stage of development, they should be used with caution, especially when handling sensitive data.

Instrument for logging and detection

Robust observability is essential for MCP environments. All tool and model invocations should be logged, including the exact parameters, identities involved, and (where feasible) cryptographic hashes of results or output. These logs form the backbone of forensic response in the event of a breach or anomaly.

Where possible, this telemetry should be integrated into the existing security monitoring infrastructure of the organization—such as Security Information and Event Management (SIEM) systems, threat detection pipelines, or compliance dashboards. This integration allows developers to inform security teams of reasonable methods to identify suspicious patterns, such as unexpected tool invocations or anomalous message flows, while minimizing false positives.

Track and patch MCP related vulnerabilities

As AI integration layers evolve rapidly, new vulnerabilities in MCP frameworks, tools, and models are likely to emerge. To stay ahead, it is recommended that organizations establish a formal process for monitoring MCP related vulnerabilities, whether through Common Vulnerabilities and Exposures (CVEs), vendor advisories, or open source issue trackers. This includes subscribing to relevant security feeds and staying informed through specialized threat intelligence resources, which are increasingly used by developers and systems integration teams to assess project risk.

Internally, an organization should maintain a clear inventory of all deployed MCP agents and tools, along with versioning, patch history, and known security concerns. A well-maintained registry enables faster triage and coordinated mitigation efforts when a vulnerability is disclosed.

Ultimately, the ability to respond quickly to newly identified risks will be critical to using MCP securely. By investing in systematic vulnerability tracking, organizations will reduce their exposure and the risk of compromise.

Scan local network for open or vulnerable MCP servers

As part of baseline security hygiene, it is recommended that organizations regularly scan their networks to identify insecure or unauthorized MCP servers before they can be exploited. MCP servers, particularly those in development or deployed without hardened configurations, may be accessible on local or public networks without proper authentication, authorization, restrictions, or monitoring. Security teams should proactively scan for such instances to identify and mitigate risk before they can be abused for nefarious activity either from internal or external threats. A plethora of MCP security scanning tools, such as MCP Scanner, Ramparts, CyberMCP, and Proximity, are available and can be used to detect several types of open services in the enterprise network, including:

- Unauthenticated MCP servers: instances accessible without login or authorization.
- Vulnerable MCP servers: deployments with known security flaws or outdated versions.
- Unauthorized MCP deployments: servers installed or running outside approved change control.
- Unregulated Internet connectivity: MCP services with open inbound or outbound traffic paths.

MCP servers may dynamically change ports, making periodic scans and differential reports valuable for detecting new or modified services. These reports give network administrators and security teams the visibility they need to audit changes, investigate anomalies, and enforce security policy across the MCP environment.

Conclusion

MCP represents a promising, but still maturing, foundation for agentic AI, enabling new capabilities in orchestration and automation across models and systems. However, this power comes with significant and evolving security concerns, from serialization risks and prompt injection to opaque update mechanisms and fragile filtering capabilities. Security professionals are beginning to raise concerns about these issues, as seen in recent analyses like Docker's exploration of MCP-related risks, bringing much needed visibility to the topic. [24] Parallel efforts from organizations, such as OASIS COSAI,

have also articulated closely related concerns around scope control, trust boundaries, and agent misuse within MCP-based systems. [25]

While MCP holds real promise as an enabling layer for agentic systems, its current security posture remains uneven and highly dependent on implementation discipline rather than protocol guarantees. Adopters should therefore proceed with caution, drawing on lessons from prior distributed and plugin-based ecosystems while applying heightened scrutiny to the novel integration and automation patterns introduced by MCP. Continued collaborative work among implementers, security researchers, and standards organizations will be essential to establish more robust and trustworthy foundations for AI infrastructure, particularly for national security and other high assurance environments.

Works Cited

[1] Appsec Co. Vulnerable MCP Servers GitHub repository. December 2025. https://github.com/appsecco/vulnerable-mcp-servers-lab

[2] Ars Technica. MCP: The new "USB-C for AI" that's bringing fierce rivals together. 2025. https://arstechnica.com/information-technology/2025/04/mcp-the-new-usb-c-for-ai-thats-bringing-fierce-rivals-together/

[3] Australian Signals Directorate's Australian Cyber Security Centre (ASD's ACSC). Careful adoption of agentic AI services. 2026. https://www.cyber.gov.au/business-government/secure-design/artificial-intelligence/careful-adoption-of-agentic-ai-services

[4] OWASP. A8:2017-Insecure Deserialization. 2018. https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization

[5] Model Context Protocol. Model Context Protocol Specification. 2025. https://modelcontextprotocol.io/specification/

[6] O. Santos. AI Model Context Protocol (MCP) and Security. 2025. https://community.cisco.com/t5/security-blogs/ai-model-context-protocol-mcp-and-security/ba-p/5274394

[7] Model Context Protocol. Authorization. 2025. https://modelcontextprotocol.io/specification/2025-11-25/basic/authorization

[8] Model Context Protocol. Security Best Practices. 2025. https://modelcontextprotocol.io/specification/2025-11-25/basic/security_best_practices#token-passthrough

[9] Model Context Protocol. Lifecycle. 2025. https://modelcontextprotocol.io/specification/2025-11-25/basic/lifecycle#lifecycle

[10] N. Maloyan and D. Namiot. Breaking the Protocol: Security Analysis of the Model Context Protocol Specification and Prompt Injection Vulnerabilities in Tool-Integrated LLM Agents. 2026. https://arxiv.org/abs/2601.17549

[11] V. S. Narajala and O. Narayan. Securing Agentic AI: A Comprehensive Threat Model and Mitigation Framework for Generative AI Agents. 2025. https://arxiv.org/abs/2504.19956

[12] K. Evans, T. Bonner, and C. McCauley. Exploiting MCP Tool Parameters. 2025. https://hiddenlayer.com/innovation-hub/exploiting-mcp-tool-parameters/

[13] Shuli Zhao, et al. Mind Your Server: A Systematic Study of Parasitic Toolchain Attacks on the MCP Ecosystem. 2025. https://arxiv.org/abs/2509.06572

[14] Invariantlabs. GitHub MCP Exploited: Accessing private repositories via MCP. 2025. https://invariantlabs.ai/blog/mcp-github-vulnerability

[15] Invariantlabs. WhatsApp MCP Exploited: Exfiltrating your message history via MCP. 2025. https://invariantlabs.ai/blog/whatsapp-mcp-exploited

[16] S. Jamshidi. Securing the Model Context Protocol: Defending LLMs Against Tool Poisoning and Adversarial Attacks. 2025. https://arxiv.org/abs/2512.06556

[17] Invariantlabs. “Toxic Flows” in Agentic Systems & MCP Servers. 2025. https://invariantlabs.ai/blog/toxic-flow-analysis

[18] Microsoft. Protecting against indirect prompt injection attacks in MCP. 2025. https://developer.microsoft.com/blog/protecting-against-indirect-injection-attacks-mcp

[19] A. Lumelsky. Critical RCE Vulnerability in Anthropic MCP Inspector - CVE-2025-49596. 2025. https://www.oligo.security/blog/critical-rce-vulnerability-in-anthropic-mcp-inspector-cve-2025-49596

[20] Model Context Protocol. Model Context Protocol servers (ARCHIVED). 2025. https://github.com/modelcontextprotocol/servers-archived

[21] Model Context Protocol. Model Context Protocol servers. 2026. https://github.com/modelcontextprotocol/servers

[22] David Soria Parra, Anthropic. Introducing the MCP Registry. 2025. https://blog.modelcontextprotocol.io/posts/2025-09-08-mcp-registry-preview/

[23] Model Context Protocol. Tools. 2025. https://modelcontextprotocol.io/specification/2025-11-25/server/tools

[24] Docker. MCP Horror Stories: The Security Issues Threatening AI Infrastructure. 2025. https://www.docker.com/blog/mcp-security-issues-threatening-ai-infrastructure/

[25] CoSAI-OASIS. Ws4 Secure Design Agentic Systems. 2026. https://github.com/cosai-oasis/ws4-secure-design-agentic-systems/blob/main/model-context-protocol-security.md#anti-scope

Disclaimer of endorsement

The information and opinions contained in this document are provided "as is" and without any warranties or guarantees. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the United States Government, and this guidance shall not be used for advertising or product endorsement purposes.

Purpose

This document was developed in furtherance of NSA’s cybersecurity mission, including its responsibilities to identify and disseminate threats to, and develop and issue cybersecurity specifications and mitigations for, National Security System, Department of War, and Defense Industrial Base (DIB) information systems. This information may be shared broadly to reach all appropriate stakeholders.

Contact

Cybersecurity Report Feedback: CybersecurityReports@nsa.gov
Defense Industrial Base Inquiries and Cybersecurity Services: DIB_Defense@cyber.nsa.gov
Media Inquiries / Press Desk: NSA Media Relations: 443-634-0721, MediaRelations@nsa.gov