Model Context Protocol Vulnerabilities: 2025

Table of Contents

Introduction

In 2025, vulnerabilities in software implementing the Model Context Protocol (MCP) led to the publication of 99 CVEs. This article examines these MCP-related vulnerabilities using data from the National Vulnerability Database (NVD) focusing on:

- Vulnerability Severity (CVSS scores)
- CVSS Vectors
    - Attack Vector and Complexity
    - Privileges Required and User Interaction
    - Impact (Confidentiality, Integrity and Availability)
- Common Weaknesses (CWEs)
- Exploitability scores (EPSS)
- Defenses

We also compare MCP-related CVEs to general AI software vulnerabilities and give practical recommendations to help security teams protect agentic AI systems.

Summary Findings

Vulnerabilities in MCP-related software are on average more severe than in general software, with higher impact across Confidentiality, Integrity and Availability and higher exploitation rates as measured by EPSS but also requiring more complex attacks and a higher degree of user interaction for successful exploitation.

When mapped against Common Weaknesses, vulnerabilities in MCP software cluster heavily around command injection, code injection, cross-site scripting, cross-site request forgery, and authentication / authorization weaknesses.

While web server security has matured over decades with established defenses (WAFs, hardened configs, HTTPS mandates), MCP remains an emerging technology with lower overall maturity and fewer proven controls. Producers and consumers of MCP-related software should therefore apply and extend their current web server security procedures, adding AI-specific mitigations like sandboxing, tool signing/verification, human oversight, least-privilege enforcement, and supply-chain scanning — when building, deploying, or adopting MCP-based solutions.

Severity

Vulnerabilities in AI-related products are more severe (average CVSS Score = 7.01) than vulnerabilities in other products (average CVSS score = 6.60). Vulnerabilities in MCP-related software are even higher, although not by a statistically significant amount. (average CVSS score = 7.25).

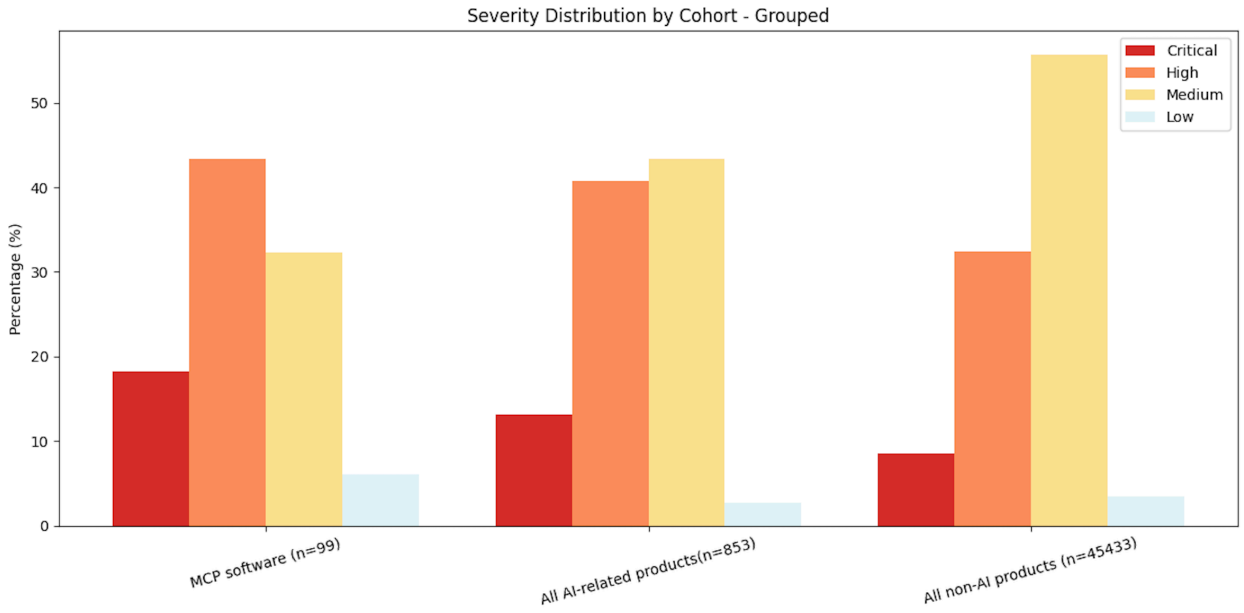

Figure 1. 2025 severity of CVEs in MCP-related software (n=99), AI-related software, AI-related software (n=853), and all other products

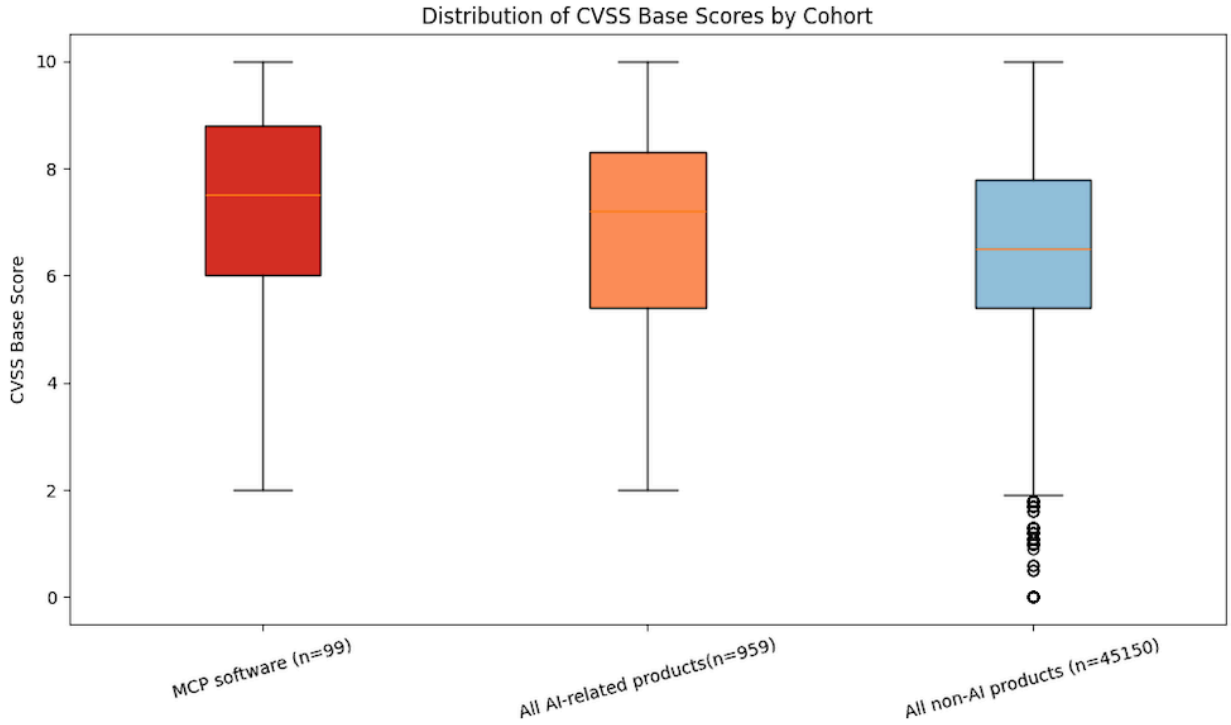

Figure 2. Box Chart distribution of CVSS base score of 2025 CVEs in MCP-related software, AI-related software, and all other products

CVSS Vectors

Using the eight vectors of the CVSS scoring system we find that vulnerabilities in MCP-related software differ significantly from other vulnerabilities. MCP vulnerabilities require fewer privileges to exploit (due to exposed endpoints, missing authentication, or trust assumptions in local/remote setups), while they exhibit higher attack complexity requiring multi-step chaining, probabilistic LLM manipulation, or specialized conditions like crafted prompts/metadata. In addition, exploited MCP vulnerabilities frequently cause higher impact across Confidentiality (e.g., exfiltration of API keys, credentials, files, or sensitive data via injections or poisoning), Integrity (e.g., manipulation of tool outputs, data tampering, or agent behavior hijacking), and Availability (e.g., resource exhaustion, denial-of-service, or system compromise leading to outages) compared to typical vulnerabilities in other software categories.

Attack Vector

Vulnerabilities in MCP- and AI-related software are more likely to be exploitable over a network and less likely to be susceptible to local exploitation (5% local for MCP-related software vs 22% for vulnerabilities in other software).

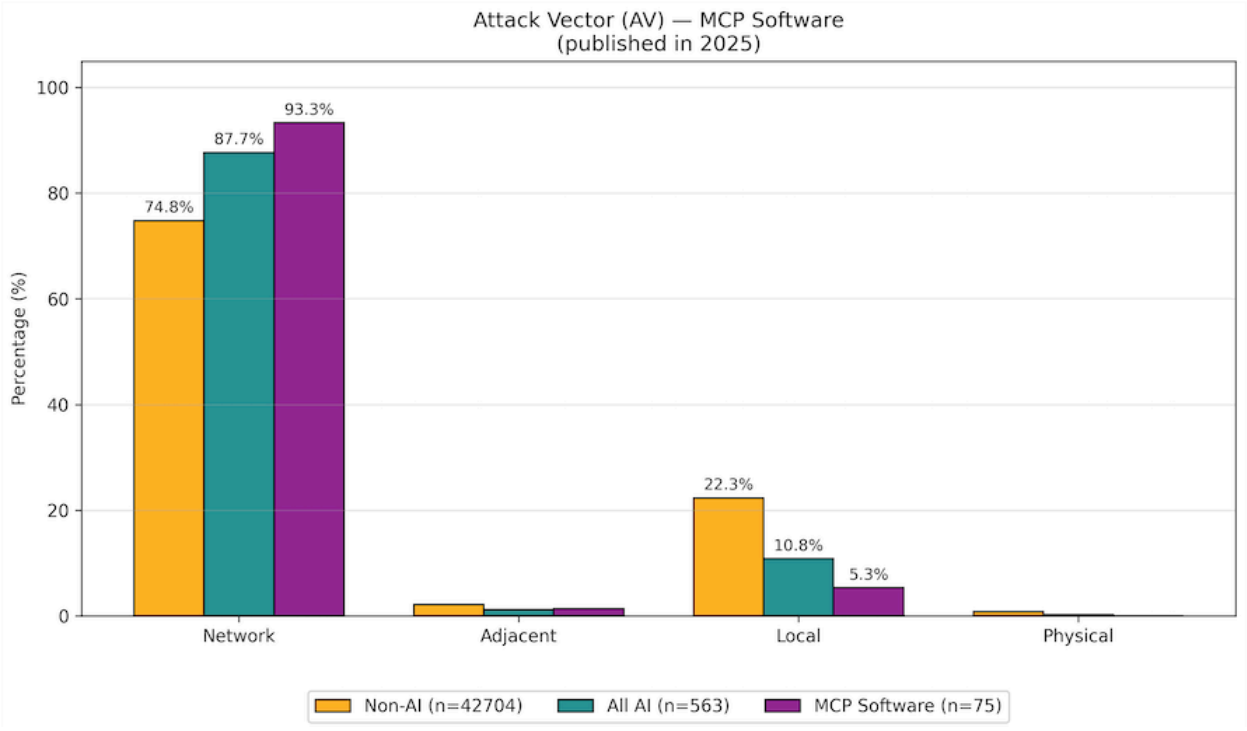

Figure 3. Attack vector frequency in MCP-related software, AI-related software ,and all other products

AI/MCP software is typically deployed in networked environments with web-exposed APIs, interconnected multi-cloud/SaaS integrations, and distributed services, creating far more network entry points than software that runs locally or with minimal network exposure.

AI systems rely on internet-fed datasets, model updates, open-source libraries, and external APIs for training and inference, introducing remote supply-chain risks, data poisoning, and adversarial input opportunities not common in other applications.

Attack Complexity

27% of MCP-related vulnerabilities have "High" attack complexity, versus 9% for AI-related vulnerabilities and 9% for all other vulnerabilities.

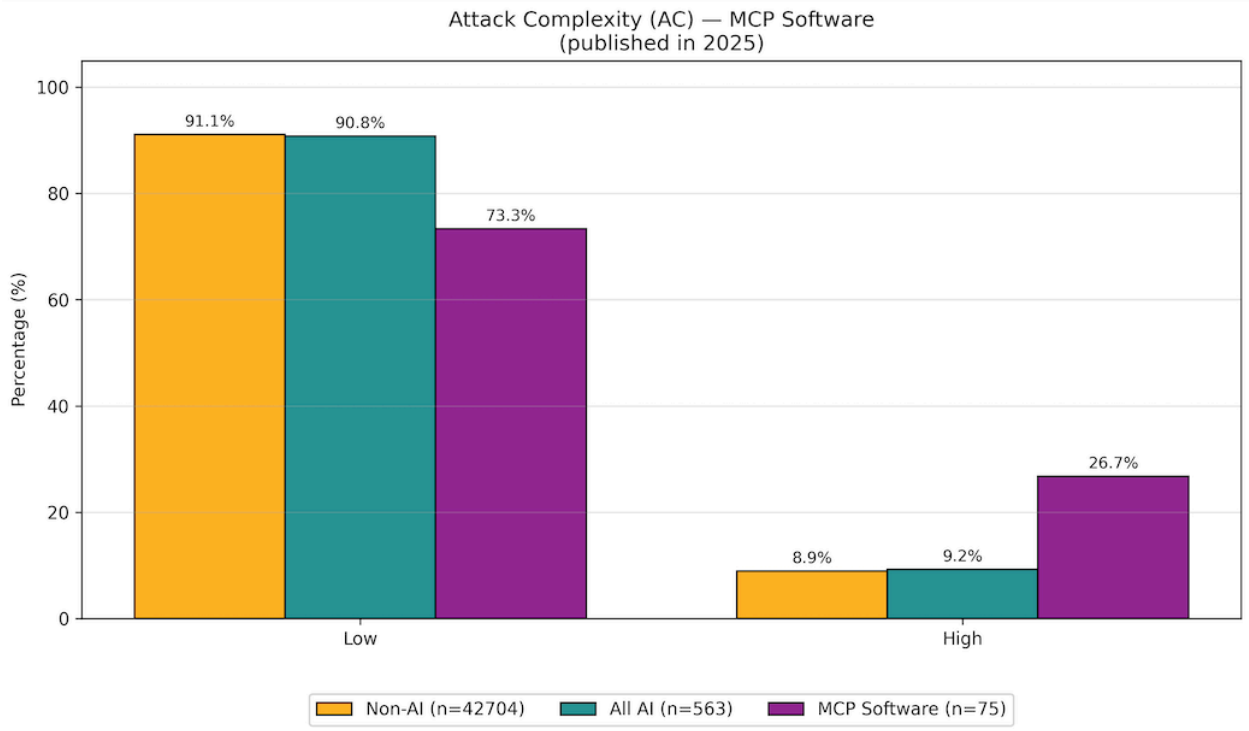

Figure 4. Attack complexity for CVEs in MCP-related software, AI-related software, and all other products.

MCP's agentic, dynamic, tool-delegating design demands more sophisticated, context-dependent attacks than typical AI prompt injections or classic software bugs. This raises the CVSS attack complexity rating:

- Exploiting MCP vulnerabilities frequently involves multi-step chaining across LLM, MCP client/server, and external tools/APIs.
- Exploitation depends on probabilistic LLM behavior (subtle prompt/tool-description crafting with unreliable outcomes).
- Successful attacks may require specialized conditions (specific configs, network positioning, malicious server setup, or supply-chain timing).

- MCP is an emerging protocol, so attacks combine novel protocol mechanics with traditional flaws, making them less straightforward.

Privileges Required

Vulnerabilities in MCP-related software require fewer privileges than other vulnerabilities. MCP (Model Context Protocol) implementations frequently expose serious issues with **minimal or no authentication prerequisites** due to their design, deployment patterns, and ecosystem maturity in 2025–2026. Key reasons include:

- **Missing or weak authentication by default** — Many MCP servers (e.g., proxies, inspectors, filesystem connectors) listen on localhost/0.0.0.0 without built-in auth, assuming "local-only" safety. This allows unauthenticated network access (via browser tricks, DNS rebinding, or direct connections), as seen in critical cases like **CVE-2025-49596** (RCE in MCP Inspector) and **CVE-2025-6514** (command injection in mcp-remote), both explicitly scored with **Privileges Required: None**.
- **Trust assumptions in agentic/local setups** — MCP often runs in developer/IDE environments (VS Code extensions, Claude tools) where components implicitly trust incoming connections or tool calls. No user credentials are needed if the attacker can reach the exposed endpoint (e.g., via malicious websites triggering WebSocket/HTTP requests to localhost).
- **Over-privileged defaults and confused deputy problems** — Servers frequently inherit the host user's broad privileges (files, APIs, tokens) without enforcing least-privilege scoping. An unauthenticated attacker exploits this directly (e.g., via injection or tool poisoning) to abuse the server's high-privilege context, without needing their own elevated access.
- **Emerging/immature ecosystem** — Early MCP tools prioritize rapid integration over security hardening (e.g., no mandatory OAuth delegation, weak token validation, or permissive specs). This leads to widespread insecure-by-default implementations, especially in open-source forks or third-party connectors, where exploits succeed remotely without prior privileges.

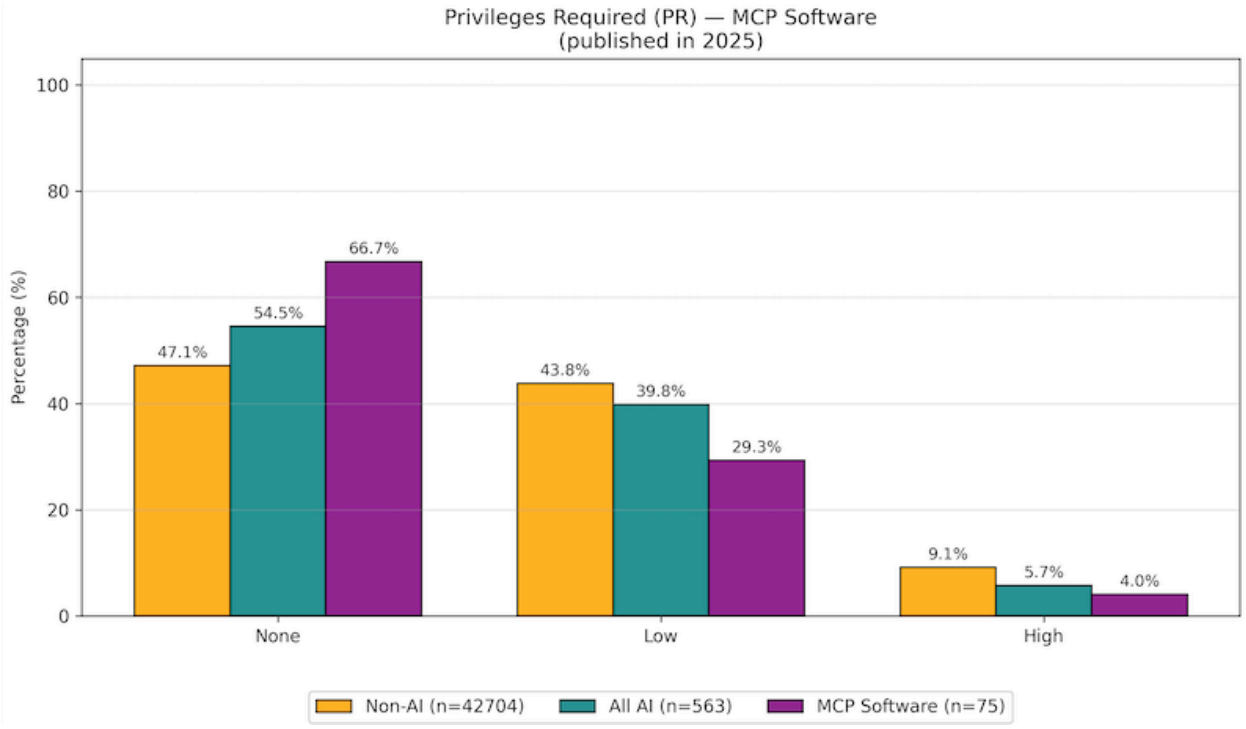

Figure 5. Privileges required to exploit CVEs in MCP-related software, AI-related software and all other products

In contrast, traditional software vulnerabilities more often require some authentication or user-level access (e.g., logged-in accounts for web apps), while MCP's "plug-and-play" nature for AI agents creates a larger unauthenticated attack surface.

User Interaction

On average, vulnerabilities in MCP software require user interaction for successful exploitation more frequently than other vulnerabilities.

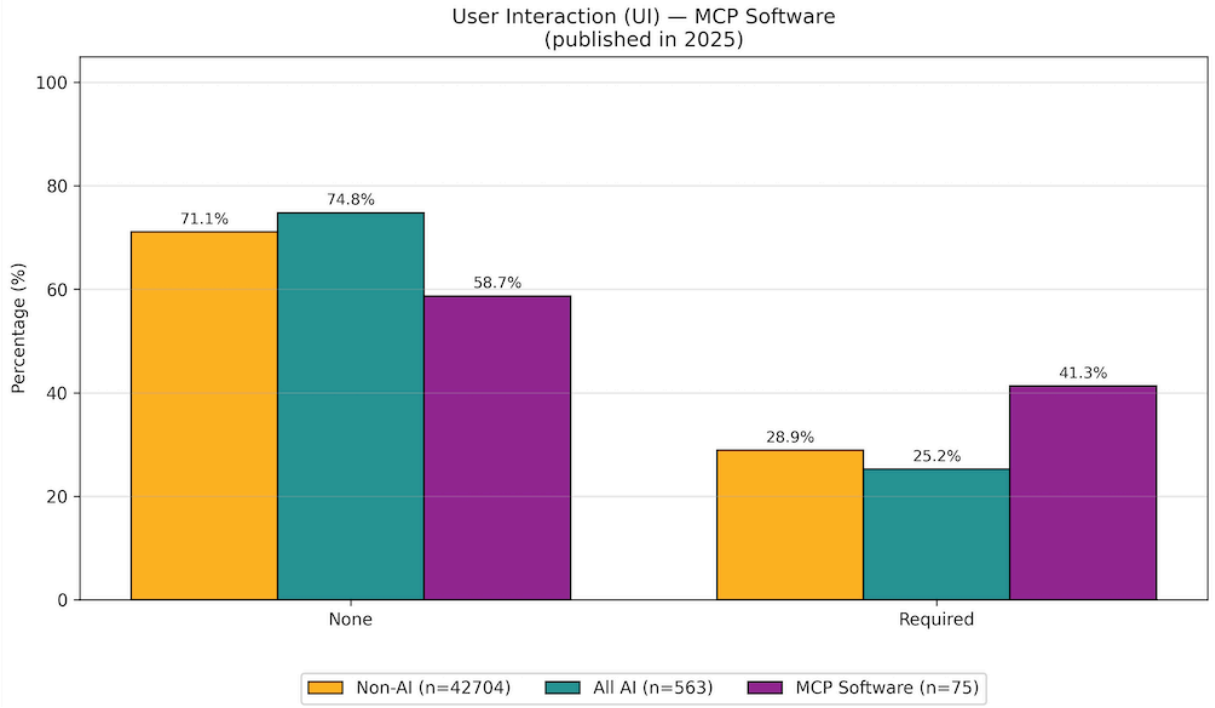

Figure 6. User Interaction required to exploit vulnerabilities in MCP-related software, AI-related software, and all other products.

MCP (Model Context Protocol) vulnerabilities often involve **higher rates of Required User Interaction** due to their agentic, interactive, and human-in-the-loop nature in 2025–2026 ecosystems. Key reasons include:

- **Reliance on user-driven workflows and natural-language prompts** — Many exploits (e.g., indirect prompt injection, tool poisoning, or conversation hijacking) depend on the victim providing a prompt or initiating an AI interaction that triggers the malicious behavior. The attacker crafts hidden instructions in tool descriptions/metadata, but successful exploitation requires the user to ask the LLM something that invokes the poisoned tool or processes the injected context.
- **Browser-based or IDE-integrated attack chains** — Critical issues like **CVE-2025-49596** (RCE in MCP Inspector) or chained browser flaws (e.g., 0.0.0.0 Day + CSRF) succeed when the user visits a malicious website, which then interacts with a local MCP component. This "drive-by" style requires the victim to actively browse/click, qualifying as Required User Interaction in CVSS.
- **Human oversight and confirmation assumptions in the protocol** — MCP specs and best practices emphasize human-in-the-loop for sensitive actions (e.g., tool calls, OAuth flows, or file operations). Many implementations add prompts for user approval/confirmation before execution. Exploits that bypass or trick this step still often need the user to engage (e.g., approve a seemingly benign request), raising the UI metric.

- **Emerging ecosystem with interactive debugging/testing tools** — Tools like MCP Inspector or developer IDE extensions (VS Code, Cursor) involve manual interaction for testing/debugging. Vulnerabilities in these (e.g., unauthenticated proxy access leading to RCE) frequently require the developer/user to run the tool or load a malicious config/page, unlike fully automated server-side flaws.

In contrast, many traditional or non-agentic vulnerabilities (e.g., remote code execution via network packets, automated supply-chain compromises) can trigger without any user action, leading to lower UI rates overall.

This pattern makes MCP exploits **more targeted and less wormable** (good for containment) but still dangerous in social-engineering or developer-targeted scenarios, where user interaction is easy to elicit.

Confidentiality

Vulnerabilities in MCP-related and broader AI-related software exhibit significantly higher CVSS Confidentiality impact scores than those in other software.

This elevated confidentiality impact stems from the fundamental nature of these systems: AI and especially MCP (Model Context Protocol) implementations frequently handle extremely sensitive data, including proprietary training datasets, user prompts containing PII/business secrets, authentication tokens, internal knowledge bases, and real-time context across tools and APIs. A successful exploit (e.g., prompt injection, context leakage, model inversion, or insecure tool access) often results in direct exposure or exfiltration of this high-value, confidential information, leading to a greater proportion of High or Critical Confidentiality ratings compared to traditional software vulnerabilities

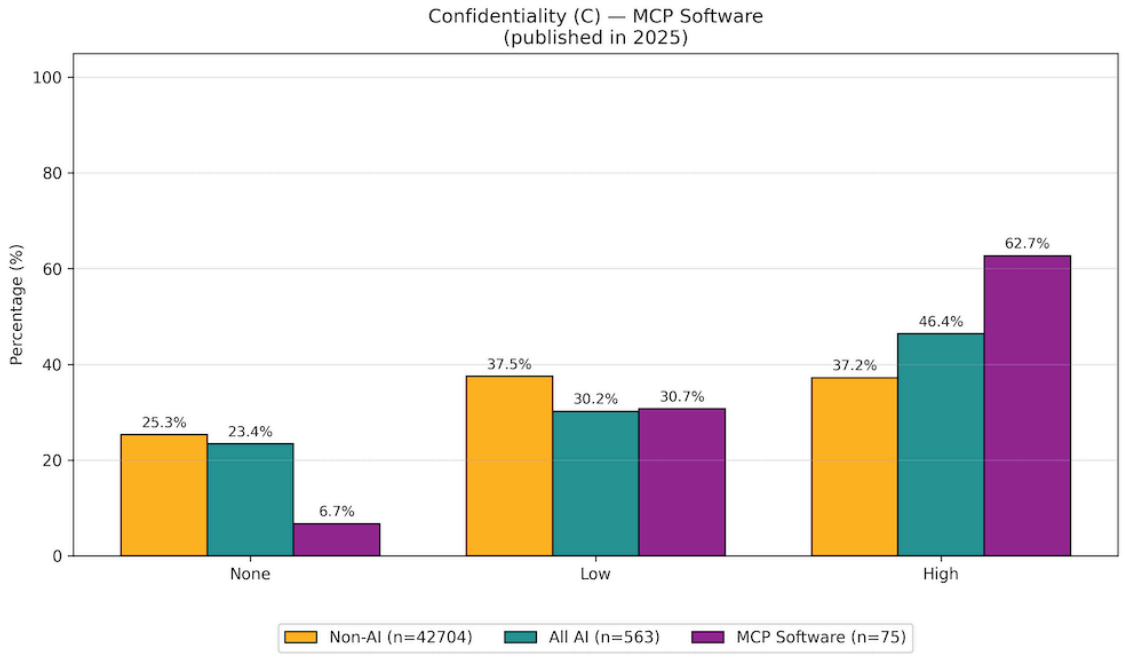

Figure 7. Confidentiality impact of 2025 CVEs in MCP-related software, AI-related software and all other products

Integrity

Vulnerabilities in MCP-related and broader AI-related software demonstrate higher CVSS Integrity Impact scores compared to other software.

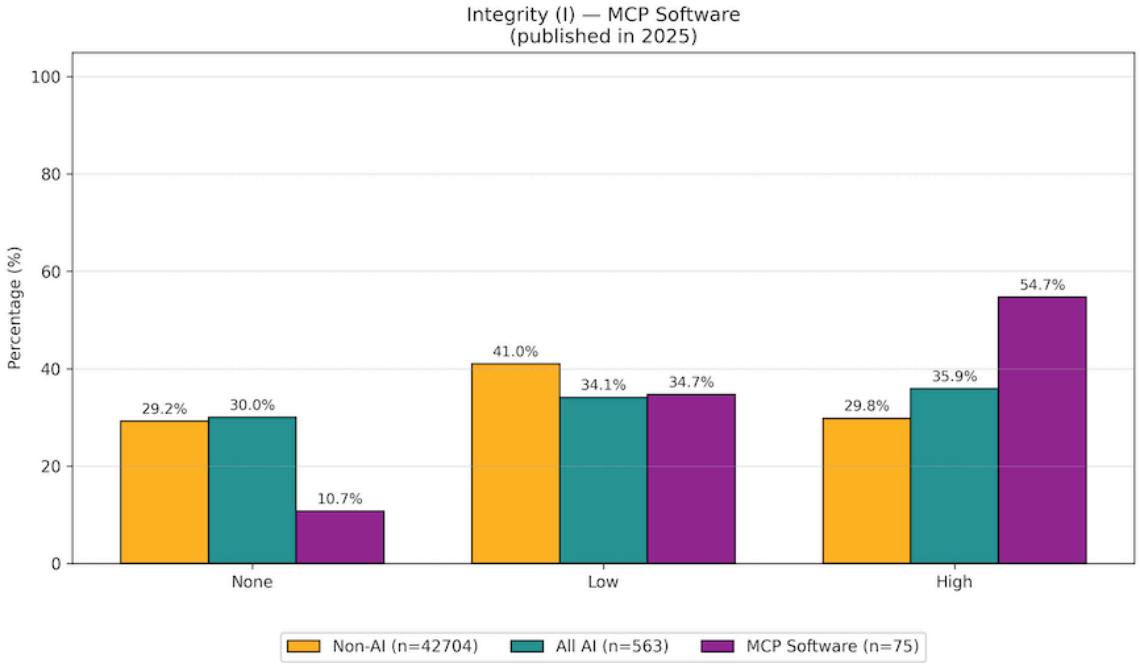

Figure 8. Integrity impact of 2025 CVEs in MCP-related software, AI-related software and all other products

We believe this elevated integrity impact arises because MCP implementations frequently operate as trusted intermediaries that can execute arbitrary code, invoke tools/APIs, modify memory/context, or alter model behavior on behalf of users. Successful exploits (e.g., prompt injection leading to tool misuse, context tampering, malicious plugin execution, or jailbreaking that bypasses safety guardrails) can result in unauthorized modification of critical data, business logic, or system state; consequences that are often classified as High or Critical for integrity.

Availability

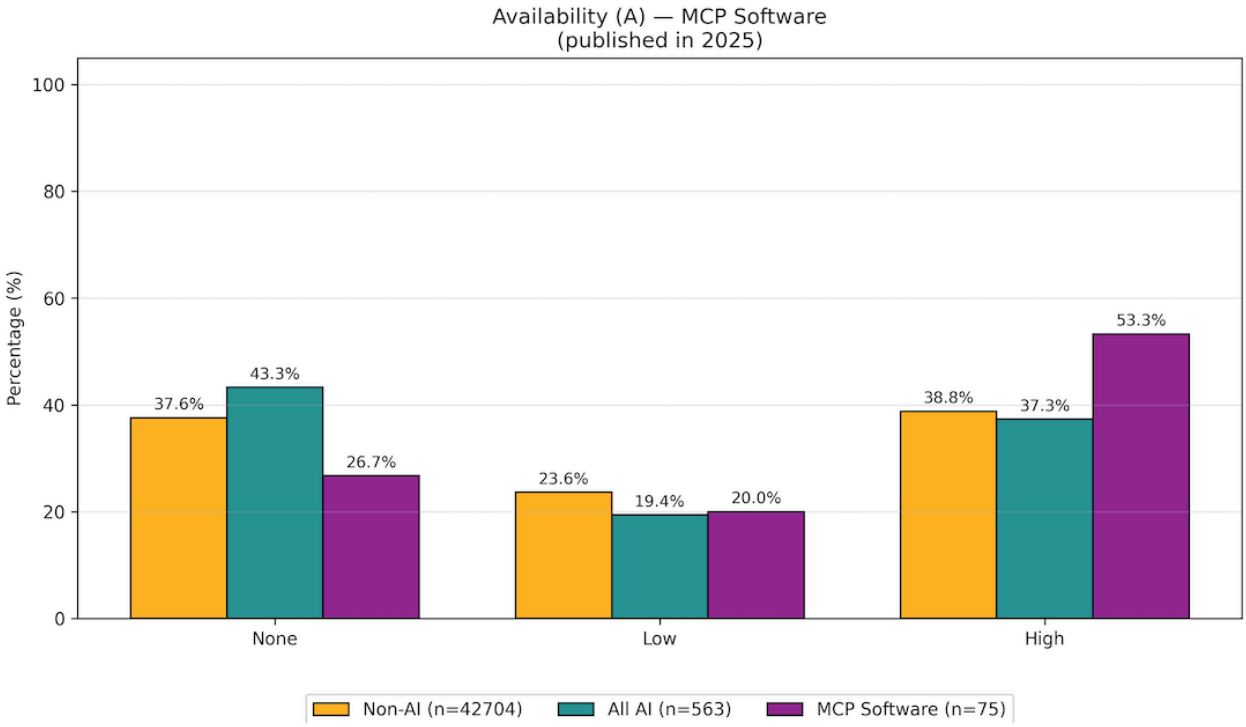

Figure 9. Availability impact of 2025 CVEs in MCP-related, AI-related and all other software.

The CVSS vector "Scope" shows no statistically significant differences among MCP-related software, Ai-related software and other software.

Common Weaknesses (CWEs) in MCP Software

CVEs in MCP software are concentrated among three groups of weaknesses:

- Injection weaknesses (CWE-77, CWE-78) comprising more than 60% of CVEs
- Cross-site scripting (CWE-79) and Server-Side Request Forgery (CWE-918)
- Improper access controls (CWE-284) and authentication (CWE-306)

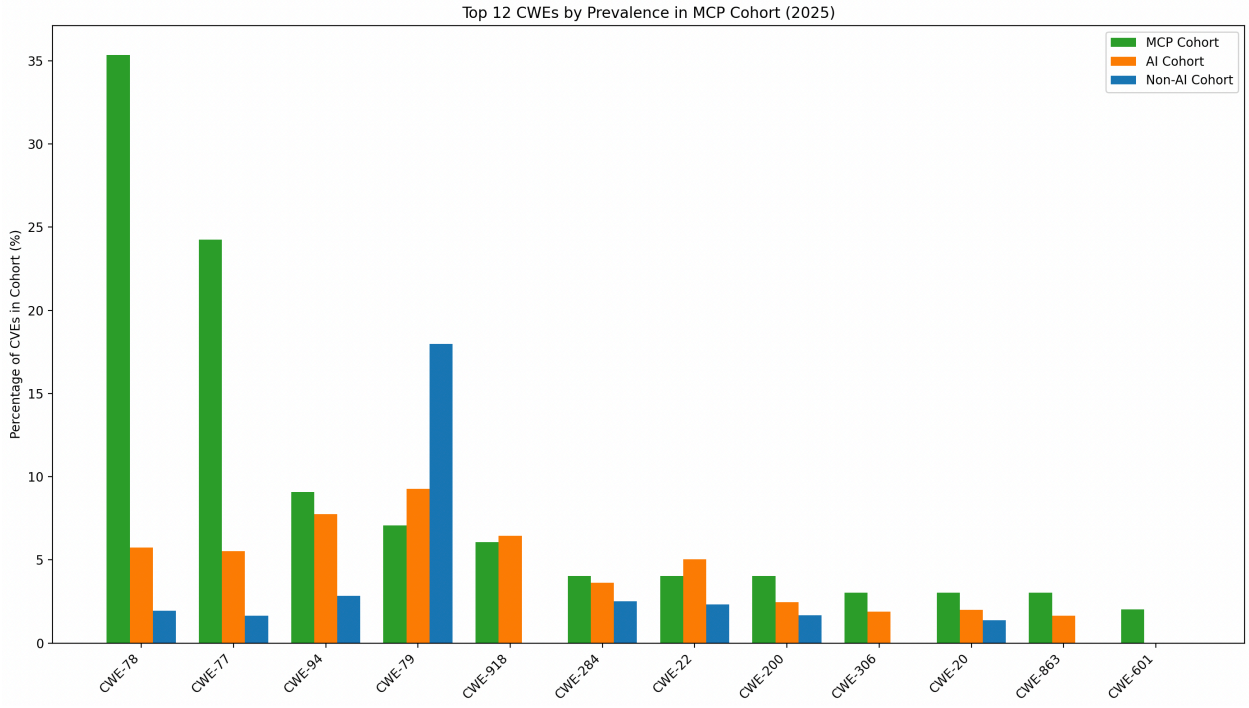

Figure 10. Most prevalent Common Weaknesses for CVEs in MCP-related software, with corresponding prevalence in AI-related software and in all other products.

We discuss the prominence of CWE-78 and CWE-77 in detail.

OS Command Injection (CWE-78)

CWE-78, titled "Improper Neutralization of Special Elements used in an OS Command, refers to vulnerabilities where untrusted input is inadequately sanitized before being incorporated into operating system commands. This can allow attackers to inject malicious commands, leading to arbitrary code execution, privilege escalation, data theft, or system compromise. The high prevalence (35%) of this CVE in our 2025 MCP cohort analysis highlights a systemic weakness in MCP, which is designed for agentic AI systems where large language models (LLMs) interact with external tools, APIs, or environments to perform actions.

The significance stems from MCP's core architecture:

- **Dynamic Tool Integration and Execution**: MCP enables AI agents to discover, negotiate, and invoke services dynamically, often involving OS-level commands for tasks like file operations, network interactions, or script execution. Without proper input validation, user prompts or agent communications can inject shell metacharacters (e.g., ;, , &), turning benign requests into malicious OS commands. For example, a vulnerability like CVE-2025-6514 in the mcp-remote project demonstrates how untrusted server connections can trigger arbitrary OS command

execution, affecting Windows, macOS, and Linux differently but universally enabling remote code execution (RCE).

- **Agentic AI's Expanded Attack Surface**: Unlike traditional software, MCP-based systems (e.g., in LLM agents like LangChain or AutoGPT) blend natural language processing with real-world actions, amplifying injection risks. Prompt engineering or adversarial inputs can exploit this and trick agents to execute unsanitized commands derived from user queries. In multi-agent ecosystems, where protocols like MCP facilitate agent-to-agent communication, injected commands can potentially be propagated across networks.
- **High Severity in Critical Contexts**: CWE-78 often scores high on CVSS (e.g., 8.0+ in our MCP data), as it can enable full system compromise. In 2025, this aligns with OWASP's Agentic AI Top 10, where injection-related threats (e.g., prompt/tool injection) rank highly due to their potential for excessive agency or unauthorized actions.

Compared to non-MCP cohorts, where CWE-78 might appear lower (e.g., ~1-2% in general vulnerabilities), its outsized presence in MCP (35%) underscores the protocol's immaturity and the rapid adoption of AI agents without robust security controls

Command Injection (CWE-77)

CWE-77, "Improper Neutralization of Special Elements used in a Command ('Command Injection')", is the parent weakness for various injection flaws into command interpreters. It encompasses a broader range than CWE-78 ("OS Command Injection"), which is specifically for operating system shell commands. CWE-77 includes injections into other command languages, such as SQL (though often tagged separately), LDAP, or application-specific interpreters. In our MCP cohort (99 CVEs from 2025), CWE-77 appears at ~24% prevalence, second to CWE-78 (~35%).

This high ranking (combined ~60% for CWE-77 and CWE-78) underscores a **fundamental architectural risk in MCP implementations**: the protocol's reliance on dynamic execution of commands derived from untrusted or semi-trusted inputs, such as LLM-generated tool calls, agent negotiations, or external service interactions.

Key Reasons for CWE-77's Prominence in MCP:

- **Broader Injection Vectors in Agentic Systems**: MCP enables AI agents to invoke tools, scripts, or subprocesses dynamically. Vulnerabilities often arise when inputs (e.g., from prompts, RAG data, or tool responses) are passed to non-OS command executors, like custom parsers, scripting engines (e.g., Python's subprocess with shell=False but flawed argument handling), or domain-specific languages. While CWE-78 captures direct shell injections (e.g., via system() or popen() with shell=True), CWE-77 tags cases where injection targets a "command" in a wider sense—common in MCP's modular, protocol-driven tool calling.

- **Tool and Subprocess Misuse**: Many MCP servers or clients use libraries like Python's subprocess, Java's ProcessBuilder, or Node.js child_process. Flaws in argument sanitization (e.g., concatenating untrusted data into command arrays) trigger CWE-77. In AI agents, this amplifies via "excessive agency," where LLMs interpret prompts and generate executable commands without strict boundaries.
- **Overlap and Tagging Practices**: In NVD, analysts sometimes use CWE-77 for general command injection when not strictly OS-shell (CWE-78). Our data shows high frequency for both, suggesting MCP vulns span pure OS RCE (CWE-78) and broader command manipulations (CWE-77), like injecting into agent workflows or non-shell executors.
- **Emerging from Prompt/Tool Injection**: MCP's exposure to indirect prompt injection (e.g., malicious tool metadata or responses) often manifests as command injection downstream. This aligns with 2025 trends in agentic AI, where command execution bypasses approvals, leading to RCE.

The dominance of injection CWEs indicates that MCP implementations prioritize interoperability over secure input handling. Developers often underestimate risks in tool calling, leading to repeatable patterns exploitable via prompt engineering or malicious servers.

Exploitability

CISA's Known Exploited Vulnerabilities (KEV) list contains only contains about 1 in every 300 vulnerabilities and is not suitable to analyze smaller data sets such as AI-related vulnerabilities or MCP-related vulnerabilities. We instead analyze our vulnerabilities in terms of their EPSS scores.

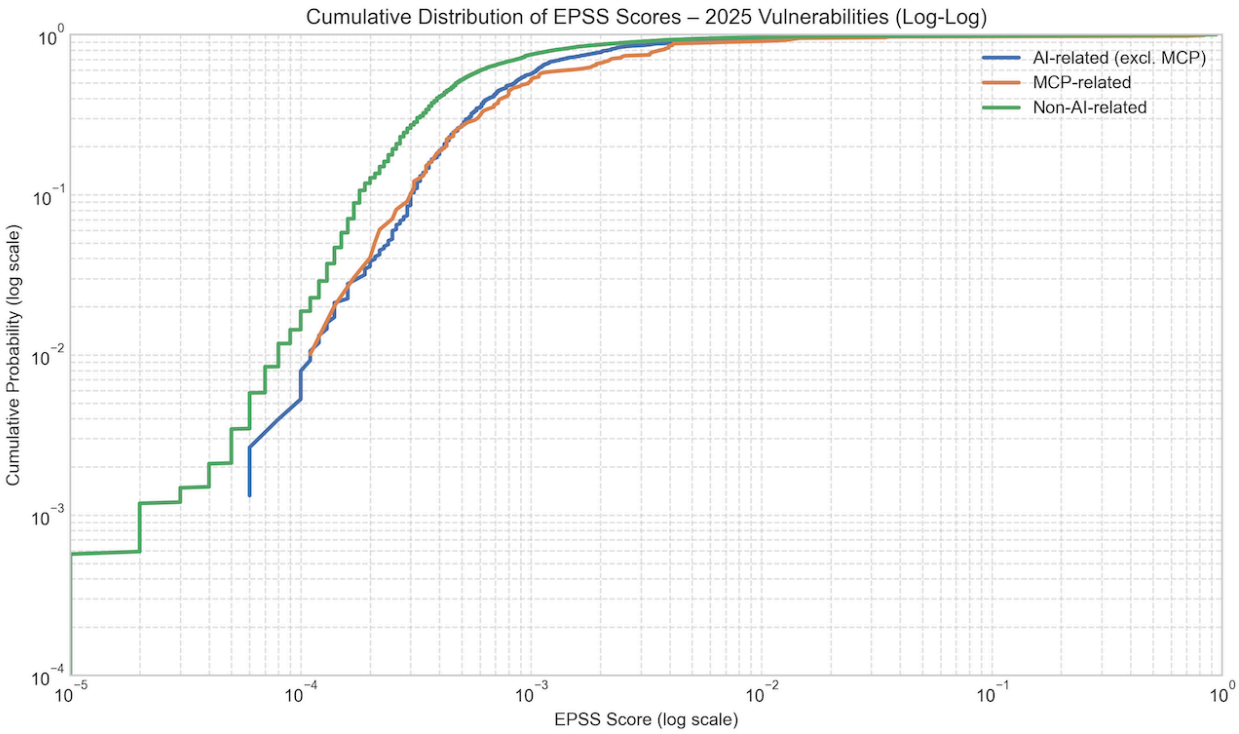

Figure 11. Cumulative Distribution Curve of EPSS Scores in MCP-related software, AI-related software and all other products.

EPSS scores for vulnerabilities in AI-related software and MCP-related software are significantly higher than for other vulnerabilities. 3.0% of vulnerabilities in MCP-related software have an EPSS score of >0.5, while this number is 0.8% for AI-related software and 0.5% in other software. In summary, vulnerabilities in the emerging AI and MCP ecosystem appear to carry materially higher near-term exploitation risk than traditional software vulnerabilities published in 2025.

Recommendations

Software Producers

Developers and organizations producing MCP-related software should prioritize known defenses against CWE-77 and CWE-78. Avoid executing external commands directly and instead use built-in libraries or APIs. Use proven processes for avoiding code injection, cross-site scripting, cross-site request forgery and authentication / authorization issues,

Recommendations for Enterprises (Software Consumers)

Organizations consuming MCP-related software should request copies of penetration test results focusing on command injection, code injection, XSS, CSRF and authentication /

authorization. Organizations with available resources should perform their own evaluation of MCP software, including fuzz testing.

Organizations wishing to take a proactive approach to the growing areas of MCP and agentic AI security should consider specialized platforms specifically designed for this purpose. These may address gaps that general tools miss by focusing on automatic discovery; MCP monitoring and logging; and policy enforcement.

References

[1] https://www.belfercenter.org/publication/AttackingAI

[2] https://www.upwind.io/feed/ai-vulnerabilities-vs-traditional-attack-surface

[3] https://www.paloaltonetworks.com/cyberpedia/generative-ai-security-risks

[4] https://checkmarx.com/learn/ai-security/why-ai-generated-code-may-be-less-secure-and-how-to-protect-it/

[5] https://airc.nist.gov/airmf-resources/airmf/appendices/app-b-how-ai-risks-differ-from-traditional-software-risks/

[6] https://www.sonatype.com/press-releases/sonatype-intelligence-report-cve-crisis

[7] https://www.cisa.gov/news-events/news/software-must-be-secure-design-and-artificial-intelligence-no-exception

Author: Arve Kjoelen

Date: 30 January, 2026