THE MOTHER OF ALL AI SUPPLY CHAINS

Anthropic's "By Design" Failure at the Heart of the AI Ecosystem

150M+ downloads

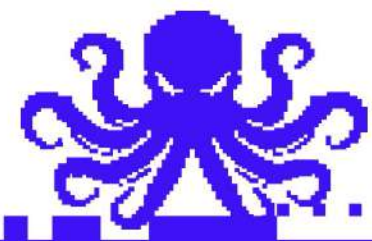
200K+ exposed servers

10 CVEs and counting

OX Research | April 2026

Moshe Ben Siman Tov, Nir Zadok, Mustafa Naamnih, Roni Bar

Table of Contents



Executive Summary:
Architectural, Not incidental

The OX Security Research team has uncovered a critical, systemic vulnerability at the core of the Model Context Protocol (MCP) — the AI agent communication standard created and maintained by Anthropic.
The flaw enables arbitrary command execution on any server running a vulnerable MCP implementation, giving attackers direct access to user data, databases, API keys, chat histories, and much more.

The vulnerability is not a one-off coding mistake. It is an architectural design decision baked into Anthropic's MCP code across every supported programming language — Python, TypeScript, Java, Kotlin, C#, Go, Ruby, Swift, PHP, and Rust. Any developer who builds on Anthropic's MCP inherits the exposure.

The blast radius is massive. This exploit allowed us to directly execute commands on six official services of real companies with real paying customers, and to take over thousands of public servers spanning over 200 popular open-source GitHub projects with hundreds of millions of downloads.

We repeatedly tried to convince Anthropic to patch their code in ways that would instantly protect millions of users. They declined each time. We informed them this research would be made public, to which they raised no objection.

During our research **we conducted over 30 responsible disclosure processes, produced over 10 CVEs rated Critical and High, and helped patch numerous projects, protecting millions of users and organizations** around the world.

We **executed commands on 6 live production platforms** with real paying customers.

We **took over thousands of public servers** spanning **200+** popular open-source projects.

We **uploaded a proof-of-concept malicious MCP to 9 of 11 major MCP marketplaces** — unchallenged.

Blast Radius at a Glance

Anthropic MCP (Python) downloads
73,277,959

LiteLLM downloads
57,256,574

FastMCP downloads
22,470,397

Total dependent repositories
32,682

Public servers found vulnerable (Shodan)
7,374

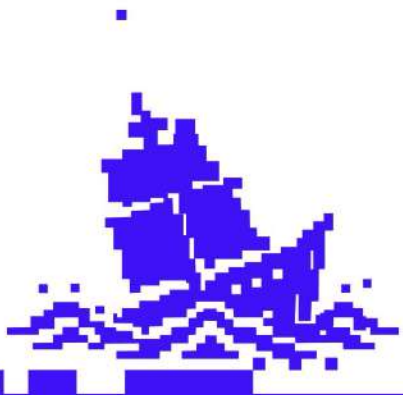
Potentially exposed servers
200,000+

MCP Marketplaces allowing arbitrary STDIO upload
9 out of 11

Real company platforms directly exploited
6

Responsible disclosures conducted
30+

CVEs issued (Critical/High)
10+

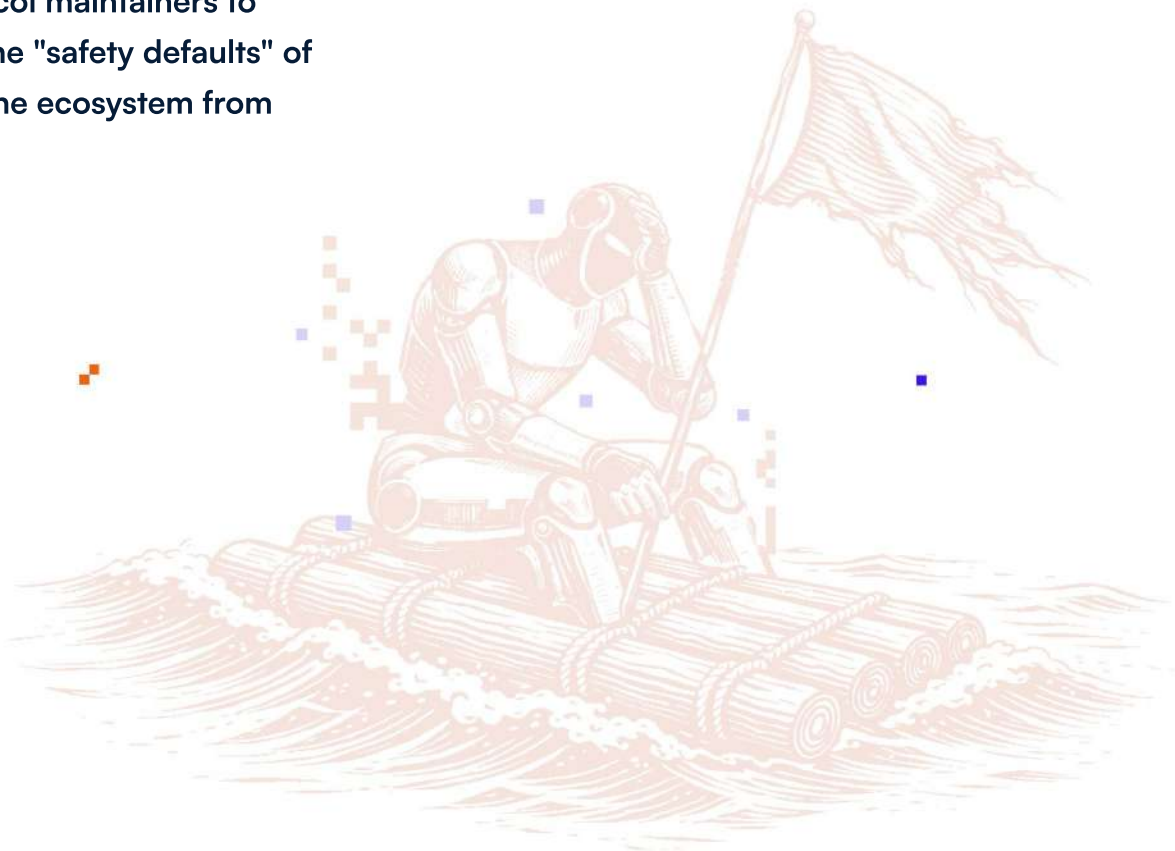


Note on Supply Chain Responsibility

This vulnerability family could have been avoided entirely if Anthropic had architected MCP to be **Secure by Design** at the root. Instead, the current model relies on **Fault-Diversion**: pushing the burden of complex security sanitization onto downstream developers.

Developers are not security engineers; we cannot expect tens of thousands of implementers to independently discover and mitigate a flaw that is baked into the official SDKs they trust. By shifting the blame rather than hardening the protocol, the industry leaves user data and organizational infrastructure exposed. **True supply chain maturity requires protocol maintainers to take responsibility for the "safety defaults" of their code, protecting the ecosystem from the root up.**

This architectural failure highlights an even broader, systemic trend. As AI-assisted code generation accelerates, individuals with limited technical expertise are deploying an unprecedented volume of projects. However, generating more code without foundational security knowledge exponentially widens the gap in organizational defenses - a phenomenon detailed in our *Army of Juniors* report. Combine this lack of security maturity with the reality that these implementations possess deep capabilities to read private chat conversations, utilize highly privileged API keys, and connect to any configured external service, and the resulting threat multiplier becomes profoundly dangerous.



Introduction:
When Expected Behavior Leads to RCE

Starting November 2025, the OX Research team conducted a **large-scale security investigation of MCP implementations.** The result: over 30 critical vulnerabilities across numerous LLM tools and open-source projects — all tracing back to a single piece of "expected behavior" in the MCP protocol.

MCP was created by Anthropic in November 2024 as a universal way to connect AI models to tools, databases, and other LLMs — a plug-and-play protocol designed to reduce the friction of writing custom integration code. **By design, it sits at the most sensitive junction in any AI-powered stack**: it has access to source code, customer data, internal workflows, and outward-facing services.

That privileged position makes a **foundational flaw in MCP's architecture especially dangerous.** And the exploit mechanism is straightforward: MCP's STDIO interface was designed to launch a local server process. But the command is executed regardless of whether the process starts successfully. Pass in a malicious command, receive an error — and the command still runs. No sanitization warnings. No red flags in the developer toolchain. Nothing.

The attack surfaces this enables include:

- **Unauthenticated remote command execution**
- **Authenticated remote command execution**
- **Information leakage and data exfiltration**
- **Crypto mining on victim servers**
- **Lateral movement within compromised networks**

Most of these attacks are trivially simple to execute once the STDIO configuration is accessible to user input — directly through a UI, via a modified network request, or through prompt injection in an AI coding assistant.

Understanding MCP

MCP (Model Context Protocol) is Anthropic's open standard for connecting large language models to external tools, data sources, and other AI agents. An MCP adapter acts as the translation layer — the "driver" that bridges the gap between an LLM and the services it needs to interact with.

MCP was adopted quickly by the development community, which is precisely why its security posture matters so much. Developers reach for MCP adapters to add AI integration capabilities without writing low-level connection logic. In doing so, they inherit whatever security properties — or vulnerabilities — exist in the protocol and its official implementations.

The risk profile of a compromised MCP is qualitatively different from a compromised API endpoint.

A successful attack on an MCP server can expose

- **Customer records and private conversations**
- **Source code repositories**
- **Internal databases and file systems**
- **API keys and authentication credentials**
- **The ability to deploy malicious workloads (e.g., crypto miners) on the victim's infrastructure**

We set out to find these weaknesses before attackers did — and to give the community, and Anthropic, the clearest possible picture of what needs to change.

Core Research:
The Root Vulnerability

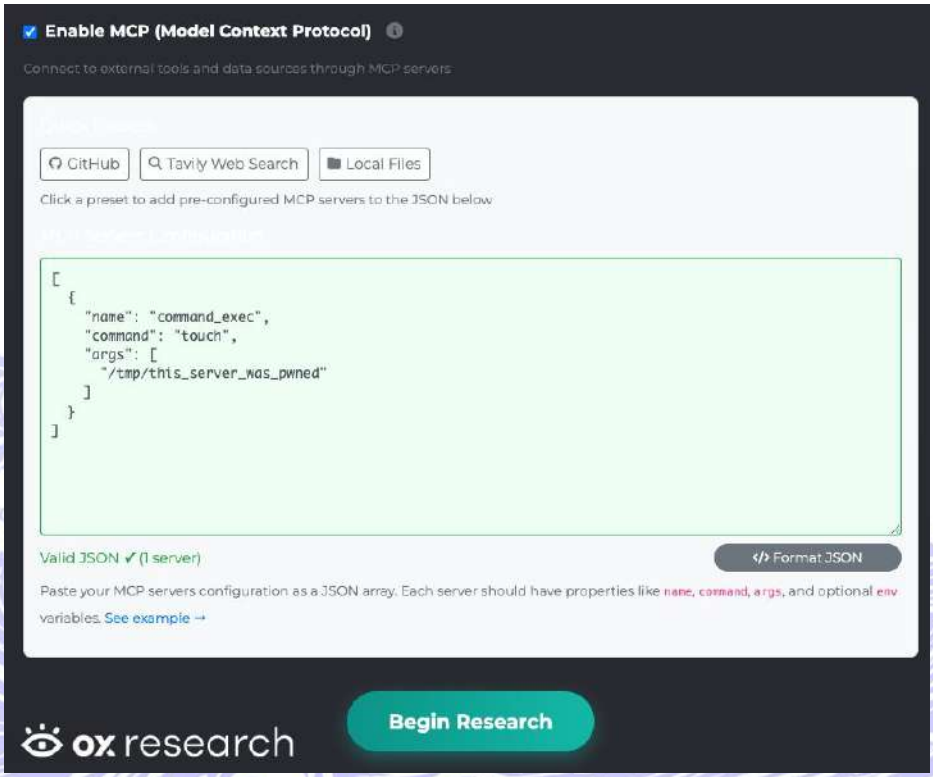
How It Started: GPT Researcher

In November 2025, while reviewing AI and LLM-related attack surfaces, the team analyzed GPT Researcher — an open-source project with over 25,000 GitHub stars that gives LLMs capabilities including RAG, deep research, and web browsing.

One feature stood out immediately: the ability to configure a custom STDIO MCP server, where the user supplies the command and arguments.

Testing revealed that any OS command passed through this interface would execute on the server — even when the fake MCP server failed to start. The error was returned to the user; the command ran anyway.
To be clear: this should never happen.
Running an arbitrary command → complete control of the server.

Exploiting GPT Researcher MCP configuration page with malicious input



Core Research

Tracing the Root Cause

The initial assumption was that the vulnerability lived in LangChain's langchain-mcp-adapters module, which GPT Researcher uses downstream. The MultiServerMCPClient function accepts user-supplied configuration and passes it directly to command execution with no filtering.

We contacted LangChain about this issue, as it enables command execution through their transport layer.

LangChain's response was: this is expected behavior. Developers are responsible for their own sanitization.

But something was still missing, we saw the commands executing but we didn't see the actual subprocess execution - and then we found it - the actual code doesn't lie inside LangChain - it lies inside modelcontextprotocol, the original MCP implementation code by Anthropic. It's not just a python issue, **it's the whole architecture of MCP that gives the same ability to execute commands directly on the underlying OS, as part of the expected behavior.**

Achilles' heel: Anthropic's MCP running process execution logic

```python
async def _create_platform_compatible_process(
    command: str,
    args: list[str],
    env: dict[str, str] | None = None,
    errlog: TextIO = sys.stderr,
    cwd: Path | str | None = None,
):
    """Creates a subprocess in a platform-compatible way.

    Unix: Creates process in a new session/process group for killpg support
    Windows: Creates process in a Job Object for reliable child termination
    """
    if sys.platform == "win32":  # pragma: no cover
        process = await create_windows_process(command, args, env, errlog, cwd)
    else:  # pragma: lax no cover
        process = await anyio.open_process(
            [command, *args],
            env=env,
            stderr=errlog,
            cwd=cwd,
            start_new_session=True,
        )

    return process
```

ox research

We reported this to Anthropic - and got the same answer as we got from LangChain.
This is the expected behavior.

We had a hypothesis that this is not just one issue, this architectural “expected behavior” is actually a widespread problem around multiple projects, from OSS to IDEs, everything that connects user input to MCP configuration.
About a week after reporting to Anthropic, and without notifying us - they released an updated security policy that states that the use of MCP adapters and specifically STDIO ones should be with caution. This change didn’t fix anything, it just made it more clear that Anthropic stance is on letting developers secure their own code, instead of securing their infrastructure.

Core Research

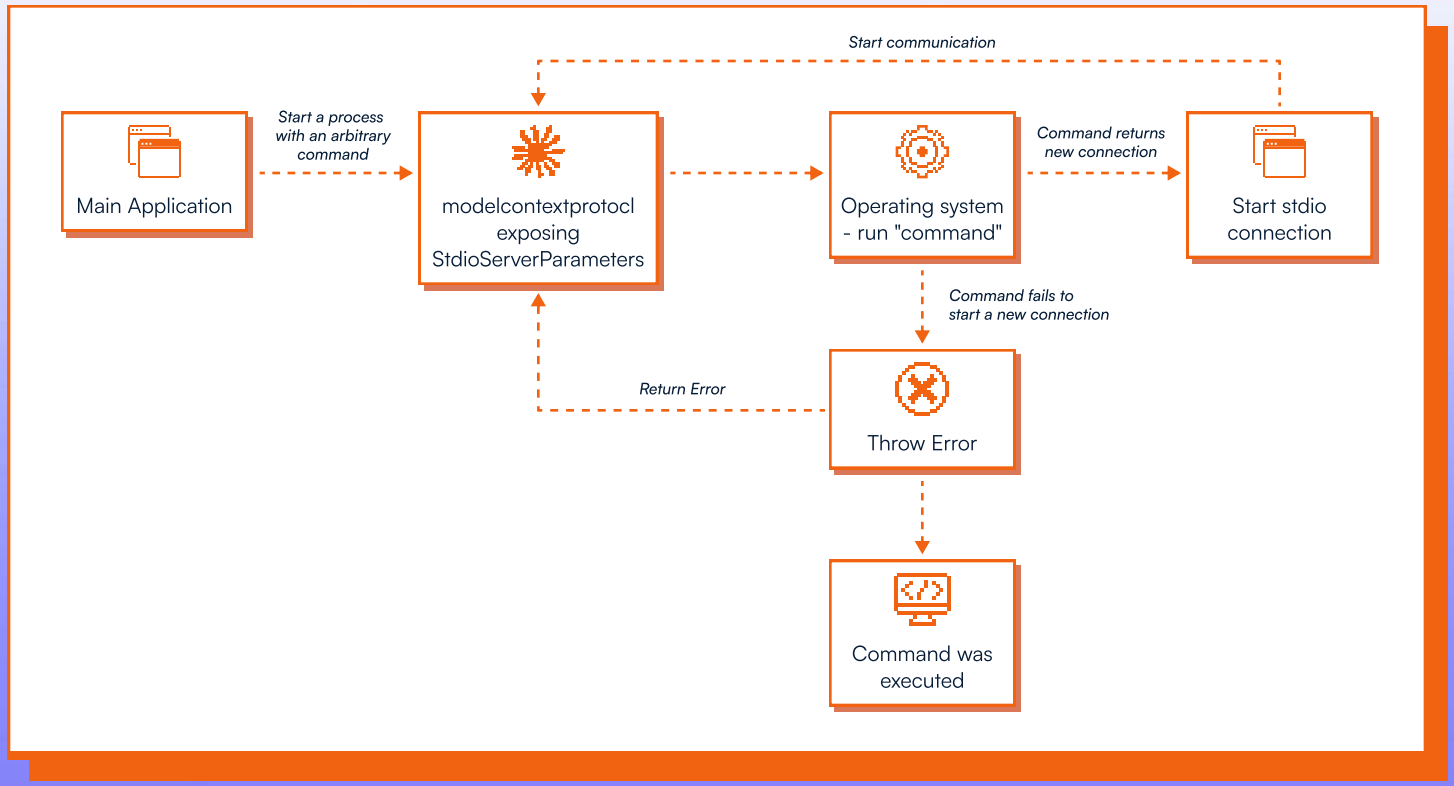
The Supply Chain Cascade

A seemingly unimportant detail in Anthropic's modelcontextprotocol project is the basis of a widespread supply chain compromise affecting millions of users, either directly or indirectly. MCP adapters give developers the ability to run any custom local MCP server on their machine, if the command starts an MCP server on stdio - the code returns success and the communication between the MCP and server starts.

But if the command fails, it just returns an error - the problem is that the command is still executed in this "return error" scenario. Abusing this logic is the basis of an arbitrary command execution vulnerability we found in many projects directly using Anthropic's MCP.

MCP STDIO Process Execution Behavior

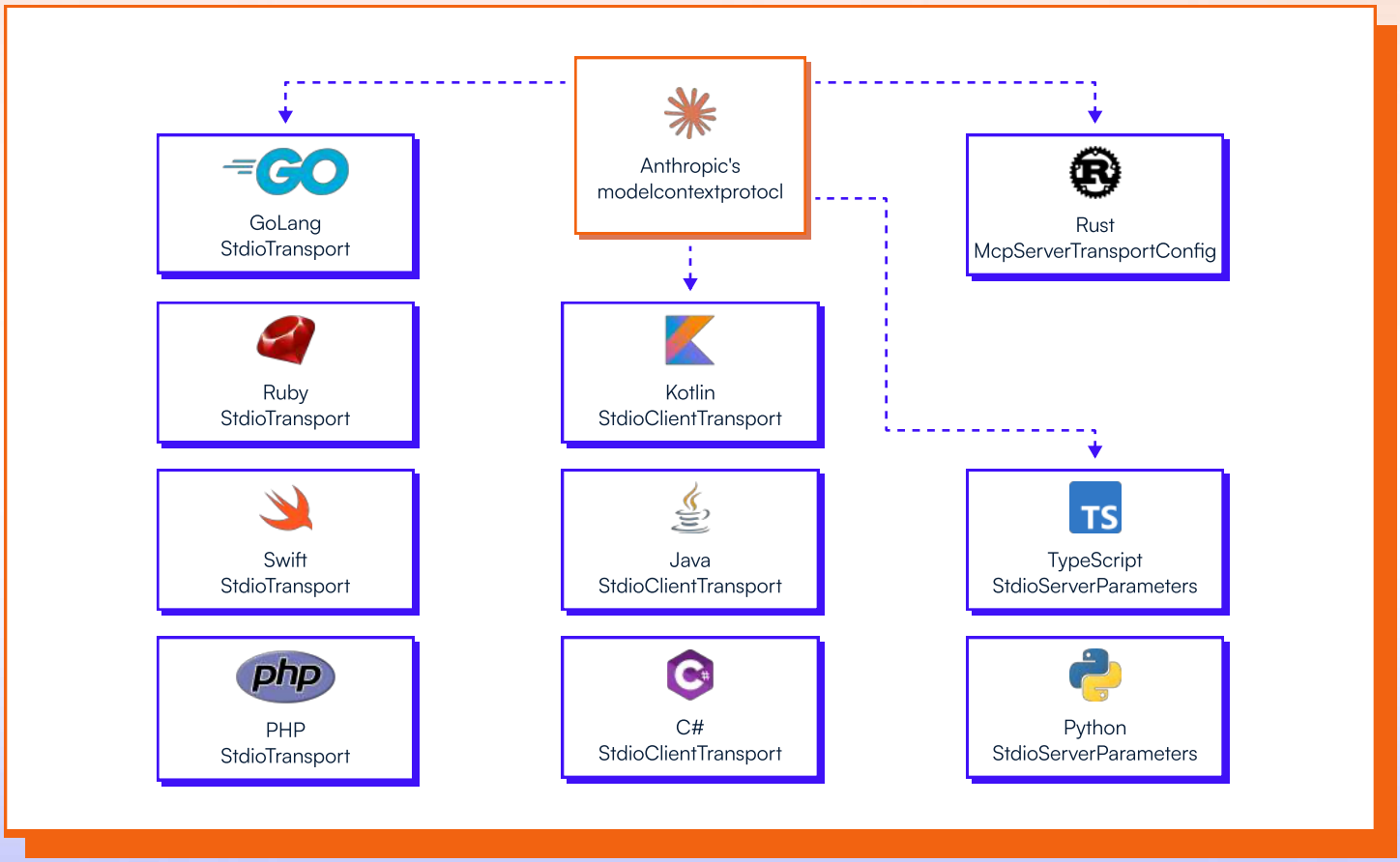
When a command is received to start a process there's no validation what that command is allowing an arbitrary command to be executed instead of a command that explicitly starts an MCP STDIO server.



Core Research

The downstream vulnerable by design functionality is found in all supported programming languages - Python, TypeScript, Java, Kotlin, C#, GoLang, Ruby, Swift, PHP & Rust.

The downstream vulnerable functions affects the MCP source code on all languages with different naming conventions and code signatures for each one.



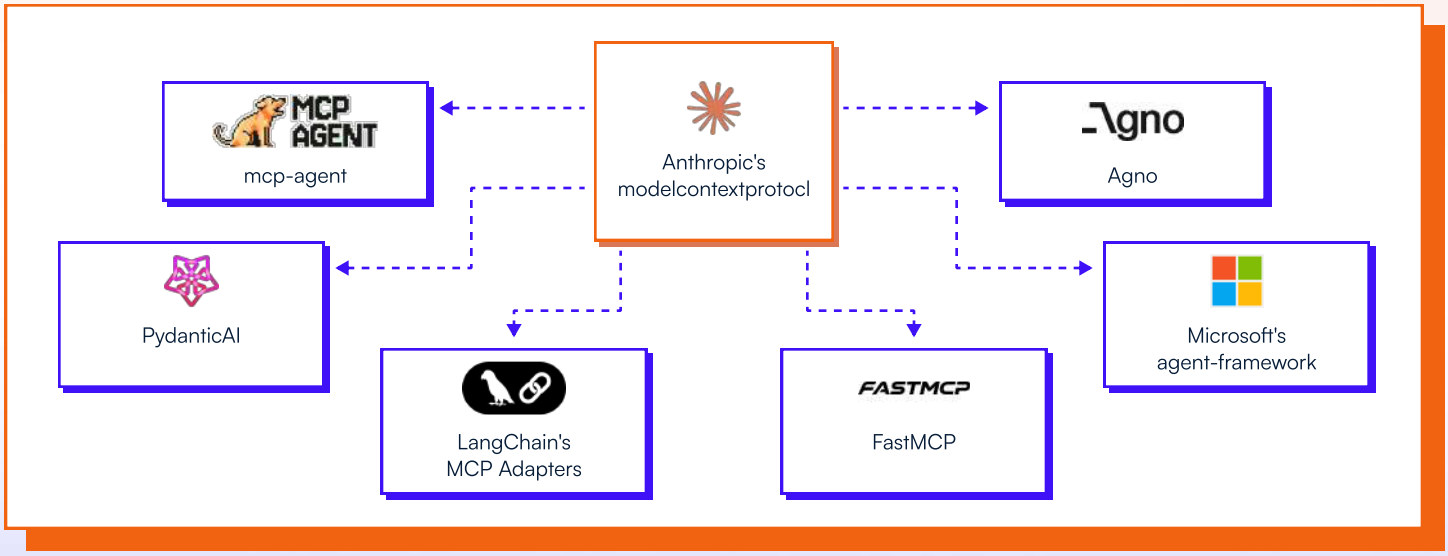
Affected Official MCP Repositories (Anthropic)

- modelcontextprotocol/python-sdk
- modelcontextprotocol/rust-sdk
- modelcontextprotocol/typescript-sdk
- modelcontextprotocol/php-sdk
- modelcontextprotocol/java-sdk
- modelcontextprotocol/go-sdk
- modelcontextprotocol/csharp-sdk
- modelcontextprotocol/kotlin-sdk
- modelcontextprotocol/ruby-sdk
- modelcontextprotocol/swift-sdk

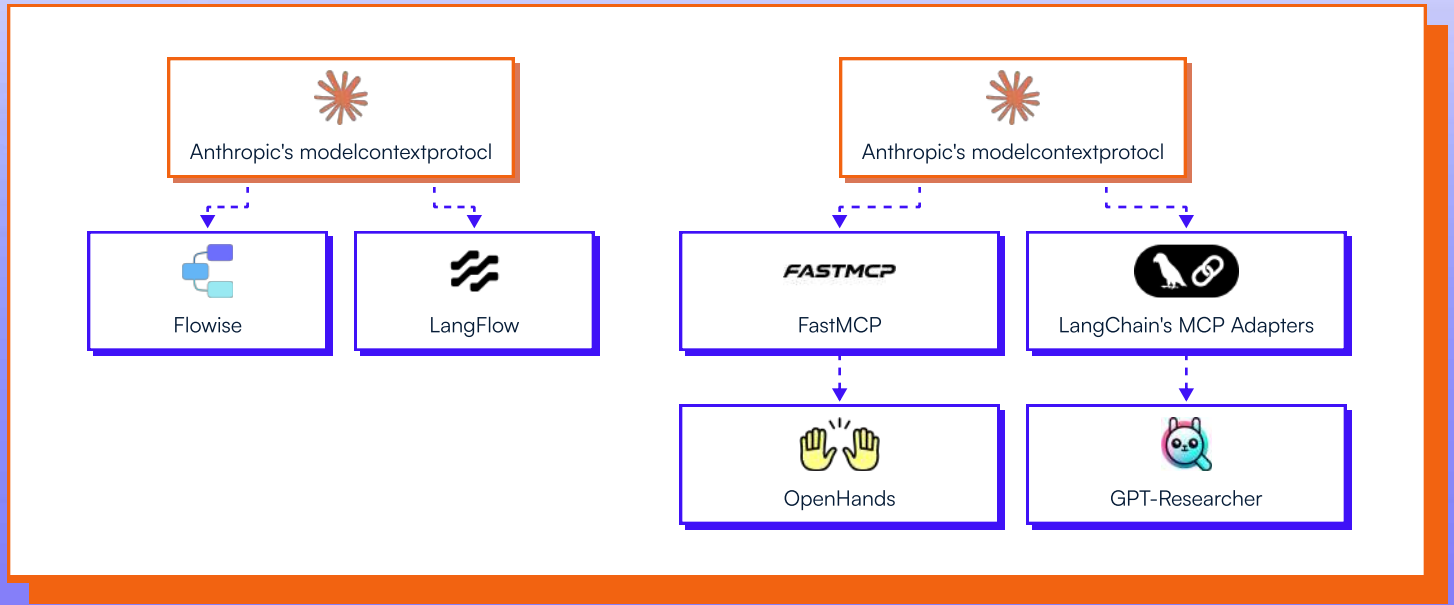
Affected MCP functionality providers

This issue affects all MCP functionality providers, such as LangChain's MCP adapters, FastMCP, Browser-Use, and other popular OSS projects developers use to add MCP communication logic to their projects - so OSS projects are either directly using Anthropic's modelcontextprotocol project, or implementing the same behavior using a different project that's directly importing from Anthropic.

The same MCP functionality is being exposed by different MCP frameworks, which use the Anthropic modelcontextprotocol package under the hood, as a direct dependency.



These are examples of a direct use of Anthropic's MCP, and an indirect use of it, using another MCP transport packge in the middle



Core Research

Vulnerable Through MCP Marketplaces

The attack does not require a dependency chain. It can be delivered directly through MCP marketplaces - directories where developers browse and install MCP server configurations into their tools.

We analyzed 11 popular MCP marketplaces and submitted a proof-of-concept MCP with arbitrary command execution capabilities to each one. We deliberately didn't want to upload malware to those marketplaces, so we created an MCP server that runs an arbitrary command which generates an empty file on the filesystem, instead of starting a real STDIO connection with the host.

Result: 9 out of 11 accepted it without challenge.

The marketplaces that accepted our submission include platforms with hundreds of thousands of monthly visitors. A single malicious MCP entry in any of these directories could be installed by thousands of developers before detection — each installation giving an attacker arbitrary command execution on the developer's machine.

Core Research

The Marketplace Trial Balloon

Marketplace	Monthly Traffic (Dec '25)	Submission	Security Review	Result
ModelContextProtocol Registry	690K	GitHub/NPM	**Bypassed**	Live & Searchable
LobeHub	285K	GitHub Auth	**Bypassed**	Live & Searchable
Cursor Directory	170K	GitHub Auth	**Bypassed**	Live & Searchable
Smithery AI	159K	GitHub Auth	**Bypassed**	Live & Searchable
MCP Market	138K	**Anonymous**	**None**	Live & Searchable
MCP Servers	122K	**Anonymous**	**None**	Live & Searchable
gLama	73K	GitHub Auth	**Bypassed**	Live & Searchable
Pulse MCP	6K	**Anonymous**	**None**	Live & Searchable
mcp.so	104K	GitHub Auth	**Bypassed**	Uploaded (Shadowed)
Cline	90K	GitHub Auth	Pending	No Response
GitHub Registry	~1.5M	Managed	Protected	Could not submit*

* GitHub's managed environment was the only outlier, proving that security gating is possible.

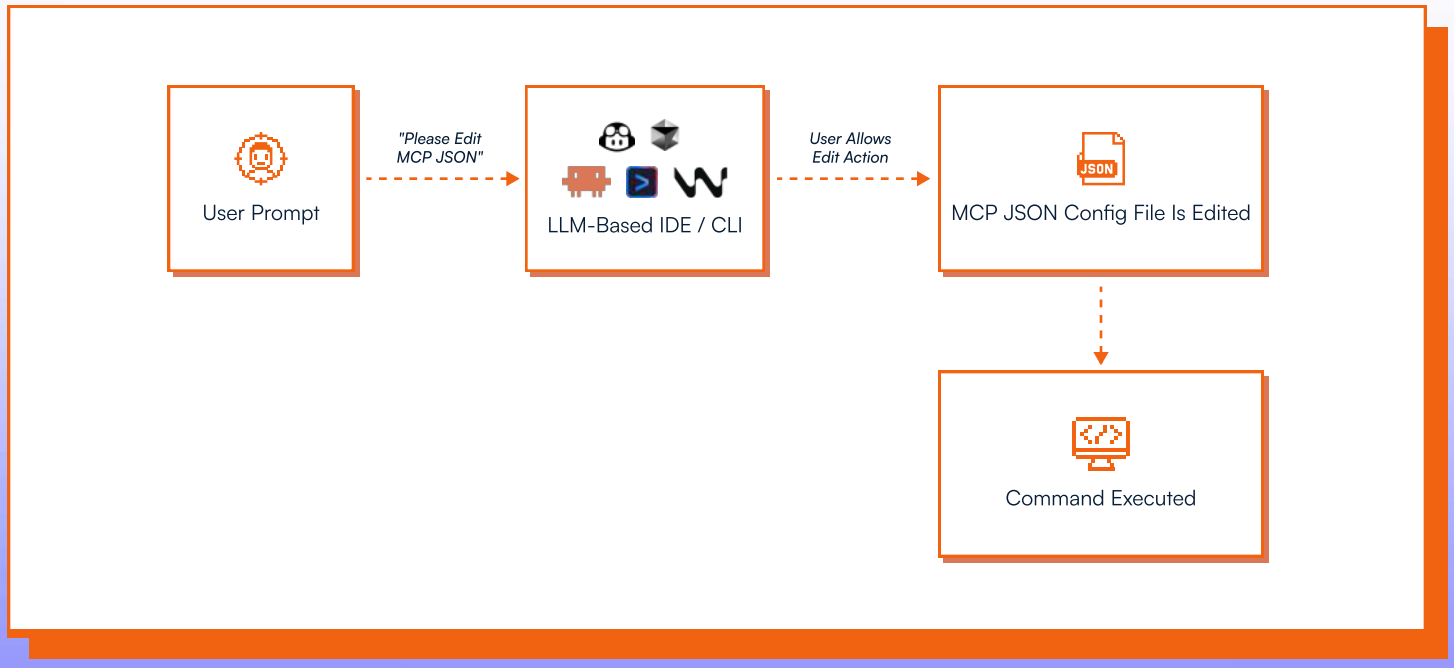
Core Research

IDEs: Turning Prompt Injection into System Access

A third attack vector bypasses servers entirely and targets developers directly through their AI coding environment. Cursor, VS Code, Windsurf, Claude Code, and Gemini-CLI all support MCP configuration — and all can be manipulated through prompt injection to modify the MCP configuration file, inserting arbitrary command execution.

We reported this vulnerability to Google's Gemini-CLI, Anthropic's Claude-Code, Cursor, Windsurf, Microsoft's VSCode.

In all cases, the user needed to approve the file edit, in some without warning, in some after two clicks of approval, and in some one click. Only one IDE had a direct prompt-to-execution with no user approval, which was Windsurf - which was assigned CVE-2026-30615.



Core Research

IDEs & Coding Assistance: Vendor responses

Google (Gemini-CLI):

Known issue, no CVE, no fix planned near-term

"We have analyzed your report and want to let you know that it is a duplicate of an existing bug that we are already tracking. For this reason, we are closing this issue."

Microsoft (VS Code):

By design

"Although your report included some good information, it does not meet Microsoft's requirement as a security vulnerability for servicing. For Copilot in VSCode to work, you need to trust the workspace, including any external links that you are requesting Copilot to visit. Additionally, the agent asks the user to specifically allow the editing of the file, which enables the execution."

Anthropic (Claude Code):

By design

"We do not consider this a valid security vulnerability as it requires explicit user permission for the file change where the user is given the opportunity to approve or deny the change."

Cursor:

By design — user must click accept on mcp.json edit

"Since users must now explicitly consent to modifications in sensitive directories, the current behavior is working as designed and represents the intended security control."

Windsurf:

No response

AI Assistant	Interaction (Default)	Interaction (Auto-Edit Enabled)	Visibility of Change	Security Warning
Windsurf	0 Clicks	Irrelevant	Hidden	None
Cursor	1 Click	1 Click	Bypassed	None
Claude Code	2 Clicks	1 Click	Partial	Rarely
Gemini-CLI	1 Click	0 Clicks	Visible	None
GitHub Copilot	1 Click	0 Clicks	Visible	Explicit

Online

Read Core Research Technically explained

Online

Read Full Technical Advisory

Case Studies:
Real-World Exploitation

LettaAI - Authenticated RCE on Production Platform

Letta AI (app.letta.com) is a platform for building stateful AI agents with memory capabilities. Its UI exposes MCP server configuration to authenticated users, but only offers Streamable HTTP and SSE connection types in the interface.
Reviewing the source code revealed a third accepted type: STDIO — with no server-side enforcement preventing its use. We intercepted the outbound "Test connection" request using a simulated man-in-the-middle approach, replaced the streamable-http configuration with a valid STDIO payload, and achieved arbitrary command execution on Letta AI's production servers.

Watch PoC

https://youtu.be/l_IMS-oLPCo

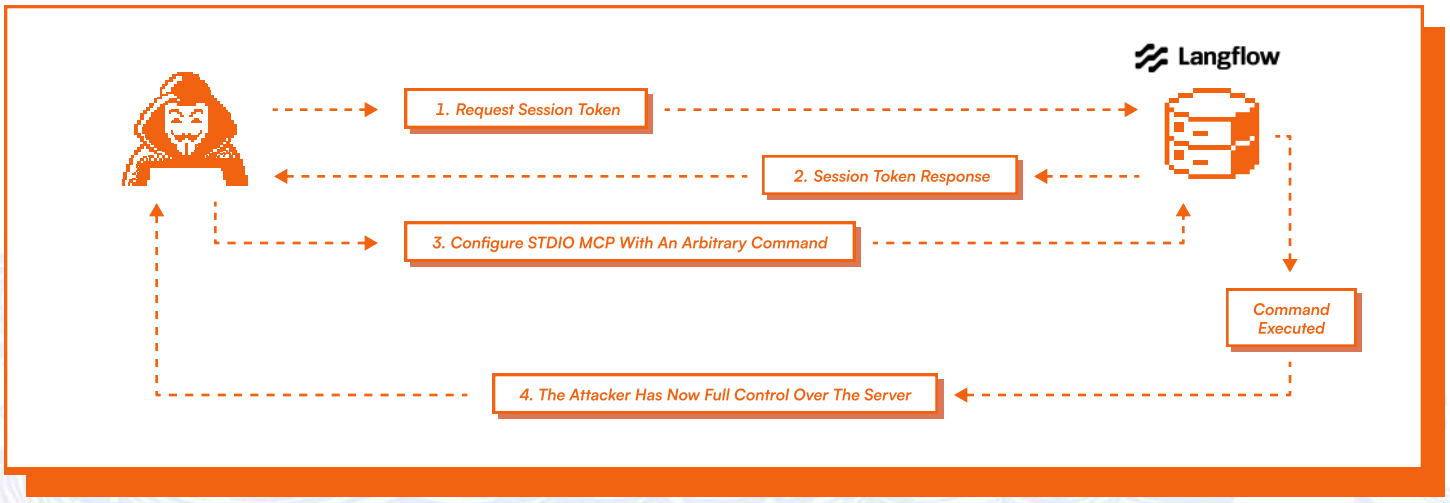
Case Studies

LangFlow - Unauthenticated RCE at Scale

LangFlow is an open-source research automation framework which is owned by IBM. While designed for local deployment, over 915 publicly accessible LangFlow instances were found on Shodan.
LangFlow's Settings interface exposes direct MCP configuration, including STDIO type, with no authentication required. A session token is available to any unauthenticated user, and can be obtained by either opening the Web-GUI or sending a simple network request. An attacker can weaponize this by sending a request to obtain the session token, followed by a crafted MCP configuration network request to the same LangFlow instance - achieving full server takeover, data exfiltration, and more, without ever logging in.

The code execution chain: src/lfx/src/lfx/base/mcp/util.py reads user configuration → passes to Anthropic's StdioServerParameters → which calls the subprocess command execution.
This issue was disclosed to LangFlow on Jan 11, 2026, and we repeatedly tried to contact LangFlow's maintainers in various ways throughout January to March, this issue was acknowledged directly only at Mar 18, 2026 inside our GHSA report.

Attacker finds a publicly available LangFlow server



Watch PoC

https://www.youtube.com/watch?v=fydGGm1Es-Y

Case Studies

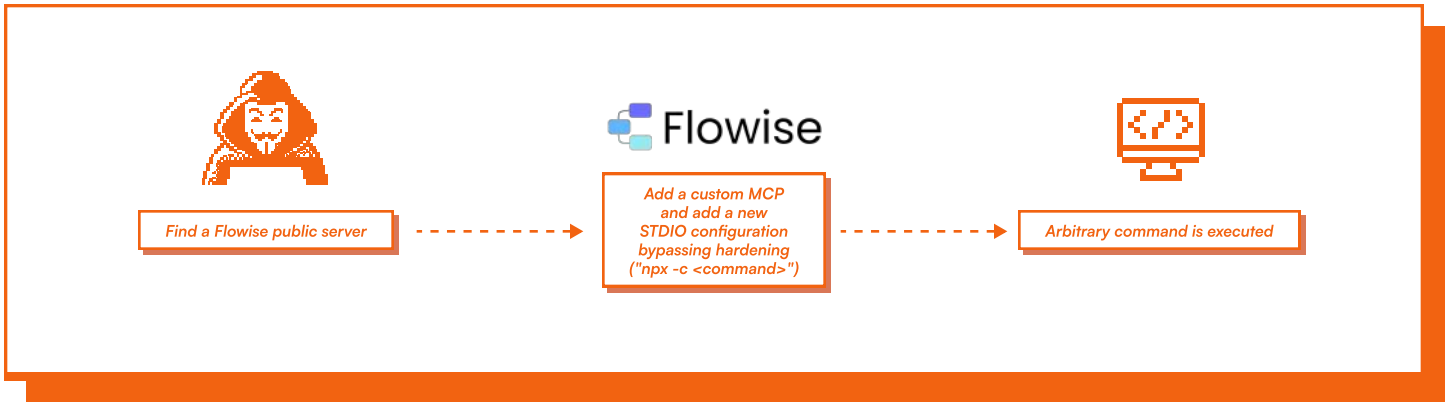
Flowise - Hardening Bypass

Unlike most affected platforms, Flowise was aware of the STDIO risk and implemented input validation: only specific commands were permitted, and special characters were stripped.

We bypassed their controls in a single step, using npx's -c flag to pass arbitrary commands through an otherwise-allowed command:

```
// Flowise blocks this directly:
{ "command": "touch", "args": ["/tmp/pwn"] }
// Bypass via npx -c:
{ "command": "npx", "args": ["-c", "touch /tmp/pwn"] }
```

The bypass worked. This illustrates that ad hoc input filtering is an insufficient defense when the underlying architecture permits arbitrary subprocess execution.



Watch PoC

https://youtu.be/YjzlTfEa3Eg

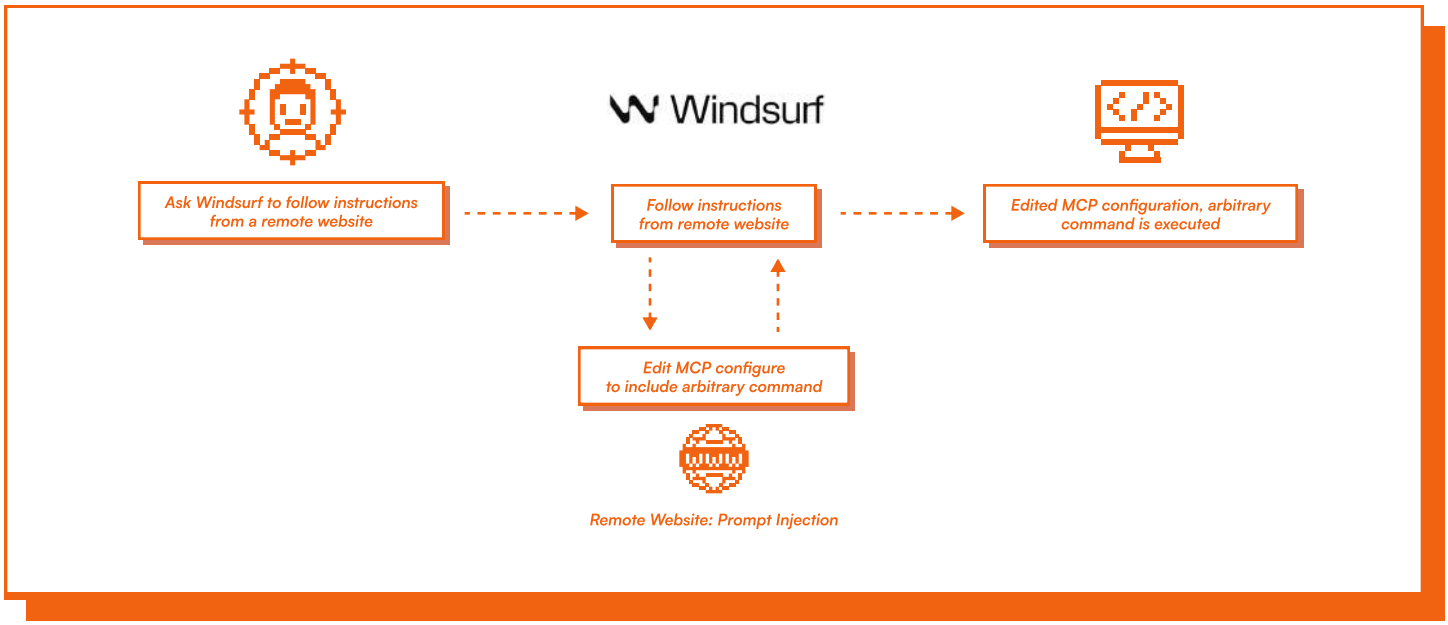
Case Studies

Windsurf - Prompt Injection to Local RCE (CVE-2026-30615)

Windsurf is an AI-powered IDE designed for developers. While it runs locally, its MCP configuration file (mcp.json) is writable by the AI agent - making it susceptible to prompt injection attacks that add malicious STDIO MCP entries.

Attack chain:

- Victim visits an attacker-controlled website and copies a prompt that appears legitimate
- The site serves different content to Windsurf's internal requests — injecting a malicious instruction
- Windsurf receives the malicious prompt and proposes edits to mcp.json — without showing the user what will change - and modifies the file.
- With no further user interaction; a new STDIO MCP entry is added and immediately executes its command on the victim's machine.



Watch PoC

https://youtu.be/psG7EpYq8Ks

Full Scope of Reported Vulnerabilities

Production Sites Directly Affected

- *Undisclosed 1*
- *DocsGPT*
- *Letta AI*
- *OpenHands*
- *Bisheng*
- *Undisclosed 2*

Libraries Exposing STDIO MCP Adapters

- *Upsonic*
- *AWSLabs — run-model-context-protocol-servers-with-aws-lambda*
- *Nvidia — NeMo-Agent-Toolkit*
- *Browser Use*

Open-Source Projects with Publicly Deployed Vulnerable Servers

- *LangFlow*
- *GPT Researcher*
- *Undisclosed 3*
- *Flowise*
- *LiteLLM*
- *LangBot*
- *Agent Zero*
- *Fay: Digital Human Framework*
- *Jaaz (jaaz.app)*
- *Promptfoo*
- *Langchain-Chatchat*

IDEs and CLIs

- *Gemini-CLI (Google)*
- *Claude Code (Anthropic)*
- *GitHub Copilot*
- *Cursor*
- *Windsurf*

The MCP Vulnerability Landscape

The following table summarizes the critical and high-severity vulnerabilities identified during this research. All vulnerabilities trace back to the root "Expected Behavior" of the Model Context Protocol's STDIO implementation.

While individual vendors have issued patches or acknowledged "by design" risks, the structural vulnerability remains unaddressed in the root MCP protocol maintained by Anthropic.

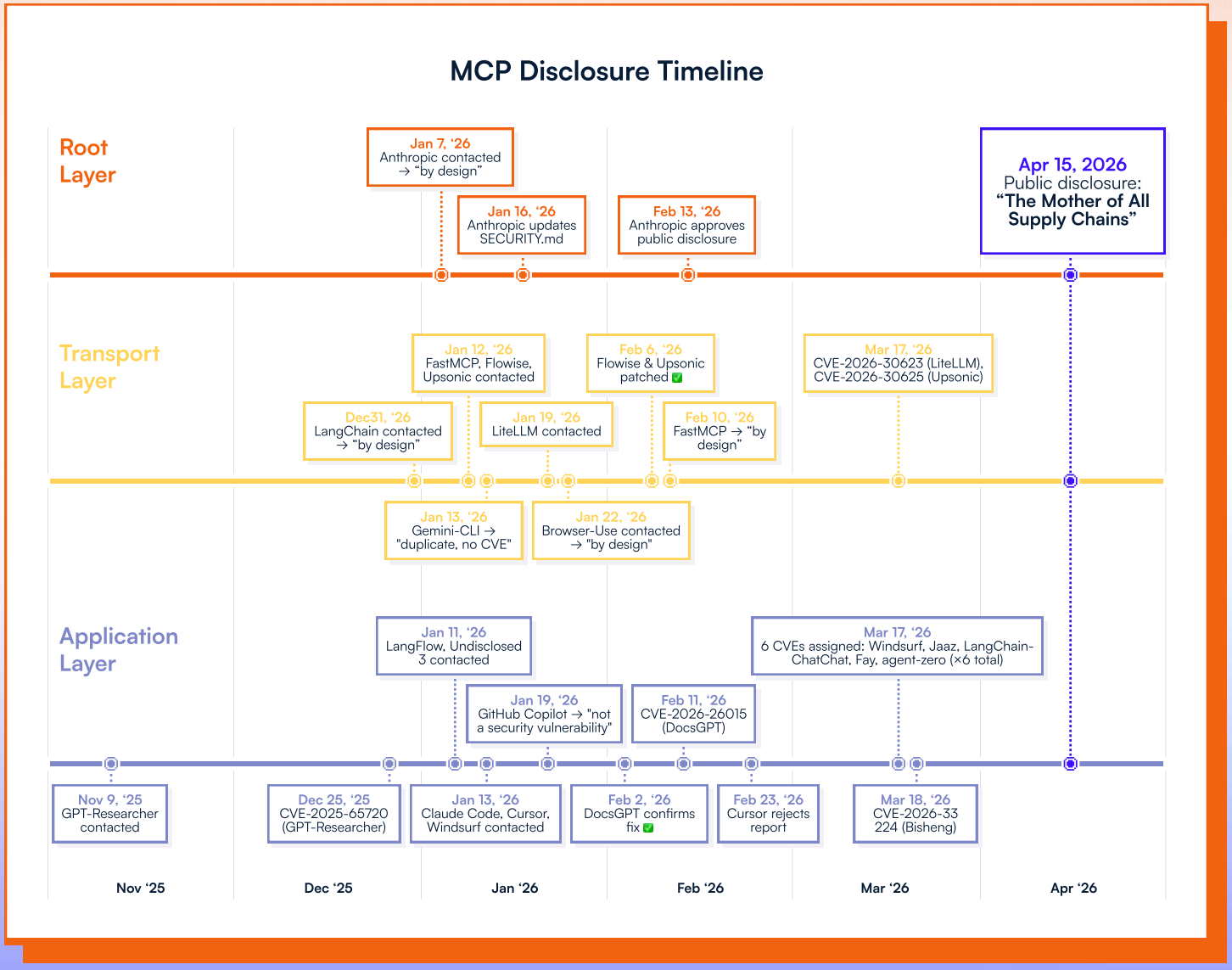
Family	CVE ID	Product	Attack Vector	Severity	Status
1. Direct UI Injection	In Progress	LangFlow	Unauthenticated UI injection via token	Critical	Reported
	CVE-2025-65720	GPT Researcher	UI injection / reverse shell	Critical	Reported
	CVE-2026-30623	LiteLLM	Authenticated RCE via JSON config	Critical	Patched
	CVE-2026-30624	Agent Zero	Unauthenticated UI injection	Critical	Reported
	Unassigned	LangBot	Unauthenticated UI injection	Critical	Reported
	Unassigned	Undisclosed 3	Authenticated UI injection	Critical	Patched
	CVE-2026-30618	Fay Framework	Unauthenticated Web-GUI RCE	Critical	Reported
	CVE-2026-33224	Bisheng	Authenticated UI injection (Open Registration)	Critical	Patched
	CVE-2026-30617	Langchain-Chatchat	Unauthenticated UI injection	Critical	Reported
	CVE-2026-33224	Jaaz	Unauthenticated UI injection	Critical	Reported
	In Progress	Undisclosed 1	Unauthenticated UI injection	Critical	Reported
	No CVE Issued	Firebase Studio	Authenticated STDIO injection	Critical	Reported
	No CVE Issued	promptfoo	Unauthenticated UI injection	Critical	Reported
	No CVE Issued	OpenHands	Unauthenticated UI injection	Critical	Reported

The MCP Vulnerability Landscape

Family	CVE ID	Product	Attack Vector	Severity	Status
2. Hardening Bypass	CVE-2026-30625	Upsonic	Allowlist bypass via npx/npm args	High	Warning
	GHSA-c9gw-hvqq	Flowise	Allowlist bypass via npx -c	High	Patched
3. Prompt Injection	CVE-2026-30615	Windsurf	Zero-click prompt injection to local RCE	Critical	Reported
	No CVE Issued	Cursor	Prompt injection (requires click)	Critical	By Design
	No CVE Issued	Claude Code	Prompt injection (requires click)	Critical	By Design
	No CVE Issued	Gemini-CLI	Prompt injection (requires click)	Critical	By Design
	No CVE Issued	GitHub Copilot (VSCode)	Prompt injection (requires click)	Critical	By Design
4. Hidden STDIO Configuration	CVE-2026-26015	DocsGPT	MITM transport-type substitution	Critical	Patched
	Unassigned	LettaAI	MITM transport-type substitution	Critical	Patched
	In Progress	Undisclosed 2	MCP STDIO Injection Via Agent Framework	Critical	Reported
5. Root & Transport Layer	No CVE Issued	MCP (Anthropic)	Injection via the MCP transport layer	Critical	Reported
	No CVE Issued	LangChain	Injection via the MCP transport layer	Critical	Reported
	No CVE Issued	FastMCP	Injection via the MCP transport layer	Critical	Reported
	No CVE Issued	browser-use	Injection via the MCP transport layer	Critical	Reported
	No CVE Issued	awslabs Run MCP AWS Lambda	Injection via the MCP transport layer	Critical	Reported
	No CVE Issued	NVIDIA NeMo Agent Toolkit	Injection via the MCP transport layer	Critical	Reported

The Road to Disclosure

Between **November 2025 and April 2026**, OX Security conducted over **30 responsible disclosure processes**. The timeline below tracks the parallel efforts to secure the **Root** (Protocol), **Transport** (Frameworks), and **Application** (End-User Services) layers.



Root & Transport Layer:
Vendor & Maintainer Responses

The following table summarizes the critical and high-severity vulnerabilities identified during this research. All vulnerabilities trace back to the root "Expected Behavior" of the Model Context Protocol's STDIO implementation.

While individual vendors have issued patches or acknowledged "by design" risks, the structural vulnerability remains unaddressed in the root MCP protocol maintained by Anthropic.

Anthropic:
"This is an explicit part of how stdio MCP servers work and we believe that this design does represent a secure default."

LangChain:
"It is the responsibility of the application author to validate and sanitize inputs from untrusted sources."

FastMCP:
"We don't consider this a vulnerability — stdio transport spawns a subprocess by design, per the MCP specification."

Proposed Remediation

The current implementation of the Model Context Protocol places the entire burden of security on the downstream developer - a structural failure that guarantees vulnerability at scale.

To protect the ecosystem, OX Security proposes a fundamental hardening of the MCP SDKs.

1. Architectural Deprecation of Unsanitized STDIO Connections

The pattern of allowing user-supplied strings to flow directly into a shell execution environment is an anti-pattern that should be deprecated.

The Fix

MCP implementations should transition to a Manifest-Only model - adapters should only execute pre-defined server aliases defined in a static, developer-controlled configuration file, rather than accepting arbitrary command strings.

Impact

Eliminates the possibility of unauthenticated users injecting payloads via network requests or UI fields.

2. Protocol-Level Command Sandboxing

If the flexibility of arbitrary command execution is required, it must not be open by default.

The Fix

Anthropic's SDKs should implement a command allowlist. High-risk binaries — sh, bash, powershell, curl,rm — should be blocked unless explicitly permitted in a high-privilege configuration.

The Failed-Start Guard

If a process fails to initialize as a valid MCP server, all associated child processes should be immediately and recursively terminated before any code execution can persist. This directly closes the root vulnerability documented in this research.

3. "Dangerous Mode" Explicit Opt-In

Currently, developers unknowingly inherit RCE-by-design. Security should be an intentional choice, not a silent default.

The Fix

Introduce a mandatory flag for any STDIO configuration utilizing dynamic arguments: allow_unsafe_local_execution: boolean.

Impact

Linters can flag its absence during development. Security teams can grep for it during audits. IDEs should trigger a high-severity warning when an AI agent attempts to modify a config containing this flag.

4. Marketplace Verification Standards

Nine out of 11 registries accepted our proof-of-concept potentially malicious MCP without challenge. The ecosystem needs supply-chain metadata standards.

The Fix

Marketplaces should require a standardized security manifest - servers declare exactly which resources they access (filesystem, network, environment variables) and are signed by a verified developer identity.

Summary

Mitigation	Implementation Level	Protection Level
Manifest-Only Execution	Protocol/SDK	Critical: eliminates the primary RCE vector
Command Allowlisting	SDK	High: prevents common shell exploitation
Unsafe Opt-In Flag	SDK/Developer	Medium: increases visibility and auditability
Marketplace Signing	Ecosystem	High: prevents malicious distribution

Any of these fixes, implemented at the Anthropic modelcontextprotocol level, would have propagated protection instantly to every downstream library and project — protecting millions of users without requiring individual patches across hundreds of repositories.

Conclusion:
An Opportunity to Truly Own the Stack

MCP is a genuinely useful protocol with rapid adoption - which is precisely what makes its security posture so consequential. The vulnerability documented here is not a fringe edge case. It is a repeating pattern, reproduced across dozens of independent projects, in every programming language Anthropic supports, and exploitable through multiple attack paths simultaneously: network requests, direct UI configuration, marketplace distribution, and prompt injection.

What made this a supply chain event rather than a single CVE is that one architectural decision, made once, propagated silently into every language, every downstream library, and every project that trusted the protocol to be what it appeared to be. Not a spectacular zero-day, but a quiet assumption, repeated everywhere, that no one thought to question.

Shifting responsibility to implementers does not transfer the risk. It just obscures who created it.

Anthropic has the ability - and, we would argue, the responsibility - to make MCP secure by default. One architectural change at the protocol level would have protected every downstream project, every developer, and every end user who relies on MCP today. That's what it means to own the stack.

OX Security Research Team

Notes

Following this research, OX Security has shipped protections across its platform.

VibeSec / AI-Generated Code OX now detects improper use of STDIO-based MCP configurations in AI-generated code, blocking patterns where user input flows directly into STDIO MCP configuration — the root pattern documented in this research.

OX Security Platform OX now flags existing STDIO MCP configurations in customer codebases where user input is present, surfacing these as actionable findings for remediation.