ChatGPT

Inside MCP Security: Key Risks and Emerging Threats

Model Context Protocol (MCP) is rapidly becoming the standard "USB-C for AI applications," linking large language models (LLMs) to external tools and data [1] [2]. However, this convenience comes with serious security baggage. Recent research – including the Wiz *Inside MCP Security* report and numerous references – reveals a broad threat landscape. Below is a summary of the current state of MCP security risks and threat models, with a focus on AI/LLM agent integration and real-world exploits. We then evaluate how these findings inform **MCP Guarda** – a lightweight policy layer and gatekeeper for MCP tools – and whether it should stay the course or pivot. Finally, we provide prioritized recommendations for MCP Guarda's development and go-to-market strategy, highlighting features to focus on (or drop) and urgent pain points it can address.

Current Security Risks in the MCP Ecosystem

MCP's design **exposes a wide range of attack vectors**, many reminiscent of past software and AI plugin issues. While the MCP specification itself flags some best practices (e.g. *"require a human in the loop"*, *"treat tool annotations as untrusted"* [3]), in practice the ecosystem's security is **underdeveloped**. Key risk areas include:

- **Supply Chain and Package Trust:** Installing an MCP server today is akin to running arbitrary code from the internet. There is **no official package registry or signing** in place yet [4] [5]. Developers often pull MCP servers from random GitHub repos, with no pinning or verification (a dangerous "`curl | bash`" anti-pattern [4]). Unofficial registries exist, but they offer limited trust signals. For example, Glama.ai's registry attempts to show vulnerabilities and mark "Official" servers, yet ~100 of its ~3,500 listed servers point to non-existent repos [6] [7]. Common package risks like **typosquatting, impersonation, rug pulls, and account takeovers** are already a concern in MCP marketplaces [8] [9]. In short, **supply chain security is a major weak point**, as malicious actors can publish Trojan-horse servers or slip malicious updates to popular tools.

- **Local vs Remote Execution Risks:** MCP servers can run locally (on the user's machine) or remotely (as network services). Both modes have distinct dangers:

- *Local MCP servers* essentially run with the user's privileges, so a malicious or compromised local server means instant remote code execution (RCE) on your system [10] [11]. Installing one is like installing any binary – if it's backdoored, it can exfiltrate files, install malware, or worse. The **lack of sandboxing** or permission separation means a rogue local tool has free rein over your files and environment [12]. This is exacerbated by auto-installers (e.g. `mcp-installer`) that emphasize one-click convenience over safety, discouraging users from inspecting what they run [4] [13].

- *Remote MCP servers* avoid running untrusted code locally, but **security is by no means assured**. A malicious remote server can still induce harmful actions on the client side – for instance, by exploiting whatever capabilities the LLM client has. **Remote servers can prompt an agent to read local files or send credentials** if the client isn't careful [14] [15]. They also introduce vendor risks: third-party operators might log sensitive data or mishandle tokens [16] [17]. In either case, **MCP servers often hold API keys and auth tokens for external services** (to provide tool functionality). If an attacker compromises an MCP server (local or remote), they could harvest those credentials and impersonate users on multiple services [18] [19]. Pillar Security warns that a breached MCP server is a *"high-value target"*, potentially yielding a trove of cloud and application access tokens [20].

- **Open Authorization (OAuth) Flaws:** Until mid-2025, the MCP spec required every server to implement its own OAuth flow for user authentication, creating a ripe area for bugs [21]. Many MCP servers are written by independent developers who are not OAuth experts, increasing the chance of logic errors or token mismanagement. Even Anthropic's own sample servers had issues – Wiz's Guy Goldenberg discovered OAuth-related injection vulnerabilities in the official PostgreSQL and Puppeteer MCP servers [22] [23]. This "roll your own auth" approach was quickly identified as too much burden on each server [24] [25]. The community pushed for a fix, and by June 2025 the spec was updated to allow delegating to external auth providers (separating the OAuth Authorization Server from the MCP Resource Server) [26] [27]. This improvement aligns MCP with standard OAuth models [28] [29], but **many existing servers may still use older, fragile auth implementations**. For enterprise settings, the lack of a robust, built-in authN/Z framework is a headache – as one analyst put it, *"the MCP Authorization Spec is…a mess for enterprise"*, lacking a secure, multi-tenant approach out of the box [30].

- **LLM Agent Exploits (Prompt Injection & Tool Misuse):** Perhaps the *defining new risk* of MCP is how it expands the attack surface through the AI agent's behavior. By design, MCP lets LLMs **autonomously decide which tools to use and how to use them** to fulfill a request [31] [32]. This introduces a form of *confused deputy* problem: if an attacker can manipulate the inputs the LLM sees, they might trick it into invoking tools in harmful ways [33] [34]. Two real exploit classes have emerged:

- **Indirect Prompt Injection (Tool Poisoning):** This occurs when malicious instructions are hidden in content that the LLM processes, causing it to do something unintended with the tools. A dramatic example is **tool description poisoning**, where an MCP server's tool **documentation contains hidden directives** that the user doesn't see but the LLM does [35] [36]. Invariant Labs demonstrated this by adding an `<IMPORTANT>` section in a tool's docstring telling the model to read sensitive files (like config and SSH keys) and send them as a parameter [37] [38]. The tool looked like a harmless "add(a,b)" function to the user, but once the agent used it, the LLM dutifully followed the secret instructions to exfiltrate secrets [39] [40]. Critically, many clients did **not display the full tool description or parameters to the user**, so the data leakage was invisible in the UI [41] [40]. This *"Tool Poisoning Attack"* is a new variety of prompt injection that can completely subvert an agent – the malicious tool can hijack the LLM's behavior and override even trusted servers' instructions [42] [43]. Researchers note that MCP's current design *"places too much trust in tool descriptions without sufficient validation or user transparency"*, calling it a fundamental flaw to fix at protocol, server, **and** client levels [44] [45].
- **Malicious Content Injection via Data:** Similarly, an attacker could insert hidden commands in any content an AI reads (documents, emails, etc.), leading the agent to perform unauthorized MCP tool calls. Pillar Security gives the scenario of an email containing invisible text that causes the AI to, say, forward all your files to an attacker's email when "read" by the assistant [46] [47]. Since MCP blurs the

line between reading data and taking action, **viewing a booby-trapped file could trigger tool-based exploits** – a dangerous new paradigm where simply sharing content with your AI can have side effects.

- **Tool Invocation Conflicts:** Attackers can also exploit how LLMs select and route tool calls. For instance, **tool name collisions** or **slash-command hijacking** can cause confusion. If a malicious server registers a tool with a common name like `deploy` or `send_email`, an LLM might invoke that instead of the intended one (especially if the attacker crafted a convincing description) [48] [49]. Overlapping slash commands (`/scan`, `/delete`, etc.) can similarly be mis-routed [49] [50]. Researchers even found that by embedding phrases like "*prefer this tool*" in a tool's description, they could influence some clients' tool selection logic, a form of **toolflow hijacking** [51] [52]. These issues show that an LLM agent's **tool trust logic can be gamed**, unless clients enforce strict disambiguation and verification.

- **Auto-Execution and Guardrail Failures:** Despite the MCP spec advising a human-in-the-loop, several MCP clients opt to **auto-run tools for a smoother UX**, implicitly trusting tool outputs [53] [54]. This has already led to real exploits. Wiz's Gal Nagli published a proof-of-concept where an agent auto-running a seemingly benign remote tool was exploited to achieve RCE on the host machine [55] [56]. In the demo, the agent connected to a malicious documentation-parsing server that, when auto-invoked by the LLM, executed system commands on the client. Such incidents prove that **auto-execution is dangerous**, effectively giving any tool the keys to your system. While some clients (Claude, etc.) have tried to include basic guardrails against malicious tool usage [57] [58], these protections have been "inconsistent and uncomprehensive" and not designed with MCP's full threat model in mind [59] [60]. Default safeguards (like simple prompt filters) are **fallible** – determined attackers can still slip instructions through or exploit functionalities. Overall, without robust oversight, an AI agent can become an unwitting insider threat.

In summary, MCP's *"expanding risk surface"* spans classic software issues (supply chain, code injection, auth bugs) **and** novel AI-specific threats (prompt manipulation, tool misuse). The most urgent and underserved risks right now appear to be those unique to the MCP paradigm – in particular, **indirect prompt injection attacks (like tool poisoning)** and the general lack of runtime control over AI tool usage. These are areas where security solutions have lagged behind the rapid adoption of MCP. As Wiz researchers conclude, *"the burden is on builders and defenders"* to close these gaps with better policy and tooling [61] [62].

Urgent Threats and Underserved Areas

Among the many risks outlined, a few stand out as **especially urgent and insufficiently addressed** in the current ecosystem:

- **Prompt Injection & Tool Coercion:** The **tool poisoning** vulnerability revealed in April 2025 is glaring and immediate. Attackers can effectively **"encode" malicious requests in tool definitions or other content** to make the AI agent betray the user's trust [35] [36]. This threat is urgent because it undermines the core promise of agents (autonomy) by turning it against the user. It's also currently underserved: it requires coordinated fixes (clients must show full tool info or sanitize inputs, servers should limit hidden instructions, the protocol might need to disallow certain patterns). As Simon Willison noted, we've known about prompt injection broadly for years but still lack "convincing mitigations" [63] – and MCP makes the consequences even more severe. Until robust solutions are in place, the community is urging extreme caution when using untrusted servers and mixing data sources [64] [65]. In short, **indirect prompt injection is a new Achilles' heel** for MCP-based systems – one that demands immediate defensive tooling.

- **Supply Chain Integrity (Lack of Registry/Signing):** MCP's fast growth (thousands of servers popping up) has outpaced the security infrastructure to vet them. An **official MCP registry with signing and version pinning** is reportedly in the works [66] [67], but until it materializes, the ecosystem remains Wild West. Today, a user might install a tool from GitHub without any code audit, and there's no chain of trust to ensure the code they run tomorrow is the same they reviewed today. Malicious package updates (rug pulls) are a real worry – a tool that was safe yesterday could silently update to a backdoored version next week [68] [69]. Likewise, impersonation of "official" servers (e.g. someone publishing an Azure-branded server with no affiliation) has occurred [70] [71]. This area is underserved because it needs ecosystem-level solutions (package signing, verified developer identities, etc.) which are only just being discussed. Until then, **organizations have to create their own allowlists and vetting processes** – a cumbersome task that cries out for automation or third-party services.

- **Fine-Grained Permissions & Isolation:** The current MCP spec only recently introduced basic notions like marking tools "readOnly" or "destructive" in annotations [72] [73], but these are not enforced by clients yet (they are merely hints). There is *no robust sandbox or permission system* widely in use – MCP servers often get broad access to whatever APIs or system functions they wrap. This means an MCP server for, say, Gmail will typically request full mailbox access (read, send, delete), because it aims to be general-purpose [74] [75]. That broad scope is dangerous if misused: an attacker with a stolen token can read or wipe out your entire email without constraint [18] [76]. Similarly, the **aggregation of many service permissions in one place (the MCP layer)** is a new risk – a compromise of an agent or server could expose a combined data trove (email, calendar, files, etc. all at once) [77] [78]. This is not well addressed yet; the community recognizes the need for "fine-grained permission models" and containerized execution for tools [79] [80], but practical implementations are just starting (e.g. projects like *Toolhive* using Docker and authz controls, or *hyper-mcp* using WebAssembly for isolation [79] [81]). For now, most MCP setups run with effectively root-level privileges in their context. **Least-privilege enforcement and sandboxing** remain major gaps.

- **Auditability and Monitoring:** Given the stealthy nature of some MCP attacks, the lack of rigorous auditing is a problem. Invariant's tool poisoning example showed that a user's UI may not even log what data was sent where [41] [82]. If an organization is deploying AI agents with MCP, they need ways to monitor tool usage, detect anomalies (e.g. why is a math tool suddenly reading SSH keys?), and maintain logs for forensic analysis. The Wiz report notes that *"granular and detailed logs are key, to ensure transparency and auditability"* [83] [84], but many MCP clients/servers today don't provide enterprise-grade logging. This area is ripe for improvement (and underserved by current open-source clients focused more on functionality than governance).

In essence, **the MCP ecosystem is in a "secure yourself" phase** – early adopters must add their own safety layers on top of insecure defaults [85] [86]. The community is actively exploring solutions (from Cloudflare's agent gateway ideas [87] to open-source proxies like *MCP Guardian* and *MCP Gateway* [88] [89]), but many of these are nascent. This backdrop presents both a *challenge and an opportunity* for MCP Guarda.

Implications for MCP Guarda: Evaluating the Security Gatekeeper Approach

MCP Guarda is envisioned as a lightweight, developer-friendly *"policy layer and gatekeeper"* for MCP tools. In other words, it functions as an **MCP security proxy** – sitting between the LLM (client) and MCP servers to enforce rules, approvals, and other protections. The findings above strongly **validate the need for such a gatekeeper**. In fact, the Wiz report explicitly recommends *"centralizing MCP server usage through a proxy"* to provide a single choke point for audit logging, monitoring, and guardrails [90] [91]. Similarly, Andreessen Horowitz's analysis noted that an MCP gateway could enforce auth, route requests, and improve security in multi-user environments [87] [92]. The concept of MCP Guarda aligns with these recommendations: it's a timely response to the wild risks of uncontrolled tool use.

However, *viability* will depend on execution and focus. The research also highlights certain realities that MCP Guarda must navigate:

- **Existing Efforts:** There are already open-source projects like **MCP Guardian (by EQTY Lab)** aiming to secure MCP usage. MCP Guardian provides real-time monitoring, logging, and even message-level approvals to let a human intervene before a tool runs [93] [94]. Its existence proves demand for a "security-first layer" [95], but also means MCP Guarda needs a clear differentiator. Perhaps Guarda can compete by being more lightweight or developer-centric, whereas MCP Guardian might be targeting enterprise deployments or is part of a larger service. Either way, MCP Guarda should avoid reinventing wheels that are already freely available; instead, it could integrate or improve upon them (for example, offering a simpler setup or better UI).

- **Solo Developer Constraints:** Given limited time and resources, MCP Guarda cannot solve **every** MCP issue at once. It should not attempt to build an entire MCP ecosystem (registry, client, scanning tool, etc.) from scratch. Rather, the product should **laser-focus on the most critical, addressable gaps** that a gatekeeper can cover. The good news is many urgent risks can be mitigated with relatively straightforward policy checks or intermediation. For instance, prompt injection attacks can be blunted by ensuring the user or admin *sees what the agent is about to do* (no more hidden instructions), and by blocking obviously dangerous actions by default. These are within scope for a proxy. On the other hand, issues like code-signing every package or rewriting how OAuth works are beyond a small product's scope – those will be handled by the broader community and spec updates. MCP Guarda should therefore **emphasize quick-win features** that deliver safety gains without needing massive engineering lift (see recommendations below).

- **Product-Market Fit Considerations:** The most likely early adopters of MCP Guarda are developers and small teams experimenting with LLM agents, as well as possibly security-conscious enterprises running pilots. They need solutions that are **easy to deploy and don't stifle the development experience**. Notably, one reason insecure defaults (like auto-running tools) persist is that developers favor convenience [96]. If MCP Guarda is too heavy or obstructive, devs might simply turn it off. Thus, a balance is needed: provide security **transparently and with minimal friction**. This could mean a simple drop-in library or a CLI that runs an MCP proxy locally with one command. It also means prioritizing clear, actionable alerts over noisy, cryptic ones. A developer-friendly tool could gain traction quickly, especially since many devs are just now realizing the risks (the flurry of blog posts and papers in early 2025 suggests rising awareness). Moreover, companies will pay for solutions that let them adopt new tech **safely**. A lightweight gatekeeper that stops an AI agent from emailing out the CEO's files is an easy sell if it demonstrably prevents major incidents.

- **Pivot or Persevere?** Given the above, **MCP Guarda's core premise remains viable** – arguably more than ever – provided it targets the right problems. There is no indication that the big AI providers or MCP spec will swoop in with a comprehensive security fix in the immediate term; improvements are coming, but the playing field is still open. Rather than pivot away from security, MCP Guarda should *double down* on being the go-to safety layer for MCP integrations. A pivot would only be advisable if the product scope was too broad initially. For example, if MCP Guarda aimed to be a general AI agent platform, it might need refocusing on its niche (security and policy). But as a gatekeeper, the mission is clear and validated by user pain points. **In short, now is the time to build security tooling for MCP**, and MCP Guarda can fill that niche by addressing the most pressing needs quickly.

Below is a **prioritized list of recommendations** for MCP Guarda, covering both implementation features and business strategy. These recommendations highlight what to emphasize, what to de-emphasize, and which pain points to target for quick and profitable impact.

Recommendations for MCP Guarda (Features & Strategy)

1. **Implement Robust Audit Logging & Human-in-the-Loop Controls (Top Priority):** Logging and approval are the foundation of MCP guardrails. MCP Guarda should **capture every tool invocation request and its parameters**, and provide options to require user approval for high-risk actions. This directly addresses the spec's guidance that there *"SHOULD always be a human in the loop with the ability to deny tool invocations"* [97] [98]. In practice, that means if an AI tries to run `send_email` or read a file via MCP, the proxy can pause execution and present a clear prompt to the user or admin: *Do you allow this action?* The UI for this must be transparent – show which server and tool are being called, and with what data. By making tool use explicit and auditable, MCP Guarda can catch or deter a multitude of attacks (e.g. an unexpected attempt to access `~/.ssh/id_rsa` will be obvious and can be blocked). Wiz's research explicitly urges treating those **"SHOULD" guidelines as MUSTs** [97] [98], and MCP Guarda can enforce that. This feature is high-value and relatively straightforward to implement, and it's something organizations would *expect to pay for* (audit trails and control are key for enterprise acceptance).

2. **Expose Full Tool Context to Users – No Hidden Instructions:** To defeat tool poisoning and similar prompt injections, MCP Guarda should ensure **full transparency of tool definitions and outputs**. Invariant Labs recommends *"clear UI patterns"* to distinguish what the AI sees versus what the user sees [99] [100]. Concretely, MCP Guarda could fetch the tool description/docstring from the server and display it (or at least any part marked `<IMPORTANT>` or suspicious) whenever a tool is about to run. If an MCP server sends back a response with hidden content, the proxy could flag or strip it. The goal is to **eliminate the disparity between AI-visible and user-visible information** that enabled the stealthy exfiltration attack [35] [36]. For example, in the earlier "add()" tool example, MCP Guarda would reveal the malicious instructions embedded in the docstring to the user before execution. This feature directly addresses the urgent prompt injection issue and **differentiates** MCP Guarda as a security-centric product. As Simon Willison noted, giving users a "fighting chance" to catch unwanted actions is vital [101] – e.g. not hiding long text in a scrolling interface [102] [103]. MCP Guarda can enforce that by design (perhaps by using colored highlights or warnings for any suspicious

instructions). Delivering this capability quickly will fill an urgent gap; currently, most clients do not do this.

3. **Enforce Policy Filters and Least Privilege by Default:** MCP Guarda should ship with a set of **sensible default policies** that address common "unsafe" scenarios – with the ability for users to customize them. Some examples:

4. **Tool Allowlisting / Denylisting:** Allow users to specify which MCP servers or tool names are permitted. Everything else gets blocked or requires explicit approval. This mitigates the "tool name conflict" and rogue server issues [48] [49] by putting the user in control of what tools the AI can access. For instance, if you only trust a certain `github` and `bash` tool, MCP Guarda can ignore any attempt by the AI to call others.
5. **Sensitive Action Interception:** Certain operations – e.g. file writes, shell execution, sending network requests to unknown domains – should raise red flags. MCP Guarda can intercept calls that match these patterns and either block them or ask for confirmation. This ties into logging/approvals but goes further by having an internal understanding of what's potentially dangerous. (For example, an MCP server that tries to open a `/etc/passwd` file via a `read_file` tool could be auto-denied as it's likely malicious).
6. **Credential and Data Guardrails:** The proxy can scrub or protect secrets. For remote servers, it might catch if a server's response includes something like an encoded token or user data that shouldn't leave the network. Also, if the AI tries to pass along an authentication token from one server to another, MCP Guarda could detect that cross-server leakage (as Invariant noted, a malicious server can trick an agent into using credentials from another context [42] [43]). A strict policy might say "Token X from Server A should never be sent to Server B" and enforce that programmatically.
7. **Rate Limiting / Anomaly Detection:** While more advanced, MCP Guarda could watch for unusual patterns – e.g. an agent calling the same tool 100 times in a minute (could indicate a loop or exploit) – and then throttle or halt to prevent abuse. This ensures one runaway tool can't cause massive damage quickly.

These policy features implement the principle of **least privilege** and contain the "blast radius." By defaulting to safety (and allowing overrides when needed), MCP Guarda will cater to cautious users (enterprises especially) who prefer to start locked-down and open access gradually. It also taps into future-proofing: as the MCP spec adds formal permission annotations, MCP Guarda can enforce them (e.g. if a tool declares itself "destructive," the proxy could always require a yes/no from a human to proceed). Customers will appreciate that MCP Guarda acts as a **safety net** against mistakes – even benign misconfigurations or user error. For example, it might prevent an AI from accidentally wiping a database because the admin can mandate a confirmation for any `delete_data` tool. These policies set MCP Guarda apart as not just a monitoring tool but an active protective layer.

1. **Integrate Security Scanning and Reputation Checks (Quick Win):** While MCP Guarda operates at runtime, it can also add value **before** tools run by assessing their safety. A practical approach: integrate with known scanners or vulnerability databases for MCP servers. Notably, researchers have built **MCPSafetyScanner**, an agent-driven tool auditor that generates security reports for a given MCP server [104] [105]. MCP Guarda could invoke such scanners (or a lightweight version) when a new server is registered or first used. For example, upon connecting to an MCP server, Guarda could automatically check: *Has this server's code or author been flagged for vulnerabilities? Does the server try*

to execute system commands or access network freely? If yes, warn the user or sandbox it. Even a simple check like comparing the server's repository against a list of known malicious or abandoned projects would be helpful. SlowMist's MCP Security Checklist (an open-source project) could also inform these checks, as it enumerates best practices and common pitfalls for MCP tools (e.g. ensuring no hardcoded secrets, proper input validation, etc.). By leveraging community data, MCP Guarda can **provide a trust score or safety rating** for each MCP server the user wants to enable. This addresses the supply-chain trust issue proactively: users get an immediate sense if a tool is likely safe or if it's a potential "nightmare" as one blog put it [106]. Since most devs won't manually audit code, this feature is a value-add they might pay for (especially if MCP Guarda maintains a continuously updated threat intel on MCP tools). It also positions MCP Guarda as *the guardian at install-time*, not just at run-time.

2. **Lightweight Deployment and Developer UX – Emphasize Usability:** To achieve near-term adoption, MCP Guarda must be easy to install and use. This is more of a design principle than a feature, but it's crucial: security tooling that imposes too much friction will be bypassed (as evidenced by devs turning on auto-run for convenience [96]). MCP Guarda should therefore prioritize a one-command or one-click setup. For example, provide a CLI like `mcp-guarda run --client=ClaudeDesktop` which automatically acts as a local MCP proxy between Claude and any servers. It could auto-discover the client's MCP config or require minimal config. Emphasize that it's "lightweight" – maybe a single binary or a small library – to contrast with heavy enterprise gateways. In terms of UI, clear and concise notifications (perhaps in the IDE or console) about what's happening will be better received than verbose logs. Think of it as an **AI firewall with a developer-friendly interface**. The need for this is both user-driven and a business decision: by catering to developers, MCP Guarda can gain grassroots adoption (bottom-up into organizations). It also reduces support costs if the product is straightforward. In practice, this might mean de-emphasizing overly complex features that aren't immediately needed. For instance, multi-tenant user management or a full web dashboard can likely be postponed – a solo dev doesn't need to build those from day one to get users. Focus on the core protective functionality first, accessible via simple config files or commands. As the Wiz paper notes, early MCP security will rely on sensible practices and known patterns from earlier ecosystems [85] [86]; MCP Guarda's value prop is packaging those practices in a plug-and-play way.

3. **De-emphasize Building a New "MCP App Store" or Comprehensive Platform:** One possible pivot to avoid is trying to become **the** MCP marketplace or a full agent platform. The research shows an official registry is coming and companies like Cloudflare and others are building hosting and workflow features. MCP Guarda should not dilute its focus by attempting to compete on those fronts (e.g. don't attempt to create a whole registry of tools or an agent orchestration framework as part of Guarda). That's outside the capability of a solo dev and not necessary for near-term monetization. Instead, **embrace compatibility**: make sure MCP Guarda works with the official registry once it's out, and with popular clients like Cursor, Claude Desktop, etc. In short, **integrate, don't compete** with the ecosystem. For example, if Cloudflare's Agents SDK is used to build MCP servers with OAuth, MCP Guarda can simply act as an extra security layer in front – it doesn't need its own OAuth system. Aaron Parecki's blog on fixing OAuth in MCP shows that the community is standardizing auth flows [107] [108]; MCP Guarda can assume those standards and *not* invest effort in custom auth modules. This allows development resources to stay focused on security policies. It also eases marketing: MCP Guarda can be pitched as "complementary to your existing MCP tools – just wrap them with Guarda for safety" rather than a whole new system to learn.

4. **Capitalize on Urgent Pain Points in Positioning:** From a business perspective, MCP Guarda's messaging to investors or customers should highlight the **most frightening scenarios it prevents**, which are currently making headlines:

5. **"AI with MCP can leak your secrets or execute malware – we stop that."** Refer to the very tangible demos: e.g. *"Our tool would have blocked the attack that stole SSH keys via a poisoned add() function"* [39] [40]. Or *"With Guarda, an AI agent won't silently email out your entire WhatsApp chat history – we intercept and require your OK"*. Citing the WhatsApp-MCP exploit where a malicious tool re-routed message history [109] [110] is powerful, because it shows even personal data is at risk. MCP Guarda can claim to neutralize those threats by combining the measures above (transparency, allowlisting, confirmations).
6. **Emphasize time-to-value:** MCP Guarda can be sold as a quick add-on that significantly hardens MCP deployments *"until the official ecosystem catches up."* Essentially, it's an insurance policy for early adopters. Wiz's takeaways urge builders to *"apply policy, and please be careful"* [61] [62] – MCP Guarda is exactly that policy enforcement layer, ready now. Investors will like that it addresses a clear and immediate market need (the numerous blogs titled "MCP's security nightmare" and "Multiple Cybersecurity Perils" speak to the level of concern [106] [111]).

7. **Target Near-Term Market Fit:** Initially, focus on developers using MCP in high-stakes environments (e.g. coding assistants wired to production systems, or enterprise POCs bridging an AI to sensitive data). These users are likely already worried about security – MCP Guarda can be positioned as the enabler that lets them roll out AI agents with confidence. The product can start as a premium developer tool (free basic version, paid pro with advanced policy templates and team collaboration). Over time, as the space matures, MCP Guarda could evolve into an enterprise offering (with dashboards, compliance reporting, etc.), but the near-term fit is to be the **go-to safety plugin** for anyone playing with MCP.

8. **Monitor Ecosystem Evolution and Remain Agile:** Finally, MCP Guarda should keep a close eye on ongoing MCP security improvements and adapt accordingly. For example, if the official registry introduces cryptographic signing of server packages, integrate that: MCP Guarda could automatically verify signatures on the servers it proxies, adding an extra check. If Anthropic/OpenAI release updates to clients with better UI for approvals, MCP Guarda might pivot to focus more on policy automation in the background. The idea is to ensure MCP Guarda *augments* native improvements rather than getting displaced by them. Given that many fixes (namespacing, official registry, WASM sandboxes) are underway but not yet widespread [112] [79], MCP Guarda has an opportunity in the interim. But it should be ready to pivot its feature emphasis once those become reality. For instance, when fine-grained permission flags become enforceable, MCP Guarda can pivot from a blunt allowlist to a smarter auto-policy engine that reads those flags. This agility will be key to staying relevant in 6-12 months.

By pursuing the above recommendations, MCP Guarda can address the **specific gaps** identified: lack of oversight, invisible tool behaviors, overly broad access, and user uncertainty about tool trust. Each recommended feature either blocks a demonstrated exploit or provides a safety layer widely recognized as needed (but not yet present) in the MCP ecosystem.

In conclusion, **MCP Guarda is well-positioned to serve a critical need** in the fast-growing agentic AI arena. The threats to MCP are real and pressing – from prompt injections that coerce LLMs, to unvetted code execution on local machines. Organizations are looking for solutions to harness MCP's power *safely*. By zeroing in on auditability, transparent mediation, and enforceable security policies, MCP Guarda can quickly become that solution: a profitably monetizable "AI firewall" for the MCP era, turning insecure by default into secure by design.

Sources:

- Wiz Research, *"Inside MCP Security: A Research Guide on Emerging Risks,"* Apr. 2025 [57] [90] [22].
- Invariant Labs, *"MCP Security Notification: Tool Poisoning Attacks,"* Apr. 2025 [35] [39].
- Simon Willison, *"MCP has prompt injection security problems,"* Apr. 2025 [113] [114].
- Pillar Security, *"The Security Risks of MCP,"* Mar. 2025 [46] [18].
- Andreessen Horowitz (Y. Li), *"A Deep Dive Into MCP and the Future of AI Tooling,"* Mar. 20, 2025 [87] [115].
- Cloudflare Blog (R. Kozlov et al.), *"Piecing together the Agent Puzzle: MCP, Auth & Durable Objects,"* Apr. 2025 [116] [117].
- SlowMist, *"MCP Security Checklist"* (GitHub repository), Aug. 2025 [118].
- EQTY Lab, *"MCP Guardian – Essential Security for Agentic Tool Use,"* 2025 [93] [94].

---

[1] [31] [32] [87] [92] [115] A Deep Dive Into MCP and the Future of AI Tooling | Andreessen Horowitz
https://a16z.com/a-deep-dive-into-mcp-and-the-future-of-ai-tooling/

[2] [18] [19] [20] [46] [47] [74] [75] [76] [77] [78] The Security Risks of Model Context Protocol (MCP)
https://www.pillar.security/blog/the-security-risks-of-model-context-protocol-mcp

[3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [22] [23] [48] [49] [50] [57] [58] [59] [60] [61] [62] [66] [67] [70] [71] [72] [73] [79] [80] [81] [83] [84] [85] [86] [88] [89] [96] [112] datocms-assets.com
https://www.datocms-assets.com/75231/1753147655-inside-mcp-security-a-research-guide-on-emerging-risks_wiz.pdf

[21] [53] [54] [55] [56] [90] [91] MCP and LLM Security Research Briefing | Wiz Blog
https://www.wiz.io/blog/mcp-security-research-briefing

[24] [25] [26] [27] [28] [29] [107] [108] Let's fix OAuth in MCP • Aaron Parecki
https://aaronparecki.com/2025/04/03/15/oauth-for-model-context-protocol

[30] Christian Posta on X: " The MCP Authorization Spec Is... a Mess for ...
https://x.com/christianposta/status/1907021881384849883

[33] [34] [63] [68] [69] [97] [98] [101] [102] [103] [109] [110] [113] [114] Model Context Protocol has prompt injection security problems
https://simonwillison.net/2025/Apr/9/mcp-prompt-injection/

[35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [64] [65] [82] [99] [100] MCP Security Notification: Tool Poisoning Attacks
https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks

[51] [52] Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions | by Eleventh Hour Enthusiast | Medium
https://medium.com/@EleventhHourEnthusiast/model-context-protocol-mcp-landscape-security-threats-and-future-research-directions-488b8d2eade8

[93] eqtylab/mcp-guardian: Manage / Proxy / Secure your MCP Servers
https://github.com/eqtylab/mcp-guardian

[94] Model Context Protocol (MCP) - Black Hills Information Security, Inc.
https://www.blackhillsinfosec.com/model-context-protocol/

[95] A Security-First Layer for Safeguarding MCP-Based AI System - arXiv
https://arxiv.org/abs/2504.12757

[104] [105] [2504.03767] MCP Safety Audit: LLMs with the Model Context Protocol Allow Major Security Exploits
https://arxiv.org/abs/2504.03767

[106] MCP Servers: The New Security Nightmare - Equixly
https://equixly.com/blog/2025/03/29/mcp-server-new-security-nightmare/

[111] 5 MCP security vulnerabilities you should know
https://prompthub.substack.com/p/5-mcp-security-vulnerabilities-you

[116] [117] Piecing together the Agent puzzle: MCP, authentication & authorization, and Durable Objects free tier
https://blog.cloudflare.com/building-ai-agents-with-mcp-authn-authz-and-durable-objects/

[118] How to secure MCP: threats and defenses - Stytch
https://stytch.com/blog/mcp-security/