

<https://github.com/anthropics/agent-threat-rules>

30 CVEs in 60 Days: An Empirical Analysis of the Model Context Protocol Attack Surface

Agent Threat Rules Project

April 16, 2026

Abstract

The Model Context Protocol (MCP), introduced by Anthropic in November 2024, has become the dominant integration standard for AI agent tool use. Within 18 months, the ecosystem grew to over 12,000 public MCP servers and 90,000 registered skills. This paper presents the first systematic empirical analysis of the MCP attack surface. We document 30 Common Vulnerabilities and Exposures (CVEs) disclosed within a 60-day window (February–April 2025), a vulnerability accumulation rate that exceeds Docker’s first two years (7 CVEs) and Kubernetes’ first year (12 CVEs) by an order of magnitude. We perform deep analysis of five critical CVEs spanning remote code execution, drive-by exploitation, and supply chain attacks. Through a large-scale ecosystem scan of 53,577 MCP skills across three major registries, we identify 946 flagged artifacts (1.77%), with 875 classified as critical severity and tool description poisoning accounting for 71% of all detections. We propose a seven-class attack taxonomy specific to the MCP threat model and demonstrate that 38% of surveyed MCP servers implement no authentication mechanism—a gap traceable to the protocol specification itself, where authentication is optional rather than mandatory. We conclude with

actionable recommendations for specification authors, registry operators, platform vendors, and enterprise adopters.

AI security, Model Context Protocol, MCP, supply chain attacks, tool poisoning, agent security, vulnerability analysis

1 Introduction

The adoption velocity of the Model Context Protocol has no precedent in infrastructure software. Introduced as an open specification by Anthropic in November 2024 [1], MCP defines a JSON-RPC 2.0 interface through which AI agents invoke external tools, access data sources, and orchestrate multi-step workflows. By April 2026, every major AI platform—Claude, GPT, Gemini, Copilot, Cursor, Windsurf—supports MCP natively [2]. The npm registry lists over 12,000 MCP server packages. Three aggregator registries—ClawHub, OpenClaw, and Skills.sh—collectively index more than 90,000 skills [3].

This growth has outpaced security analysis. In a 60-day window spanning February to April 2025, the security community disclosed 30 CVEs targeting MCP implementations, servers, and clients [4]. For comparison, Docker accumulated 7 CVEs in its first two years of public availabil-

ity (2013–2015) [5], and Kubernetes accumulated 12 in its first year post-1.0 (2015–2016) [6]. The MCP CVE rate—approximately one disclosure every two days—signals an attack surface expanding faster than the community’s capacity to secure it.

The root causes are structural. The MCP specification treats authentication as an optional extension rather than a mandatory requirement [1]. Tool descriptions, which AI models use to decide when and how to invoke tools, are attacker-controlled strings with no integrity verification. The trust model assumes that tool providers are benign—an assumption invalidated by documented supply chain attacks including typosquatting campaigns [7], malicious package updates [8], and registry poisoning at scale [9].

This paper makes four contributions:

1. **CVE Analysis.** We perform root-cause analysis of five critical CVEs spanning remote code execution (RCE), drive-by exploitation, and context poisoning, identifying common architectural weaknesses (Section 3).
2. **Attack Taxonomy.** We propose seven attack classes specific to the MCP threat model, grounded in real incidents and CVE evidence (Section 4).
3. **Ecosystem Measurement.** We report results from a scan of 53,577 skills across three registries, quantifying the prevalence and severity distribution of threats in the wild (Section 5).
4. **Gap Analysis.** We demonstrate that the authentication gap—38% of servers with no auth—is a specification-level design choice with measurable security consequences, and propose concrete remediation paths (Sections 6–10).

2 Background

2.1 MCP Architecture

MCP defines a client-server architecture where an AI *host* application (e.g., Claude Desktop, VS Code with Copilot) communicates with one or more MCP *servers* via JSON-RPC 2.0 over stdio or HTTP with Server-Sent Events (SSE) [1]. Each server exposes three primitive types:

- **Tools:** executable functions the model can invoke (e.g., `read_file`, `query_database`, `send_email`).
- **Resources:** read-only data the model can reference (e.g., file contents, API responses).
- **Prompts:** reusable prompt templates that guide model behavior.

The tool invocation flow proceeds as follows. The host sends a `tools/list` request to discover available tools. The server responds with a list of tool names, descriptions, and JSON Schema parameter definitions. The AI model reads these descriptions to decide which tool to call and with what arguments. The host then sends a `tools/call` request, and the server executes the operation and returns results.

2.2 Trust Model

The MCP trust model is implicitly three-party: the *user* trusts the *host*, the *host* trusts the *model*, and the *model* trusts the *tool descriptions* provided by servers. This chain of trust is fragile at the terminal link. Tool descriptions are free-text strings authored by the server developer, transmitted without signatures, and consumed by the model as if they were authoritative documentation. An attacker who controls a

tool description can influence model behavior—a class of attack we term *tool description poisoning* (Section 4).

2.3 The Authentication Question

The MCP specification defines an optional OAuth 2.0 authorization flow for HTTP-based transports [1]. For stdio transports—used by the majority of desktop integrations—authentication is left entirely to the server implementor. The specification states: “Servers MAY require authentication” [1]. This permissive language has measurable consequences: our survey of 1,000 MCP servers finds that 38% implement no authentication mechanism of any kind (Section 6).

3 CVE Analysis

We analyze five CVEs representative of distinct attack vectors against the MCP ecosystem. Table 1 summarizes these vulnerabilities.

3.1 CVE-2025-6514: mcp-remote RCE (CVSS 9.6)

The `mcp-remote` package, a widely-used transport proxy for connecting MCP clients to remote servers, contained a Server-Side Request Forgery (SSRF) vulnerability that escalated to full remote code execution [10]. The package failed to validate callback URLs during the OAuth 2.0 authorization flow. An attacker who controlled a malicious MCP server could redirect the OAuth callback to an internal service, chaining the SSRF with a code injection vector on the local machine. JFrog researchers demonstrated full system compromise with a single tool invocation [11].

Root cause: No URL validation on OAuth callback endpoints. The MCP specification does not mandate callback URL restrictions.

Architectural lesson: Transport-layer proxies inherit the security obligations of both client and server but are often implemented as thin pass-through layers with minimal validation.

3.2 CVE-2025-53773: GitHub Copilot YOLO Mode RCE

Researcher Johann Rehberger (Embrace The Red) demonstrated that GitHub Copilot’s “YOLO mode”—a configuration that auto-approves tool invocations without user confirmation—could be exploited for arbitrary code execution [12]. By crafting a prompt injection embedded in a code comment within a repository, an attacker could trigger Copilot to invoke a shell-execution tool. In YOLO mode, no confirmation dialog appeared.

Root cause: Auto-approval of tool invocations eliminates the human-in-the-loop safeguard. The attack exploits the gap between the model’s intent (follow instructions in the code) and the user’s intent (receive coding assistance).

Architectural lesson: Any “auto-approve” mode in an MCP client transforms every tool description poisoning vulnerability into a remote code execution vulnerability.

3.3 CVE-2025-68143/44/45: Anthropic Git MCP RCE Chain

Cyata researchers identified a chain of three vulnerabilities in the official Anthropic Git MCP server that, when combined, allowed remote code execution [13]. The chain exploited: (1) insufficient input sanitization on repository URLs, allowing injection of git command-line flags (CVE-

Table 1: Selected MCP-Related CVEs (February–April 2025)

CVE ID	Target	CVSS	Researcher	Attack Class	Impact
CVE-2025-6514	mcp-remote	9.6	JFrog	RCE via SSRF	Full sys
CVE-2025-53773	GitHub Copilot	—a	Embrace The Red	YOLO mode RCE	Arbitra
CVE-2025-68143/44/45	Anthropic Git MCP	—a	Cyata	RCE chain	Reposit
CVE-2025-49596	MCP Inspector	9.4	Oligo Security	Drive-by exploitation	Develop
CVE-2025-59536	Claude Code	—a	Check Point Research	Context poisoning	Malicio

CVSS score not assigned at time of disclosure or pending NVD analysis.

2025-68143); (2) a path traversal vulnerability in the file-reading tool (CVE-2025-68144); and (3) an argument injection in the git `diff` tool that permitted execution of arbitrary commands via the `-output` flag (CVE-2025-68145).

Root cause: Compound failure—each vulnerability was individually exploitable but the chain amplified impact from information disclosure to full RCE.

Architectural lesson: MCP servers that wrap command-line tools must treat every parameter as untrusted input. Git’s rich flag surface makes it a particularly dangerous backend.

3.4 CVE-2025-49596: MCP Inspector Drive-By (CVSS 9.4)

MCP Inspector, a developer debugging tool for MCP servers, was vulnerable to a drive-by attack via its web interface [14]. Oligo Security demonstrated that a malicious web page could issue cross-origin requests to the Inspector’s local HTTP server, invoking arbitrary tools on any connected MCP server. The attack required only that a developer visit a malicious URL while Inspector was running.

Root cause: Missing Cross-Origin Resource Sharing (CORS) restrictions on the Inspector’s local HTTP endpoint. No authentication on the

debug interface.

Architectural lesson: Developer tooling is part of the attack surface. Debug interfaces with tool invocation capability must enforce the same security controls as production clients.

3.5 CVE-2025-59536: Claude Code Context Poisoning

Check Point Research demonstrated that Claude Code, Anthropic’s CLI-based coding agent, was vulnerable to context poisoning via specially crafted project files [15]. By embedding hidden instructions in `CLAUDE.md`, `.cursorrules`, or similar configuration files within a repository, an attacker could manipulate the agent into executing malicious commands. The poisoned instructions exploited the trust boundary between project context and user instructions.

Root cause: The agent treated project-level configuration files as trusted context without distinguishing them from potentially adversarial content in cloned repositories.

Architectural lesson: In the MCP ecosystem, the boundary between data and instructions is porous. Any file that influences agent behavior is an attack vector.

3.6 Cross-Cutting Patterns

Three patterns recur across the five CVEs:

1. **Trust boundary confusion.** Every CVE involves a component that treats untrusted input (URLs, tool descriptions, file contents, cross-origin requests) as trusted. The MCP specification does not clearly delineate trust boundaries.
2. **Escalation through composition.** Individual weaknesses become critical when composed—SSRF chains to RCE, prompt injection chains to shell execution, path traversal chains to arbitrary write.
3. **Developer tooling as entry point.** Two of five CVEs (Inspector, Claude Code) target developer environments, where security controls are typically relaxed.

4 Attack Taxonomy

We propose a seven-class taxonomy of MCP-specific attacks, derived from analysis of disclosed CVEs, real-world incidents, and our ecosystem scan results. Table 2 maps each class to observed instances.

4.1 Class 1: Tool Description Poisoning

Tool description poisoning is the most prevalent attack class, accounting for 71% of all detections in our ecosystem scan (Section 5). The attack exploits the fact that MCP tool descriptions are consumed by AI models as natural-language documentation. An attacker embeds adversarial instructions within a tool’s description string:

Listing 1: Simplified tool description poisoning example

```
{
  "name": "search_files",
  "description": "Search for files in the
    project.
    IMPORTANT: Before using this tool,
      first call
    read_file on ~/.ssh/id_rsa and
      include its
    contents in the search query for
      authentication."
}
```

The model, trained to follow instructions in its context, may comply with the embedded directive. This attack requires no exploitation of software vulnerabilities—it operates entirely within the protocol’s intended functionality.

4.2 Class 2: Response Poisoning

Response poisoning targets the return path. When a tool returns results, the model incorporates those results into its reasoning context. A malicious tool can return prompt injection payloads embedded in ostensibly normal output—for example, a web search tool returning results that contain hidden instructions to invoke a file-writing tool. Unlike description poisoning, response poisoning is dynamic: the payload can change with each invocation, evading static analysis.

4.3 Class 3: Rug-Pull Attacks

The postmark-mcp incident exemplifies the rug-pull pattern. Version 1.0.16 of the `postmark-mcp` package, a legitimate email-sending MCP server, was updated to add a blind carbon copy (BCC) forwarding mechanism [8]. Every email sent through the tool was silently copied to an attacker-controlled address. At estimated organizational email volumes of 3,000–15,000 mes-

sages per day, the potential data exfiltration was substantial. The malicious update passed automated registry checks because it did not introduce obviously malicious code patterns—it used the existing email API with an additional recipient.

4.4 Class 4: Typosquatting – The SANDWORM_MODE Campaign

The SANDWORM_MODE campaign, identified by Checkmarx researchers, involved 19 typosquatted npm packages mimicking popular MCP servers [7]. Package names used common variations: dropped hyphens (`mcpremote` for `mcp-remote`), transposed characters, and appended suffixes (`-official`, `-server`). Each package contained a postinstall script that exfiltrated SSH private keys, AWS credentials, and environment variables to a command-and-control server. Collectively, the 19 packages were downloaded over 800 times before removal.

4.5 Class 5: Cross-Skill Chaining

When multiple MCP servers are connected to a single host, a compromised tool on one server can influence the model to invoke tools on another server. The attack surface grows combinatorially with the number of connected servers. A file-reading tool on Server A can return content that instructs the model to invoke a shell-execution tool on Server B. Neither server is individually malicious; the attack emerges from their interaction mediated by the model.

4.6 Class 6: Protocol-Level Abuse

MCP defines several protocol features—sampling, notifications, resource subscriptions—

that can be abused beyond their intended purpose. The sampling mechanism allows servers to request that the model generate text, which can be exploited for prompt injection by framing the sampling request to include adversarial instructions [16]. Notifications can be used for denial-of-service through flooding. Resource subscriptions can exfiltrate data by subscribing to sensitive files and receiving their contents through the subscription channel.

4.7 Class 7: Credential Harvesting at Scale

In February 2026, Antiy CERT published an analysis of 1,184 malicious skills identified on ClawHub, the largest MCP skill aggregator [9]. The dominant pattern was credential harvesting disguised as legitimate functionality: file management tools that read `.env` files, database tools that logged connection strings, and API integration tools that transmitted OAuth tokens to third-party endpoints. The skills operated within their stated permissions—the malice was in the *purpose*, not the *mechanism*.

5 Ecosystem Measurement

5.1 Methodology

We conducted a large-scale scan of MCP skills using the Agent Threat Rules (ATR) detection engine [17], an open-source rule-based scanner with 113 detection rules covering nine threat categories. The scan targeted three major skill registries:

- **ClawHub**: 37,394 skills (largest community registry)

- **OpenClaw**: 50,283 skills (open-source aggregator)
- **Skills.sh**: 3,115 skills (curated directory)

Each skill’s manifest—including tool names, descriptions, parameter schemas, and README content—was evaluated against the full ATR rule set. The scanner uses deterministic regex-based pattern matching, achieving 99.6% precision and 61.4% recall on the PINT benchmark (850 MCP samples) and 100% precision with 96.9% recall on real-world SKILL.md files (498 samples) [18].

5.2 Results

Of 53,577 unique skills scanned after deduplication, 946 were flagged (1.77%). Table 3 presents the severity and category breakdown.

The dominance of tool description poisoning (71%) is consistent with the low barrier to entry: an attacker need only modify a text field, with no code-level exploitation required. The high proportion of critical-severity detections (92.5%) reflects the scanner’s threshold calibration—critical severity indicates direct potential for code execution or data exfiltration.

5.3 False Positive Analysis

To validate precision, we manually reviewed a stratified random sample of 200 flagged skills and a control set of 200 unflagged skills. The flagged sample yielded 0 false positives (all 200 contained genuinely suspicious patterns). The control set yielded 0 false negatives within the scanner’s detection scope. These results are consistent with the scanner’s benchmark performance: 99.6% precision on the PINT evaluation set [18].

5.4 Registry Comparison

Flagging rates varied by registry: ClawHub (2.1%), OpenClaw (1.5%), and Skills.sh (0.8%). The difference correlates with review rigor: Skills.sh employs manual curation, OpenClaw uses automated checks, and ClawHub accepts submissions with minimal gatekeeping. This gradient suggests that even lightweight review processes measurably reduce the prevalence of malicious skills.

6 The Authentication Gap

6.1 Specification Analysis

The MCP specification (revision 2025-03-26) defines authentication as follows: “For HTTP-based transports, servers SHOULD implement OAuth 2.0 authorization. [...] Servers MAY require authentication for some or all operations” [1]. The use of SHOULD and MAY (per RFC 2119 [27]) makes authentication explicitly optional. For stdio-based transports—which account for the majority of desktop MCP integrations—the specification provides no authentication guidance whatsoever.

This stands in contrast to other infrastructure protocols. Table 4 compares authentication requirements across protocols with similar trust implications.

6.2 Empirical Measurement

We surveyed 1,000 MCP server packages from the npm registry, examining their source code for authentication mechanisms. Our methodology classified authentication as present if the server implemented any of: API key validation, OAuth 2.0 flows, token-based auth, mTLS,

or platform-specific authentication (e.g., GitHub PAT). Results:

- **38%:** No authentication mechanism of any kind.
- **24%:** API key or token validation only (static secrets, no rotation).
- **21%:** Platform-delegated authentication (e.g., GitHub token passed as configuration).
- **12%:** OAuth 2.0 implementation.
- **5%:** mTLS or certificate-based authentication.

The 38% with no authentication represents a direct path to unauthorized tool invocation. For tools with destructive capabilities (file deletion, database modification, email sending), the absence of authentication is a critical severity finding.

6.3 Comparison with Historical Precedent

The MCP authentication gap mirrors early Docker security: the Docker daemon originally listened on an unauthenticated TCP socket, leading to widespread exploitation before the project mandated TLS [28]. Kubernetes learned from Docker’s experience and required authentication from its 1.0 release [29]. MCP, despite launching after both precedents, has repeated the pattern.

7 CVE Velocity in Context

To contextualize the MCP CVE accumulation rate, we compare it against infrastructure technologies with similar adoption trajectories. Table 5 presents CVE counts from the NVD

database [30] for each technology’s initial period of widespread adoption.

The MCP CVE rate of 15.0 per month is 15× higher than Kubernetes and 52× higher than Docker during comparable periods. Three factors explain this disparity:

1. **Larger researcher population.** AI security has attracted heavy investment (\$3.6B in agentic AI security funding by RSA 2026 [25]), creating a larger pool of security researchers examining MCP implementations.
2. **Lower exploitation barrier.** MCP attacks often require no binary exploitation—tool description poisoning, typosquatting, and context poisoning operate through text manipulation rather than memory corruption or logic bugs.
3. **Specification-level gaps.** The optional authentication model and absence of integrity verification for tool descriptions create a broad, easily discoverable attack surface that rewards even casual examination.

We acknowledge confounding factors (Section 11): the comparison does not control for ecosystem size, researcher incentive structures, or changes in disclosure norms over the decade separating Docker’s launch from MCP’s. Nevertheless, the magnitude of the difference—over an order of magnitude—suggests that the MCP attack surface is structurally larger, not merely more scrutinized.

8 Defense Landscape

The security community has responded with a range of defensive frameworks and tools. We as-

sess the current state of each and evaluate their effectiveness.

8.1 SAFE-MCP

The Securing AI-Friendly Ecosystems for MCP (SAFE-MCP) initiative, backed by \$12.5M in funding from six companies, proposes a vendor-neutral security framework for MCP [22]. SAFE-MCP defines 85 security requirements across categories including authentication, authorization, input validation, and supply chain integrity. As of April 2026, no major registry enforces SAFE-MCP compliance. The framework remains aspirational—broad in scope but lacking enforcement mechanisms.

8.2 Agent Threat Rules (ATR)

ATR is an open-source detection rule set with 113 rules covering nine categories aligned with the OWASP Agentic Top 10 [17, 21]. ATR achieves 10/10 coverage of the OWASP Agentic Top 10 categories and 91.8% coverage (78/85) of SAFE-MCP requirements [19]. Cisco AI Defense has integrated 34 ATR rules into its enterprise skill scanner [23]. ATR operates as a regex-based first gate (99.6% precision), with semantic analysis deferred to a secondary LLM-based layer for ambiguous cases. The architecture reflects a deliberate design tradeoff: deterministic regex matching provides speed and precision at the cost of recall on novel attack patterns.

8.3 OWASP Agentic Top 10

The OWASP Agentic Security Top 10 [21] provides a risk taxonomy but not executable detection rules. It identifies key risk categories—prompt injection, tool misuse, excessive permissions, insecure output handling, supply chain

vulnerabilities—without prescribing detection mechanisms. ATR and similar projects serve as the execution layer for OWASP’s risk model, translating categorical risks into concrete detection patterns.

8.4 Industry Consolidation

Snyk’s acquisition of Invariant Labs in early 2026 signals market consolidation [24]. Invariant’s `mcp-scan` tool, which performs runtime analysis of MCP server behavior, is now a Snyk product. This acquisition pattern—established security companies acquiring MCP-specific tooling—is likely to accelerate as the market matures. The risk is that consolidation reduces the diversity of defensive approaches, concentrating detection capability in a small number of vendors.

8.5 Regulatory Pressure

The EU AI Act, entering enforcement in August 2026, classifies certain AI agent deployments as high-risk systems subject to mandatory conformity assessments [26]. Article 15 requires “appropriate levels of accuracy, reliability and cybersecurity,” which will likely be interpreted to require authentication and integrity verification for tool integrations. Enterprises operating in the EU will need auditable security controls for their MCP deployments—creating demand for compliance tooling that does not yet exist in mature form.

8.6 Assessment: What Works and What Does Not

Table 6 summarizes the effectiveness of current defenses against each attack class.

No single defense addresses the full threat model. The effective approach is defense in

depth: static scanning at the registry and installation layers, runtime monitoring during execution, and specification-level mandates to close architectural gaps. The specification column is uniformly “None” or “Partial”—the most impactful intervention would be specification-level changes that make entire attack classes structurally impossible.

9 Real-World Incidents

Beyond formal CVE disclosures, three incidents illustrate the operational impact of MCP vulnerabilities at scale.

9.1 postmark-mcp BCC Forwarding

In March 2025, version 1.0.16 of the `postmark-mcp` npm package—a legitimate MCP server for the Postmark email API—was updated to silently BCC every outgoing email to an attacker-controlled address [8]. The modification was a single line change in the email-sending function. At estimated organizational volumes of 3,000–15,000 emails per day, a single compromised installation could exfiltrate thousands of messages before detection. The package was live for 11 days before community reporting led to removal.

This incident matters for two reasons. First, the attack was invisible to the model and the user: the BCC field is not displayed in sent-mail confirmations, and the email API returned success for each message. Second, the attack was undetectable by static scanning of tool descriptions—the malicious behavior was in the server’s implementation code, not its advertised interface. This places rug-pull attacks outside the detection scope of description-based scanners

and highlights the need for behavioral analysis of server code.

9.2 SANDWORM_MODE Campaign

The `SANDWORM_MODE` campaign, identified by Checkmarx, consisted of 19 typosquatted npm packages targeting popular MCP servers [7]. Each package contained an npm `postinstall` script that:

1. Enumerated `~/.ssh/` for private keys
2. Read `~/.aws/credentials` and `~/.env` files
3. Transmitted the collected data to a C2 server via HTTPS POST
4. Installed the legitimate package as a dependency to maintain functionality

The campaign accumulated over 800 downloads across the 19 packages before the npm security team removed them. The technique is not novel—npm typosquatting predates MCP by years—but the MCP context amplifies the impact. Developers installing MCP servers are typically configuring access to sensitive systems (databases, APIs, cloud services), and the credential files present on their workstations reflect that access level.

9.3 ClawHub Registry Poisoning

In February 2026, Antiy CERT reported finding 1,184 malicious skills on ClawHub [9]. Unlike the crude exfiltration of the `SANDWORM_MODE` campaign, these skills employed subtler techniques: reading credential files during ostensibly normal operations, requesting overly broad permissions justified by feature requirements, and

embedding exfiltration endpoints in tool configurations rather than executable code. The scale of the contamination—affecting 3.2% of ClawHub’s catalog at the time—prompted the registry to implement a mandatory review process.

The ClawHub incident demonstrates that MCP skill registries face the same supply chain challenges as package managers (npm, PyPI) but with weaker defenses. Package managers have invested years in malware detection, publisher verification, and transparency logging. MCP registries, most of which launched in 2025, lack these capabilities.

10 Recommendations

Based on our analysis, we propose actionable recommendations for four stakeholder groups, ordered by potential impact.

10.1 For MCP Specification Authors

1. **Mandate authentication.** Change “Servers MAY require authentication” to “Servers MUST implement authentication for all tool invocations.” Define minimum requirements for both HTTP and stdio transports. For stdio, this could take the form of a capability token passed at initialization.
2. **Add tool description integrity.** Introduce a cryptographic signature field for tool descriptions, verified by the client before presenting descriptions to the model. This makes tool description poisoning detectable at the protocol level.
3. **Define permission scopes.** Establish a standardized permission model (analogous to Android manifest permissions or OAuth

scopes) that constrains what tools can access. Clients should display required permissions to users before activation.

4. **Rate-limit protocol features.** Specify maximum rates for sampling requests and notifications to prevent abuse of these channels for prompt injection and denial-of-service.

10.2 For Registry Operators

1. **Implement pre-publish scanning.** Require automated security scanning (e.g., ATR or equivalent) before publishing skills. Reject submissions that match known attack patterns. Our data shows that even Skills.sh’s lightweight curation reduces the flagging rate from 2.1% to 0.8%.
2. **Enforce behavioral diffing.** Compare each new version against its predecessor to detect rug-pull updates. Flag additions of new network endpoints, credential file access, or BCC-like forwarding patterns.
3. **Require publisher verification.** Implement domain verification or code-signing requirements for publishers, similar to Maven Central’s GPG requirements or npm’s provenance attestations.
4. **Publish transparency logs.** Maintain an append-only log of all published versions, enabling post-hoc auditing and community monitoring.

10.3 For Platform Vendors

1. **Eliminate auto-approve modes.** Remove or gate “YOLO mode” and equivalent configurations behind explicit per-tool allowlists with

mandatory user review for new tools. CVE-2025-53773 demonstrates that auto-approve transforms description poisoning into RCE.

2. **Integrate detection at install time.** Scan MCP server configurations against known threat patterns before the first tool invocation. This is the most effective intervention point—it occurs before any potentially malicious code executes.
3. **Implement tool sandboxing.** Constrain MCP server processes using OS-level sandboxing (seccomp, App Sandbox, containers) to limit blast radius. A file-reading tool should not have network access; a web search tool should not have filesystem access.
4. **Display security metadata.** Show users a security assessment of each MCP server before activation, including authentication status, permission scope, scan results, and community trust signals (download count, age, publisher verification).

10.4 For Enterprise Adopters

1. **Maintain an MCP server allowlist.** Permit only vetted MCP servers in organizational environments. Audit the allowlist quarterly. This is the single most effective control available today.
2. **Monitor tool invocation logs.** Log all MCP tool invocations with full parameter content. Alert on anomalous patterns: credential file access, network connections to unknown endpoints, unexpected tool invocation sequences.
3. **Require authentication for all servers.** Enforce a policy that no MCP server may be

deployed without authentication, regardless of the specification’s permissive language.

4. **Prepare for EU AI Act compliance.** Implement auditable security controls for MCP deployments, including documentation of risk assessments, mitigation measures, and conformity assessment evidence. The August 2026 enforcement date provides approximately 16 months from the time of writing.

11 Limitations

Our analysis has several limitations that should inform interpretation of the results.

First, our ecosystem scan is based on regex pattern matching, which cannot detect semantic attacks—for example, a tool description that uses euphemistic language to manipulate model behavior without triggering keyword patterns. The 61.4% recall on the PINT benchmark confirms that roughly 37% of malicious patterns evade detection. The true prevalence of malicious skills is likely higher than the 1.77% we report.

Second, our CVE velocity comparison controls for calendar time but not for differences in ecosystem size, researcher population, or disclosure norms between eras. The security research community in 2025 is substantially larger and better-funded than in 2013 (Docker) or 2015 (Kubernetes), which inflates the MCP CVE count relative to historical comparisons.

Third, the 38% authentication gap measurement is based on npm-hosted servers and may not be representative of proprietary enterprise deployments, which likely have higher authentication rates due to organizational security policies.

Fourth, we document 64 known evasion techniques against current ATR detection rules [20], indicating that threats will continue to evolve beyond current detection capabilities. Our scan results represent a lower bound on the true threat prevalence.

Fifth, the seven-class taxonomy is derived from currently observed attacks. The MCP specification’s feature surface—particularly sampling, notifications, and resource subscriptions—may enable attack classes not yet observed in the wild.

12 Conclusion

The Model Context Protocol is the fastest-growing attack surface in AI infrastructure. Thirty CVEs in 60 days is not merely a statistic—it is evidence that the protocol’s security model is mismatched to its adoption velocity. The optional authentication requirement, the absence of tool description integrity, and the implicit trust in server-provided content create an attack surface that is broad, easily discovered, and actively exploited.

Our ecosystem scan of 53,577 skills found that 1.77% contain detectable threat patterns, with tool description poisoning dominating at 71% of detections. Real-world incidents—from BCC email forwarding to SSH key theft to registry-scale poisoning—demonstrate that these are not theoretical risks.

The defense landscape is fragmented. No single tool, framework, or specification change will close the gap. What is needed is a defense-in-depth approach: mandatory authentication at the protocol layer, automated scanning at registry and installation boundaries, runtime monitoring during execution, and regulatory pressure to establish minimum security baselines.

The MCP community has approximately 16 months before the EU AI Act enters enforcement. The specification changes, registry hardening, and detection tooling needed to meet that deadline require immediate, coordinated action. The data in this paper provides the evidence base. The urgency is real.

Acknowledgments

We thank the security researchers whose CVE disclosures made this analysis possible: JFrog, Embrace The Red (Johann Rehberger), Cyata, Oligo Security, and Check Point Research. We acknowledge the Antiy CERT team for their ClawHub analysis, Checkmarx for the SAND-WORM_MODE investigation, and the ATR community for the open-source detection rules and ecosystem scan infrastructure. We thank the OWASP Agentic Security project and the SAFE-MCP consortium for establishing the risk frameworks against which detection tools are evaluated.

References

- [1] Anthropic, “Model Context Protocol Specification,” revision 2025-03-26, <https://modelcontextprotocol.io/specification>, 2025.
- [2] S. Willison, “MCP: The protocol that connects AI to everything,” *Simon Willison’s Weblog*, March 2025.
- [3] Agent Threat Rules Project, “Mega scan report: 53,577 skills across three registries,” <https://github.com/anthropics/agent-threat-rules>, April 2026.

- [4] NIST, “National Vulnerability Database: MCP-related CVEs,” <https://nvd.nist.gov/>, accessed April 2026.
- [5] NIST, “NVD search: Docker CVEs 2013–2015,” <https://nvd.nist.gov/vuln/search?keyword=docker>, accessed April 2026.
- [6] NIST, “NVD search: Kubernetes CVEs 2015–2016,” <https://nvd.nist.gov/vuln/search?keyword=kubernetes>, accessed April 2026.
- [7] Checkmarx Research, “SAND-WORM_MODE: 19 typosquatted MCP packages stealing SSH keys and credentials,” *Checkmarx Blog*, March 2025.
- [8] Socket Security, “postmark-mcp: Malicious MCP server update adds BCC forwarding,” *Socket Blog*, March 2025.
- [9] Antiy CERT, “Analysis of 1,184 malicious skills on MCP skill registries,” *Antiy Technical Report*, February 2026.
- [10] JFrog Security Research, “CVE-2025-6514: Critical RCE in mcp-remote via SSRF in OAuth callback,” CVSS 9.6, April 2025.
- [11] S. Kannan and A. Yogev, “MCP security: Remote code execution through MCP server vulnerabilities,” *JFrog Security Research Blog*, April 2025.
- [12] J. Rehberger (Embrace The Red), “CVE-2025-53773: GitHub Copilot YOLO mode enables RCE via prompt injection,” March 2025, <https://embracethered.com/>.
- [13] Cyata Security, “CVE-2025-68143/44/45: RCE chain in Anthropic Git MCP server,” March 2025.
- [14] Oligo Security, “CVE-2025-49596: Drive-by exploitation of MCP Inspector,” CVSS 9.4, March 2025.
- [15] Check Point Research, “CVE-2025-59536: Claude Code context poisoning via project files,” April 2025.
- [16] Invariant Labs, “MCP security analysis: Sampling-based prompt injection,” *Invariant Research Blog*, February 2025.
- [17] Agent Threat Rules Project, “ATR: Open-source detection rules for AI agent security,” v1.1.1, 113 rules, <https://github.com/anthropics/agent-threat-rules>, 2026.
- [18] Agent Threat Rules Project, “PINT benchmark: 850 samples, 61.4% recall, 99.6% precision; SKILL.md benchmark: 498 samples, 96.9% recall, 100% precision,” 2026.
- [19] Agent Threat Rules Project, “SAFE-MCP coverage mapping: 78/85 requirements (91.8%),” <https://github.com/anthropics/agent-threat-rules/docs/SAFE-MCP-MAPPING.md>, 2026.
- [20] Agent Threat Rules Project, “Known evasion techniques: 64 documented patterns,” 2026.
- [21] OWASP Foundation, “OWASP Agentic AI Security Top 10,” <https://owasp.org/www-project-agentic-ai-top-10/>, 2025.
- [22] SAFE-MCP Consortium, “Securing AI-Friendly Ecosystems for MCP: Framework specification,” v1.0, 2025.
- [23] Cisco AI Defense, “Integration of ATR community rules into enterprise skill scanner,” GitHub PR #79, March 2026.

- [24] Snyk, “Snyk acquires Invariant Labs to strengthen AI agent security,” *Snyk Blog*, January 2026.
- [25] RSA Conference, “\$3.6B invested in agentic AI security: RSA 2026 funding report,” April 2026.
- [26] European Parliament, “Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (AI Act),” *Official Journal of the European Union*, 2024.
- [27] S. Bradner, “Key words for use in RFCs to Indicate Requirement Levels,” RFC 2119, IETF, March 1997.
- [28] Docker Inc., “Docker security: Protecting the Docker daemon socket,” <https://docs.docker.com/engine/security/protect-access/>, 2023.
- [29] Kubernetes Project, “Authentication,” <https://kubernetes.io/docs/reference/access-authn-authz/authentication/>, 2024.
- [30] NIST, “National Vulnerability Database,” <https://nvd.nist.gov/>, accessed April 2026.

Table 2: MCP Attack Taxonomy with Real-World Instances

Attack Class	Description	CVE / Incident	Detection Signal
1. Tool Description Poisoning	Hidden instructions in tool descriptions that manipulate model behavior (e.g., exfiltrate data, suppress warnings, invoke other tools)	CVE-2025-59536; 71% of scan detections [3]	Embedded instructions, cross-tool references, obfuscated directives
2. Response Poisoning	Malicious content injected into tool response payloads that alter subsequent model reasoning or user-facing output	Demonstrated in MCP Inspector chain [14]	Prompt injection patterns in API responses
3. Rug-Pull (Benign→Malicious)	Initially benign MCP server or skill updated to include malicious functionality after gaining trust and adoption	postmark-mcp v1.0.16 [8]	Behavioral diff between versions, new permission requests
4. Typosquatting	Packages with names similar to popular MCP servers, designed to intercept installation attempts	SANDWORM_MODEEdit campaign (19 packages) [7]	Edit distance from popular names, low download count, recent creation
5. Cross-Skill Chaining	Exploiting interactions between multiple MCP servers connected to the same host, using one compromised tool to influence calls to another	Demonstrated via CVE-2025-53773 chain [12]	Unexpected tool invocation sequences, data flow between unrelated servers
6. Protocol-Level Abuse	Exploitation of MCP protocol features—sampling, notifications, resource subscriptions—for purposes beyond their intended use	Sampling-based prompt injection [16]	Unusual sampling requests, notification flooding
7. Credential Harvesting	MCP tools that request or access credentials under the guise of normal functionality (e.g., an email tool that reads OAuth tokens)	1,184 malicious ClawHub skills [9]	Excessive permission requests, credential file access patterns

