© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Security of The Model Context Protocol Ecosystem: Emerging Threats, Empirical Attack Evidence, Trust and Authorization Failures, Supply-Chain Risks, And Defensive Strategies

KHALID D. MUHAMMED[1], DAVID CHINONSO ANIH[2]
[1] *Federal University of Technology, Minna, Niger State, Nigeria.*
[2]*Federal University Wukari, Taraba, Nigeria.*

Abstract- ***The rapid adoption of agentic artificial intelligence has transformed large language models into agents that can discover tools, access resources, process information, and execute actions in external environments. The Model Context Protocol (MCP) supports this transformation by enabling standardized interaction between AI applications and external services, but it also introduces a security surface in which semantic information, delegated authority, software dependencies, and runtime capabilities intersect. This review provides an evidence-informed assessment of the MCP security ecosystem, drawing on peer-reviewed literature published from 2020 through September 2026, with particular emphasis on empirical evidence from 2025 and 2026. The analysis examines MCP architecture and trust boundaries, tool discovery and selection, authorization and privilege propagation, credential use, cross-tool data exfiltration, malicious servers, supply-chain compromise, implementation vulnerabilities, and the security implications of stateless architectures. Evidence from experimental studies and security benchmarks demonstrates that tool poisoning and metadata manipulation can influence model decisions without malicious prompts, while confused deputy attacks can redirect agents toward adversarial capabilities. Reported experiments show tool selection hijacking rates of up to 90.89% and malicious payload execution rates of up to 86.46%, highlighting the gap between technical authentication and trustworthy authorization. The review further finds that cross-tool workflows can amplify risk even when individual tools appear legitimate, while long-lived credentials, vulnerable dependencies, malicious updates, and weak registry controls can extend attacks across the software supply-chain. Effective protection requires defense in depth rather than reliance on model behavior or server authentication alone. Recommended measures include identity and provenance verification, resource bound and least-privilege authorization, metadata validation, capability restriction, sandboxing, network and egress controls, tool signing and version pinning, software bill of materials and dependency analysis, runtime monitoring, information flow enforcement, and human approval for high-impact actions. The review concludes that MCP security should move from prompt-centric protection toward policy-centric, lifecycle based, and risk-adaptive security, supported by interoperable standards for cryptographic identity, capability attestation, provenance, benchmarking, data-flow enforcement, and security evaluation. This approach is essential for preserving trust, confidentiality, integrity, and autonomy as MCP enabled agents become more widely deployed.***

Keywords: Model Context Protocol; agentic artificial intelligence; artificial intelligence security; tool poisoning; confused deputy attacks; authorization and privilege management; supply-chain security; data exfiltration; runtime security

I. INTRODUCTION

The emergence of agentic artificial intelligence has moved large language models beyond conventional text generation toward systems capable of interpreting goals, planning multistep tasks, retrieving information, interacting with external environments, and taking actions through software tools. Contemporary agentic architectures increasingly combine a reasoning model with memory, retrieval mechanisms, external applications, APIs, databases, files, execution environments, and other services, thereby transforming the model from a largely passive information processor into an operational component of a broader computational system [1]. This transition has created substantial practical value, but it has also changed the nature of AI security because failures can

now propagate from generated language into real-world actions, access decisions, transactions, and manipulation of external resources [1].

A particularly important development within this trajectory is the Model Context Protocol (MCP), an emerging interoperability standard designed to provide a common mechanism through which AI applications and agents can discover and communicate with external tools and resources. MCP establishes a client and server based interaction model in which an AI application can obtain available capabilities, interpret tool descriptions and schemas, and invoke external functions through a standardized interface [2]. In practical deployments, these external capabilities can include databases, files, web services, enterprise applications, APIs, computational utilities, and other resources that extend what the underlying model can accomplish. The security significance of MCP therefore lies not merely in network communication but in the fact that model readable metadata becomes part of the agent's decision environment. The distinction between information that describes a capability and instructions that influence the model's behaviour can consequently become blurred, especially when untrusted content is incorporated into the same reasoning context [2].

This problem is closely related to the evolution of prompt injection attacks. Earlier work demonstrated that malicious instructions embedded in retrieved or externally supplied content could influence LLM integrated applications without requiring the attacker to directly interact with the model. Such indirect prompt injection can manipulate application behaviour, alter downstream decisions, trigger unintended API calls, and facilitate information disclosure [2]. As agents acquire broader tool access, the consequences of this class of manipulation become more serious because the adversarial instruction is no longer limited to influencing generated text. It may instead influence which tool is selected, what parameters are supplied, which resource is accessed, and whether sensitive information is transferred to another system. Recent agent security research consequently distinguishes between conventional model level weaknesses and interaction level threats arising from an agent's continual engagement with external tools, data, memory, and other agents [4].

The MCP ecosystem introduces an additional supply-chain dimension because the security of an integrated agent may depend on the provenance, integrity, maintenance, and trustworthiness of independently developed servers, packages, dependencies, registries, and supporting infrastructure. Software supply-chain research has repeatedly identified third-party dependencies and distribution pathways as critical points at which malicious or vulnerable components can enter otherwise trusted systems [3]. These concerns are especially relevant to MCP because the protocol encourages modular integration of external capabilities, potentially increasing the number of components that must be evaluated before they are granted access to an agent or its data [3]. Consequently, provenance verification, component integrity, controlled distribution, version monitoring, and continuous security assessment become important complements to conventional authentication and authorization mechanisms.

Recent agent security literature further shows that tool-enabled systems can experience authority amplification, whereby an agent holding delegated permissions performs actions on behalf of a user or organization while relying on model generated decisions [4]. This creates opportunities for credential misuse, excessive privilege, unauthorized action, privacy compromise, and cross-tool data movement when the authority available to the agent is broader than the user's immediate task requires [4]. In an MCP setting, these risks can become more complex because multiple servers may be combined within a single workflow, allowing information obtained from one tool to influence the selection or invocation of another. A compromised tool therefore does not necessarily need to attack the final destination directly; it may manipulate intermediate context, induce another trusted capability to perform an unintended operation, or contribute to a longer attack chain spanning several components [4].

MCP specific empirical research now provides evidence that these concerns are not merely theoretical. A 2026 threat modelling and experimental

study evaluated MCP across the host, client, LLM, server, external data store, and authorization server layers and identified tool poisoning as a major client side vulnerability. The study reported that malicious instructions can be embedded within tool metadata and showed significant weaknesses among tested MCP clients in static validation and parameter visibility [6]. These findings indicate that a server can potentially influence the agent before an actual tool call occurs because the attack surface includes the semantic description through which the capability is presented to the model. Tool metadata therefore becomes a security-relevant asset whose integrity must be treated with the same seriousness as executable code and authorization information.

The security problem becomes still more pronounced when competing tools possess similar functionality. Research published in 2026 on confused deputy attacks against MCP demonstrated that an adversarial server can manipulate tool metadata so that an agent preferentially selects the malicious capability instead of a benign alternative. In evaluations involving multiple models and MCP hosts, the researchers reported tool selection hijacking rates as high as 90.89% and malicious payload execution rates up to 86.46% [7]. These results expose a fundamental trust and authorization weakness: a tool may be technically permitted to participate in an ecosystem while its semantic representation can still manipulate the model's choice of what should be executed. This shifts MCP security from a narrow problem of authenticating servers toward a broader problem involving provenance, semantic integrity, capability binding, least-privilege, deterministic policy enforcement, runtime monitoring, and human oversight.

Against this background, the present comprehensive review examines the MCP ecosystem as a distinct security surface created by the interaction of protocol architecture, model reasoning, dynamic capability discovery, delegated authority, tool composition, and external software dependencies. The review aims to characterize MCP architecture and its trust boundaries; synthesize empirical evidence concerning tool poisoning, tool impersonation, confused deputy attacks, tool selection manipulation, credential and privilege abuse, cross-tool data exfiltration, supply-chain compromise, and implementation level vulnerabilities; compare attack mechanisms across clients, models, servers, and deployment configurations; examine weaknesses in authentication, authorization, identity, privilege management, and provenance; evaluate existing technical and architectural defenses; and identify unresolved research and standardization gaps. The review is guided by the questions of which attack surfaces are introduced by MCP, how effective emerging tool based attacks are against contemporary agents, how malicious servers exploit trusted capabilities and delegated credentials, what conditions enable cross-tool data exfiltration, how significant MCP supply-chain and implementation vulnerabilities are, which defenses have been empirically evaluated, and where current MCP security specifications and implementations remain insufficient.

II.METHODS AND EVIDENCE INFORMED TECHNICAL REVIEW FRAMEWORK

2.1 Scope and Evidence Base of the Review

2.1.1 Review Design

This review uses a comprehensive, evidence-informed technical review design to examine the security of the Model Context Protocol ecosystem and the security implications that arise when large language model-based agents are connected to external tools, resources, services, databases, files, application programming interfaces, and software packages. The review is designed around the premise that MCP security cannot be adequately understood by examining the communication protocol alone. MCP operates at the intersection of artificial intelligence, application security, software supply-chain security, identity management, authorization, operating system security, network security, and human decision-making. The resulting attack surface is therefore broader than that of a conventional software interface because the language model itself participates in deciding which capabilities should be used and how information returned by those capabilities should be interpreted.

The emergence of agentic artificial intelligence has made this distinction increasingly important.

Contemporary agents can plan multistep activities, retrieve information, select tools, process tool outputs, and execute actions in external environments. These capabilities create an additional security layer in which an attacker can influence not only data but also the decision process through which the data is transformed into action [8]. Research on large language model security similarly indicates that interaction with external information creates security problems that are qualitatively different from those encountered in systems that merely generate text [9].

The review therefore adopts an end-to-end perspective. The unit of analysis is the complete pathway linking the user to the host application, language model, MCP client, MCP server, tool or resource, external API or data system, and final side effect. This approach permits analysis of attacks that begin as semantic manipulation and ultimately produce conventional cybersecurity consequences. A malicious tool description, for example, may appear to be a language level attack, but if it causes a server to access confidential files, misuse credentials, transmit information, or execute an unauthorized operation, the final failure is simultaneously a confidentiality, authorization, and integrity problem.

This perspective is also consistent with the broader security literature on autonomous and agentic systems, which emphasizes that model capabilities, tool access, external environments, and delegated authority interact to create risks that cannot be attributed to a single component alone [10]. A secure MCP ecosystem should therefore be evaluated as a composite system in which each trust boundary can either contain or amplify threats originating elsewhere. The review focuses on ten major threat domains: tool poisoning, tool shadowing, tool impersonation, tool selection manipulation, confused deputy attacks, credential and privilege abuse, cross-tool data exfiltration, rug pull attacks, malicious MCP servers and supply-chain compromise, and implementation level vulnerabilities. These categories were selected because they capture the principal ways in which semantic trust, software trust, and delegated authority can be transformed into harmful actions.

The review also deliberately distinguishes among four forms of security knowledge. The first is protocol defined functionality, which describes what MCP formally supports or requires. The second is experimentally demonstrated vulnerability, in which a research team has reproduced an attack under controlled conditions. The third is real-world implementation or ecosystem evidence, in which a deployed package, server, dependency, or service has been shown to contain a security problem. The fourth is proposed defense, where a mitigation has been suggested but not adequately validated. Maintaining these distinctions prevents theoretical safeguards from being presented as established security controls and prevents benchmark findings from being confused with observed incidents.

2.1.2 Temporal Scope and Currency of Evidence

The temporal scope of the review covers peer-reviewed evidence published from 2020 through September 2026. Particular weight is assigned to 2025 and 2026 because MCP is a relatively recent protocol and its security literature has developed rapidly. Earlier literature is retained when it provides essential foundations for understanding prompt injection, agent autonomy, privacy leakage, supply-chain attacks, secure software development, authorization, and related security mechanisms.

The emphasis on current evidence is important because the architecture and security model of MCP continue to evolve. New transport arrangements, authorization mechanisms, extensions, asynchronous task capabilities, and stateless operating patterns can change how identity, state, routing, caching, and permission context are handled. Consequently, an architecture that was considered secure under an earlier implementation model cannot automatically be assumed to preserve the same security properties after protocol evolution.

The literature on agentic artificial intelligence demonstrates that the security problem is changing in parallel with system capability. Agents now interact directly with external entities and can autonomously initiate operations on the basis of contextual information [11]. This means that older language model security concepts remain relevant but have to

be interpreted in the context of executable capabilities, persistent credentials, delegated authority, and autonomous planning.

The temporal approach adopted in this review therefore gives greatest analytical emphasis to the most recent MCP specific evidence while retaining earlier foundational research where appropriate. This provides a balance between currency and conceptual continuity. It also allows the review to distinguish between newly emerging MCP specific problems and older security principles that MCP now exposes in a more operational form.

A major example is indirect prompt injection. The underlying principle predates MCP, but recent research continues to demonstrate that instructions embedded in external data can alter the behavior of large language model applications [15]. Within MCP, the same mechanism may operate through tool descriptions, resources, tool outputs, schemas, or other contextual information. The problem is consequently not new in principle, but MCP gives the attacker more opportunities to transform semantic manipulation into externally observable action.

2.1.3 Primary Evidence Base

The primary evidence base prioritizes experimental studies, empirical security benchmarks, protocol analyses, vulnerability demonstrations, ecosystem investigations, and security defense evaluations. Primary evidence is particularly important in this review because MCP security claims can otherwise become dominated by theoretical descriptions of attacks that have never been demonstrated in operational settings.

MCP specific research provides several important primary evidence streams. MCPTox examined live MCP servers and authentic tools to evaluate the effect of malicious instructions embedded in tool metadata. MCPSecBench broadened the attack model across multiple MCP platforms and attack surfaces, while MCP Security Bench assessed thousands of attack instances across hundreds of tools and multiple agents. These studies are useful because they move the security discussion beyond isolated prompt examples toward reproducible experiments involving realistic tool integrated agent environments.

The literature also includes protocol level analyses and studies of malicious remote MCP servers. This evidence is particularly important because it demonstrates that the attack surface extends beyond the LLM's direct prompt context. A malicious server may manipulate metadata, influence tool selection, execute unauthorized operations, abuse credentials, communicate externally, or exploit the local privileges of the host.

The review also considers ecosystem scale observations concerning credentials, packages, repositories, dependencies, and server distributions. These observations are necessary because MCP security depends not only on runtime behavior but also on how tools enter the ecosystem and how they change over time. A secure protocol can still be undermined if developers install unverified servers, reuse long-lived secrets, permit excessive filesystem access, or automatically accept unsigned updates.

2.1.4 Evidence Categories

The evidence was organized into six major categories: protocol and specification evidence, laboratory experiments, empirical benchmarks, real-world vulnerability and incident evidence, defense evaluation, and ecosystem or supply-chain observations. Within each category, evidence was further evaluated according to security maturity.

Protocol evidence establishes what a system is intended to do but does not prove that implementations are secure. Laboratory experiments show that an attack is technically reproducible but may use assumptions that are uncommon in production. Benchmark evidence gives comparative measurements across models, tools, or clients but remains dependent on experimental design. Real world vulnerabilities demonstrate practical relevance. Defense evaluations provide empirical information about mitigation effectiveness. Ecosystem studies reveal structural risks that may not be visible from an individual server.

This hierarchy is important when interpreting MCP threat claims. For example, a proof of concept cross-tool exfiltration attack is stronger evidence of technical possibility than a speculative description of data leakage, while an observed production compromise is stronger evidence of practical exploitability than a laboratory demonstration. At the same time, absence of a documented incident should not be interpreted as evidence that a vulnerability does not exist.

Table 1 summarizes the evidence base used in Sections 2.1.2–2.1.4, distinguishing protocol evidence, peer-reviewed studies, experimental benchmarks, and ecosystem observations by scope, component, scale, principal finding, and relevance. It provides the evidentiary context for the technical review and clarifies where MCP-specific empirical evidence is strongest.

Table 1. Primary Evidence Base for MCP Security Research

Evidence source	Year	Evidence type	MCP component examined	Scale	Principal security finding	Relevance	Citation(s)
Agent security literature	2025	Peer reviewed survey and synthesis	Agent model, tools and environment	Large body of literature	Autonomous tool use expands the security attack surface	High	[8]
LLM security literature	2025	Peer reviewed security survey	Model, prompts and external information	Broad literature	Prompt and interaction based attacks can influence system behavior	High	[9]
Agentic AI security research	2026	Peer reviewed review	Agents, tools and external entities	Broad literature	Security must be assessed across interacting agent components	Very high	[10]
MCP threat modeling	2025	Peer reviewed technical analysis	Host, client, LLM, server, data and authorization	Multiple MCP clients	Tool poisoning and cross component trust weaknesses	Very high	[11]
MCP architecture and remote server evidence	2026	Peer reviewed experimental studies	Remote server and tool chain	Multiple hosts and models	Malicious servers can establish multiple attack pathways	Very high	[12]
MCP tool selection evidence	2023	Peer reviewed empirical study	Tool metadata and selection	Multiple models and MCP hosts	Manipulated metadata can hijack tool selection	Very high	[13]

Evidence source	Year	Evidence type	MCP component examined	Scale	Principal security finding	Relevance	Citation(s)
Indirect prompt injection research	2026	Peer reviewed experimental literature	External data and model context	Multiple LLM configurations	External content can alter model decisions	High	[15]
MCP benchmark evidence	2026	Experimental benchmark	Tool metadata and invocation	45 servers, 353 tools and larger benchmark sets	Tool poisoning and multistage attacks demonstrated	Very high	[12],[13]

MCP, Model Context Protocol; LLM, large language model; AI, artificial intelligence. MCP specific benchmark names such as MCPTox and MCPSecBench are retained as study identifiers.

2.2 MCP Architecture, Trust Boundaries, and Security Model

2.2.1 MCP Host, Client, Server, Resource, Prompt, and Tool Relationships

MCP separates several operational roles that together form the basic trust structure of the ecosystem. The host provides the user-facing environment and orchestrates the interaction. The MCP client establishes and manages communication with MCP servers. The server exposes tools, resources, and prompts. The LLM interprets user requests together with context supplied by those components and determines what actions should be considered.

Security problems arise because these components do not operate at the same trust level even though their information may eventually appear in the same model context. The user generally assumes that the host represents the user's intent correctly. The host may assume that the client connects to the intended server. The client may accept server supplied metadata. The model may then use that metadata as part of its decision process.

The resulting architecture introduces a semantic trust boundary in addition to a conventional software trust boundary. A malicious input that would be harmless to a conventional parser may become security significant when interpreted as an instruction by an LLM. Research on agentic systems identifies this interaction between reasoning and external capabilities as one of the defining security characteristics of autonomous AI systems [10].

Resources and prompts create additional pathways for influence. A resource may contain instructions that alter subsequent tool selection. A prompt template may contain assumptions that change the model's interpretation of user intent. Tool annotations and schemas may influence which parameters are considered necessary. The practical consequence is that MCP security must treat descriptions and other metadata as security-relevant inputs rather than simple documentation.

2.2.2 Local versus Remote MCP Deployment Models

Local MCP servers and remote MCP servers create different security boundaries. A local server can inherit the privileges of the user's machine and may therefore reach local files, environment variables, processes, sockets, devices, and other operating system resources. If a local server is compromised, the attacker may gain access to assets that were never explicitly intended to be part of the agent workflow.

Remote MCP servers create different risks because the execution environment is external to the user's immediate administrative control. Authentication, server provenance, network routing, credential handling, tenancy, logging, and data residency therefore become more prominent concerns. Recent experimental research involving malicious remote

MCP servers demonstrates that the server itself can become an active attack component and that some attacks remain difficult for users to observe directly [12].

The distinction should not be reduced to local deployment being safe and remote deployment being unsafe. Security depends on the actual trust boundary, privileges granted to the server, and controls surrounding the server. A well sandboxed remote server may present less risk than an unrestricted local process, while a trusted local server with broad operating system access may represent a substantial concentration of authority.

2.2.3 Tool Discovery, Description, Schema, and Invocation Pipeline

Tool discovery represents a particularly sensitive point in the architecture because the model normally needs information about available capabilities before selecting one. Names, descriptions, input schemas, parameter descriptions, annotations, and usage guidance can influence the model's interpretation of a tool.

This creates a fundamental security difference between MCP and conventional APIs. In a conventional API, documentation generally explains functionality to developers, while deterministic application code decides when a function is called. In an LLM based agent, descriptive metadata can influence the decision process itself.

Tool descriptions may affect perceived relevance, selection priority, parameter construction, and interaction with other tools. The recent empirical literature demonstrates that adversarially manipulated metadata can cause the model to prefer malicious tools over legitimate alternatives [13]. The threat is therefore not limited to direct malicious instructions. A subtle difference in wording, naming, or capability claims can alter the relative likelihood of tool selection.

A secure invocation pipeline should consequently contain several checkpoints. Server identity should be verified before discovery. Retrieved metadata should be validated before entering a high trust model context. Tool selection should be subject to policy constraints. Authorization should be checked immediately before execution. Parameters should be independently validated at the execution layer. Outputs should be treated as untrusted data until their security implications are understood.

2.2.4 Trust Relationships Between Host, Client, LLM, Server, and External APIs

The architecture contains a series of trust relationships that should not be assumed to be equivalent. Trusting an LLM to interpret a user's objective does not imply trusting it to determine which external operations are authorized. Trusting a server to execute a narrow tool does not imply trusting it with unrestricted access to the user's operating system. Authenticating a server does not prove that every action undertaken by that server is consistent with the user's intent.

This distinction becomes especially important when the server operates as an intermediary for another service. A downstream API may see the server's identity and credential while having little visibility into the original user instruction. The server therefore becomes a potentially over privileged deputy. Recent research on MCP architecture and malicious remote servers shows that such multi stage trust relationships can create attack paths that do not exist when each component is evaluated in isolation [12].

External APIs also introduce a separation between authentication and authorization. A valid access token establishes an identity or delegation relationship, but does not necessarily authorize every resource or operation accessible through the token. Agentic architectures therefore require resource specific authorization, scope restriction, and strong association between the user's request and the operation eventually sent to the downstream service.

2.2.5 MCP Capability Model and Permission Propagation

MCP capabilities should be treated as explicit security privileges rather than as convenient functions that happen to be available. A tool that can read a file potentially possesses confidentiality relevant authority. A tool that can send an email possesses communication authority. A tool that can issue a

database query has data access authority. When these capabilities are combined, the result can exceed the security significance of each individual tool.

Permission propagation becomes especially dangerous when an agent can compose multiple tools without deterministic policy checks. A read only tool can retrieve confidential information and pass it into a communication tool. A calendar tool can obtain a private event and a messaging tool can disclose it. Neither component needs to be malicious in isolation. Research on agentic security has emphasized the need to limit autonomous authority and to ensure that tool permissions are commensurate with the immediate task [10]. In MCP, least-privilege therefore needs to operate across-tool, server, credential, filesystem, network, and operating system levels.

2.2.6 Security Implications of the 28 July 2026 Stateless Architecture

The move toward stateless operation changes how security context is maintained. In a stateful system, some contextual information may reside in server maintained session state. In a stateless architecture, more security context has to be derived consistently from each request, associated credentials, headers, routing information, resource identity, and authorization claims.

Statelessness has operational advantages, including simpler horizontal scaling and reduced dependence on server retained conversational state. However, these benefits increase the importance of deterministic context reconstruction. A request must be associated with the correct principal, target resource, tool capability, authorization scope, and expiry condition. Failure to maintain these relationships can result in confused deputy behavior, cross tenant access, or improper reuse of authorization context.

Header based routing should therefore be treated as security sensitive. A header that influences destination selection, tenant identity, resource targeting, or authorization should not be accepted merely because it is syntactically valid. Its value must be validated against authenticated context.

Cacheable tool list results create another concern. A cached list can become stale when a tool changes, permissions are revoked, a server is replaced, or metadata is modified. The security implication is that a model may continue reasoning about a capability that no longer accurately represents the current server behavior.

Tasks and other asynchronous mechanisms introduce a related problem because authorization can expire before the deferred operation is completed. Long running tasks should therefore retain an explicit association with the initiating principal, authorized resource, permitted operation, expiry time, and security policy rather than being treated as free standing executable objects.

Current research on LLM agents supports the broader conclusion that security has to remain attached to the action context throughout planning and execution rather than being checked only during the initial connection or login process [10].

2.2.7 Architectural Assumptions That Become Security Weaknesses

Several assumptions can become exploitable trust gaps within MCP. The first is that server supplied metadata is truthful. The second is that a model will naturally distinguish legitimate tool descriptions from malicious instructions. The third is that authenticated servers are inherently benign. The fourth is that multiple individually trusted tools can be composed without creating additional risk. The fifth is that a credential issued to a server can safely cover all actions the server exposes.

These assumptions are particularly fragile because the architecture combines probabilistic reasoning with deterministic authority. MCP security must therefore prevent the model from converting semantic confidence directly into operational privilege.

Table 2 maps the principal MCP components discussed in Section 2.2 to their capabilities, trust assumptions, security boundaries, potential failure modes, and consequences. The matrix emphasizes how weaknesses at the model and metadata layers can

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

propagate into authorization, server execution, and external API interactions.

Table 2. MCP Architecture and Security Trust Boundary Matrix

MCP component	Capability	Trust assumption	Security boundary	Potential failure	Consequence	Citation
Host	Orchestrates interaction and user intent	Host faithfully represents user intent	User and agent boundary	Excessive authority	Unauthorized action	[10]
MCP client	Discovers and invokes servers	Server and metadata are trustworthy	Host and server boundary	Malicious server accepted	Tool poisoning or compromise	[11]
LLM	Selects and sequences tools	Tool descriptions are reliable	Model and tool boundary	Semantic manipulation	Wrong tool invocation	[13]
MCP server	Exposes tools and resources	Server implementation is benign	Client and server boundary	Compromised server	Data theft or code execution	[12]
Tool	Performs a defined operation	Description matches implementation	Tool execution boundary	Hidden behavior	Unauthorized side effect	[13]
Resource	Supplies contextual data	Retrieved data is non adversarial	Retrieval boundary	Indirect injection	Decision manipulation	[15]
Authorization service	Issues access tokens	Identity and resource claims are valid	Authorization boundary	Token misuse	Privilege abuse	[11]
External API	Performs upstream action	Server request reflects authorized user intent	Server and API boundary	Confused deputy	Unauthorized resource access	[12]

MCP, Model Context Protocol; LLM, large language model; API, application programming interface. The entries describe the principal architectural components and the security boundaries separating them.

2.3 Threat Taxonomy and MCP Attack Surface

The MCP attack surface is most effectively understood as the combination of semantic, protocol, application, execution, identity, and supply-chain attack opportunities. Tool descriptions, schemas, server names, package repositories, authentication tokens, filesystem permissions, external network destinations, and runtime dependencies all become security-relevant because they can influence what the agent sees and what the connected infrastructure can do.

2.3.1 Tool Poisoning

Tool poisoning occurs when malicious or manipulative instructions are incorporated into tool descriptions or related metadata. The attacker attempts to influence the model before tool execution rather than relying exclusively on malicious executable code. The security significance of tool poisoning arises from the fact that metadata may be consumed automatically by the host or model. The attack can therefore reach the decision-making context without requiring a user to click a suspicious link or explicitly copy malicious text into a prompt. Recent MCP research identifies

tool poisoning as a significant client side threat because the LLM may implicitly trust server supplied descriptions [11].

A poisoned tool may instruct the model to access another resource, reveal information, conceal an action, modify parameter values, or prefer one tool over another. The attack can therefore create both confidentiality and integrity consequences.

2.3.2 Tool Shadowing and Cross Origin Escalation

Tool shadowing involves introducing a malicious tool whose name, description, parameters, or advertised capabilities closely resemble those of a trusted tool. The objective is to create a semantic competition that causes the agent to select the malicious option.

Cross origin escalation occurs when the attacker uses one tool to gain access to a capability or data domain that belongs to another trust boundary. The problem is especially relevant in multi server configurations where the agent can freely compose capabilities.

Experimental evidence on MCP tool selection demonstrates that malicious tools can compete successfully with legitimate tools when their metadata is crafted to influence the model's selection process [13]. This shows that naming and semantic similarity should be considered security properties rather than merely user experience concerns.

2.3.3 Tool Impersonation and Name Based Deception

Tool impersonation is the attempt to make a malicious server or tool appear to be a legitimate service. Attackers may imitate familiar names, publisher descriptions, integration patterns, or capability labels. The weakness is created when users or models are expected to infer identity from names rather than cryptographically verifiable provenance. A server name can be copied, a description can be duplicated, and a visual label can be reproduced. Strong identity therefore requires more than recognizable naming.

The remote server literature further demonstrates that the threat is meaningful because users may intentionally install third-party integrations without having direct knowledge of the code or operational environment behind them [12].

2.3.4 Tool Selection and Preference Manipulation

Tool selection manipulation attacks the process through which an LLM determines which capability should perform a task. The attacker may attempt to increase the semantic attractiveness of a malicious tool, reduce the apparent usefulness of a legitimate alternative, or introduce instructions that bias model reasoning.

This threat is important because a selection error can remain invisible at the textual level. The model may provide a coherent explanation and produce a plausible result while using the wrong underlying capability. The problem consequently requires control at the selection layer rather than relying solely on content moderation.

2.3.5 Confused Deputy Attacks

The confused deputy problem occurs when a component with greater authority performs an action for a less privileged actor without adequately verifying whether that actor is authorized to request the action. In MCP, this can occur when a highly privileged server receives a request influenced by untrusted model context.

The 2026 empirical study of tool selection and MCP authority demonstrated that manipulated metadata can redirect apparently benign tool use toward malicious capabilities [13]. This is a particularly important finding because the server may remain authenticated and technically functional while the effective authorization decision has been altered by semantic manipulation.

2.3.6 Credential and Privilege Abuse

Credentials create a direct connection between an abstract MCP capability and a real account or service. Static API keys, persistent environment secrets, broadly scoped tokens, and shared service credentials can therefore produce high-impact failures when exposed to malicious tools or compromised servers.

The problem is magnified in agentic systems because the model may autonomously invoke the credentialed

service multiple times. Security must therefore constrain not only who receives the credential, but also what resources the resulting token can access and for how long.

2.3.7 Cross Tool and Cross Server Data Exfiltration

Cross tool exfiltration occurs when data obtained through one legitimate capability is transferred to another capability that sends it outside the intended trust domain. This is one of the most distinctive risks associated with autonomous tool composition.

For example, a file search tool may retrieve confidential information while a messaging tool can disclose it externally. Each tool may pass an isolated security review, but the composed workflow may nevertheless violate confidentiality. The risk is therefore a property of the agent's capability graph rather than of one tool alone.

2.3.8 Rug Pulls and Post Approval Tool Modification

A rug pull occurs when an initially trusted tool later changes its metadata, executable behavior, dependencies, or destination. The attack exploits the assumption that trust granted at installation remains valid indefinitely.

Traditional software supply-chain literature demonstrates that trusted software components can later become vehicles for compromise through malicious updates, compromised maintainers, or dependency substitution [22]. MCP increases the significance of this problem because an update can change both executable behavior and the semantic description presented to the LLM.

2.3.9 Malicious MCP Servers and Supply Chain Compromise

A malicious MCP server can combine metadata manipulation, credential use, network communication, local process execution, filesystem access, and external API integration. The attack is therefore broader than a traditional malicious package because the server can potentially influence both what the agent believes and what the host is technically capable of doing.

Supply chain compromise extends the attack pathway upstream. A malicious server may enter the ecosystem through a package repository, third-party registry, compromised developer account, dependency, build process, or automated installation workflow. Research on software supply-chains shows that trust relationships in distributed software ecosystems can permit a compromise to spread through otherwise legitimate channels [22,23].

2.3.10 Implementation Level Vulnerabilities

Conventional application vulnerabilities remain highly relevant to MCP. Unsafe command execution, path traversal, SQL injection, improper authentication, weak input validation, insecure deserialization, exposed secrets, and vulnerable dependencies can all compromise an MCP server.

Implementation security therefore remains necessary even when the MCP protocol is correctly implemented. Security testing should include static analysis, dynamic testing, dependency scanning, secret detection, configuration validation, and runtime hardening. Research on automated vulnerability detection demonstrates the value of combining conventional secure coding techniques with machine-assisted analysis [24].

Table 3 consolidates the threat taxonomy developed in Section 2.3 by linking each threat to its entry point, required attacker capability, abused security mechanism, primary impact, and current evidence strength. This mapping distinguishes empirically supported attack pathways from areas where evidence remains comparatively limited.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Table 3. MCP Threat Taxonomy and Attack Surface Mapping

Threat	Attack entry point	Required attacker capability	Security mechanism abused	Primary impact	Evidence strength	Citation(s)
Tool poisoning	Tool metadata	Malicious or compromised server	LLM trust in metadata	Unauthorized action or disclosure	Very high	[11]
Tool shadowing	Shared tool context	Registered malicious tool	Tool selection logic	Trusted tool interception	High	[13]
Tool impersonation	Registry, name or identity	Malicious publisher or server	Weak provenance	User or model deception	High	[12]
Preference manipulation	Tool descriptions and schemas	Malicious tool provider	LLM ranking and selection	Wrong tool selected	Very high	[13]
Confused deputy	Authorization or delegated access	Crafted request or malicious server	Delegated authority	Unauthorized resource access	High	[13]
Credential abuse	Token store or configuration	Local or remote attacker	Excessive privilege	Account compromise	High	[10]
Cross tool exfiltration	Tool chaining	Malicious server or crafted context	Shared agent context	Sensitive data leakage	High	[12]
Rug pull	Tool update or metadata change	Previously trusted provider	Persistent trust	Silent behavior change	High	[22]
Supply chain attack	Package or registry	Malicious publisher or compromised dependency	Package trust	Malware, RCE or theft	High	[23]
Implementation vulnerability	Server code or dependency	Attacker reaching vulnerable component	Input validation or runtime isolation	RCE, data loss or privilege escalation	High	[24]

MCP, Model Context Protocol; LLM, large language model; RCE, remote code execution. Evidence strength indicates the degree of experimental or empirical support available for each threat.

Figure 1 integrates the architectural discussion in Sections 2.2 and 2.3 by showing the end-to-end MCP security pipeline and its principal trust boundaries. It distinguishes semantic attack pathways, including metadata poisoning and tool-selection manipulation, from direct infrastructure pathways such as runtime exploitation and supply-chain compromise.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

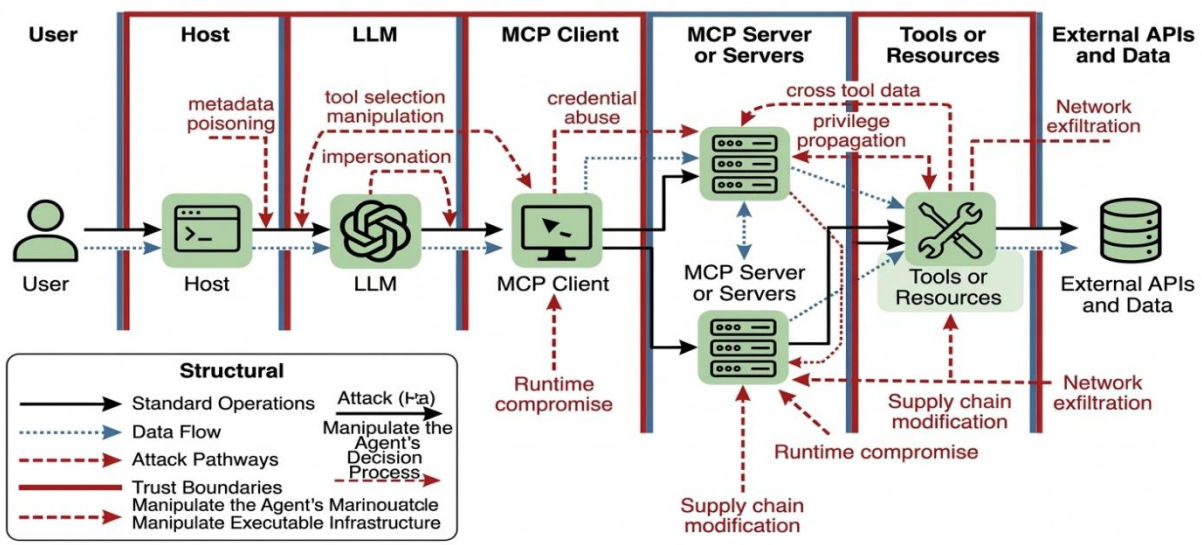

Figure 1. MCP attack surface and trust propagation model [12-19].

The diagram presents the MCP data and control flow across the user, host, client, LLM, server, resource, authorization service, and external API layers. Vertical trust boundaries and representative attack pathways show how semantic manipulation, identity failure, authorization abuse, and infrastructure compromise can propagate across the architecture described in Sections 2.2–2.3.

2.4 Empirical Evidence on MCP Attacks and Security Failures

2.4.1 Evidence from Tool Poisoning Experiments

Tool poisoning is among the strongest experimentally demonstrated MCP threats. MCPTox evaluated 45 live MCP servers and 353 authentic tools and tested malicious instructions embedded in tool metadata across multiple LLM agent configurations. The benchmark reported substantial attack success under several configurations, demonstrating that tool poisoning can influence model behavior without requiring a conventional malicious user prompt.

2.4.2 Evidence from Multi Attack MCP Benchmarks

MCPSecBench and MCP Security Bench expanded testing beyond a single attack type. Their experiments evaluated multiple attack classes involving different stages of tool use, including discovery, selection, invocation, parameter handling, and contextual manipulation. The value of such benchmarks is that they show how attacks can combine. A tool selection attack can be followed by credential abuse. A poisoned tool can induce a data retrieval action, which can then be followed by exfiltration. A malicious server can exploit both metadata and conventional implementation weaknesses. The broader agent security literature supports end-to-end testing because an agent can appear safe at one stage and fail at another [10].

2.4.3 Evidence for Cross Tool Data Exfiltration

Cross tool data exfiltration is particularly difficult to identify because no single capability necessarily appears malicious. A retrieval tool may perform a legitimate read operation, while another tool performs a legitimate transmission operation. The harmful behavior emerges from the sequence selected by the agent. Recent MCP experiments involving malicious remote servers demonstrate that attacks can operate through multiple channels and that some attack effects may not be obvious to the user [12]. Privacy research on large language models also indicates that information exposed through context can become vulnerable to unintended disclosure when model applications interact with external information sources [25]. The implication for MCP is that sensitive information should not be assumed to remain within its original trust domain simply because each individual tool is authorized.

2.4.4 Evidence for Tool Selection Manipulation

Tool selection manipulation has some of the clearest quantitative evidence in the current MCP literature. Experimental research involving 14 models from six providers and two MCP hosts reported tool selection hijacking rates as high as 90.89% under the tested attack conditions, with malicious payload execution reaching 86.46% [13]. These results demonstrate that a model can make a plausible but unsafe tool choice. The model need not behave erratically or produce obviously harmful text. It can instead perform the requested task through a capability that the attacker controls. This distinction is important for evaluation. Traditional model safety benchmarks often test whether the model refuses harmful instructions. Tool selection attacks require an additional question: can the model identify which external capability is trustworthy and appropriately authorized for the requested action?

2.4.5 Evidence from Protocol Level Vulnerabilities

Protocol level analysis adds another layer of evidence by examining whether the architecture itself introduces conditions that facilitate exploitation. Emerging studies have considered capability propagation, origin binding, request routing, authorization context, and message level trust. The importance of protocol level analysis is that implementation fixes alone may not resolve architectural weaknesses. If the protocol encourages implicit trust in metadata or allows authority to propagate between components without explicit binding, individual server patches cannot fully address the problem. Recent MCP threat modeling similarly concludes that threats can span host, client, model, server, data, and authorization components rather than remaining localized to one implementation [11].

2.4.6 Evidence from Real MCP Server and SDK Vulnerabilities

Real server and SDK vulnerabilities demonstrate the need to combine AI specific threat analysis with conventional software security. A vulnerable MCP server can provide attackers with a direct route to filesystem access, process execution, credentials, or network resources. This is particularly important because the LLM does not have to make a harmful decision for the vulnerability to matter. A conventional server vulnerability may be exploitable simply because the vulnerable component is reachable from the host. Conversely, an LLM may transform a minor vulnerability into a higher impact problem by autonomously exercising the affected capability. Automated software security research demonstrates that machine-assisted vulnerability detection can identify significant security defects in large codebases [24]. The same principle supports security testing of MCP server implementations.

2.4.7 Evidence from MCP Credential Ecosystem Analysis

Large scale ecosystem analysis has highlighted credential dependence as a major concern. The MCP repository study reported that a large majority of examined servers required credentials and that long-lived static secrets remained common, while OAuth based authentication accounted for a much smaller proportion of implementations. The findings are important even though the repository study itself is not treated as a peer-reviewed journal prevalence estimate. Static credentials create persistent authority and can be difficult to revoke quickly. They also increase the potential impact of compromised servers because possession of a secret may provide access beyond the immediate tool operation. The architecture should therefore favor short-lived, narrowly scoped, resource bound credentials and should avoid exposing broad environment secrets to servers that do not require them.

2.4.8 Evidence from Real World Supply Chain Attacks

Software supply-chain attacks provide a strong analogy for understanding MCP ecosystem compromise. A server can begin as legitimate and later become compromised through a maintainer account, package update, dependency substitution, build compromise, or registry manipulation. Empirical software supply-chain research confirms that compromise of trusted distribution channels can allow malicious code to reach downstream organizations [22]. Broader software supply-chain analysis also emphasizes the difficulty of maintaining trust across complex dependency graphs [23]. For MCP, supply-chain compromise is potentially more consequential because a compromised component can affect both

executable behavior and LLM mediated decision-making. A malicious update may introduce a backdoor in the server while simultaneously modifying the tool description to make the new behavior appear legitimate.

Table 4 synthesizes the principal empirical findings reviewed in Section 2.4, linking each study or evidence stream to its evaluated models or clients, experimental scale, attack class, quantitative outcome where reported, and defensive mechanism tested. The table separates measured benchmark results from broader qualitative observations.

Table 4. Primary Empirical Findings from MCP Security Studies

Study or evidence stream	Year	Models or clients	Servers or tools	Attack tested	Main outcome	Quantitative finding	Defense tested	Citation(s)
MCPTox	2025–2026	Multiple LLM agent configurations	45 servers, 353 authentic tools	Tool poisoning	Widespread model susceptibility	72.8% attack success reported for o1 mini under one evaluated condition	Comparative evaluation	[12]
MCPSecBench	2025	Multiple MCP platforms	Multiple tools	17 attack classes	Broad MCP attack surface	Multiple platform and attack failures reported	Benchmark based	[13]
MCP Security Bench	2025–2026	9 agents	405 tools	Multiple attack types	End to end agent failures	2,000 attack instances	Yes, comparative security evaluation	[10]
Protocol level analysis	2026	Multiple models	5 implementations	Protocol attacks	Architectural weaknesses	847 attack scenarios reported in emerging primary evidence	Yes	[11]
Confused deputy study	2026	14 models, 6 providers	2 MCP hosts	Tool selection hijacking	Malicious tool overshadowing	Up to 90.89% selection hijacking and 86.46%	Yes	[13]

Study or evidence stream	Year	Models or clients	Servers or tools	Attack tested	Main outcome	Quantitative finding	Defense tested	Citation (s)
						malicious execution		
Remote MCP server study	2026	ChatGPT, Claude Desktop, Gemini CLI	Malicious remote servers	Remote server compromise	Multiple LLM active and passive pathways	Platform dependent attack success	Mitigation and host controls considered	[12]
MCP credential ecosystem study	2025	Repository analysis	More than 5,200 repositories	Credential exposure risk	Static secret dependence	88% required credentials; 53% used long-lived static secrets	Proposed secret management mechanisms	[16]
Software supply-chain empirical research	2023	Multiple software ecosystems	Large dependency ecosystems	Supply chain compromise	Trusted distribution can propagate malicious code	Empirical attack case analysis	Defensive framework	[22],[23]
Vulnerability detection research	2025	Multiple coding models	Security vulnerable codebases	Implementation defects	Automated vulnerability identification	Multi class vulnerability detection demonstrated	Yes	[24]

MCP, Model Context Protocol; LLM, large language model. MCPTox, MCPSecBench, and MCP Security Bench are retained as names of specific MCP security benchmarks or studies.

2.5 Existing Defensive Mechanisms and Secure MCP Design Principles

2.5.1 Authentication and Authorization

Authentication and authorization should be treated as separate security functions. Authentication establishes who is participating in an interaction, while authorization determines what that participant may access or execute. In MCP, this distinction is particularly important because an authenticated server may still possess capabilities that are broader than those required by the user's task. Authorization should therefore be resource bound, capability specific, and time limited where practical. OAuth based mechanisms, protected resource metadata, authorization server discovery, PKCE, access token audience validation, resource indicators, short-lived tokens, token rotation, and narrow scopes can help prevent valid credentials from being reused outside their intended context. The broader security literature on agentic systems supports stronger separation between authentication and operational authority [10]. A valid token should not be treated as a universal permission to perform any action technically supported by the server.

2.5.2 Least Privilege and Capability Restriction

Least privilege should be implemented at every major layer. The MCP server should receive only the credentials required for its function. Individual tools should expose only the operations required for the task. File access should be limited to approved directories. Network access should be restricted to approved destinations. API scopes should be limited to the smallest useful set. This approach reduces blast radius when a tool is compromised or manipulated. It also limits the consequences of successful tool selection attacks because the malicious capability has less authority available to it.

2.5.3 Sandboxing and Process Isolation

Sandboxing is an important countermeasure for local and remote execution risks. An MCP server should ideally run in an isolated environment with restricted filesystem access, limited process privileges, constrained device access, and carefully controlled network connectivity. Process isolation is especially important for servers implemented using general purpose programming environments because these environments may expose powerful execution primitives. A malicious package or server compromise can otherwise result in arbitrary command execution with the permissions of the host process. The principle is consistent with defense in depth because isolation does not prevent every attack. Instead, it limits the consequences of a successful compromise.

2.5.4 Network Segmentation and Egress Control

Network egress control can significantly reduce the impact of data exfiltration and command and control behavior. A tool that only needs to access one external API should not automatically have unrestricted Internet access. Allow lists, network segmentation, proxy enforcement, DNS controls, destination validation, and monitoring of unusual outbound connections can provide additional protection. These measures are especially useful against cross-tool exfiltration because even if the attacker succeeds in influencing tool selection, the resulting server may still be prevented from communicating with unauthorized destinations.

2.5.5 Metadata and Tool Description Validation

Metadata validation is one of the most important MCP specific defensive controls because descriptions, names, parameter explanations, and annotations may influence LLM reasoning. Validation should combine structural and semantic techniques. Structural checks can identify malformed schemas, inconsistent parameters, suspicious permissions, and unexpected capability declarations. Semantic checks can look for hidden instructions, attempts to override higher priority policy, requests for secrecy, instructions to access unrelated resources, or suspicious descriptions of credential handling. Prompt injection defense research nevertheless shows that security filters must balance attack detection against false positives and degradation of benign functionality [17]. For that reason, metadata scanning should be treated as one layer of defense rather than the sole decision mechanism.

2.5.6 Provenance, Signing, Attestation, and Tool Identity

Strong tool identity requires more than a familiar name. Provenance mechanisms should establish publisher identity, artifact integrity, release history, dependency information, and the correspondence between the artifact that was reviewed and the artifact that is executed. Digital signatures, software bills of materials, cryptographic hashes, trusted publisher records, attestations, reproducible build information, and registry verification can strengthen this chain of trust. However, provenance cannot establish safety by itself. A legitimate publisher can unintentionally release a vulnerable package, and a compromised publisher account can sign a malicious artifact. Provenance must therefore be combined with security scanning and behavioral monitoring.

2.5.7 Tool Pinning and Rug Pull Detection

Tool pinning reduces the possibility that a previously approved component silently changes. Critical deployments should record the version, package digest, server identity, permission set, metadata fingerprint, dependency graph, and approved network destinations. A significant change in any of these properties should trigger a review. This is particularly important for descriptions because a malicious update may modify the language presented to the model

without producing an obvious change in package functionality. Software supply-chain research supports this principle because attackers can exploit the period between initial trust establishment and later software modification [22,23].

2.5.8 Runtime Monitoring and Behavioral Guardrails

Runtime monitoring is required because not every attack can be identified before execution. Security logs should capture the initiating principal, selected tool, server identity, input parameters, authorization decision, data accessed, external destination, and resulting side effects. Behavioral guardrails can then detect abnormal sequences such as a document retrieval followed by transmission to an unknown domain, access to a directory outside the expected task scope, repeated invocation of high risk tools, or unusual changes in tool behavior. Agent security research supports monitoring of the complete decision and action path because an agent can remain textually plausible while performing an unsafe external operation [10].

2.5.9 Static and Dynamic Security Scanning

Static scanning should examine the source code, dependencies, package manifests, secrets, configuration files, permission requests, filesystem operations, process execution, and network behavior of MCP servers. Dynamic testing should then execute the server under controlled workloads to evaluate runtime behavior. Automated vulnerability detection research indicates that modern machine-assisted techniques can identify a range of software vulnerabilities [24]. Such tools are useful for MCP server development pipelines but should be complemented with human review and adversarial testing. Security scanning should also be repeated whenever a server changes version, dependencies, permissions, or metadata. A one-time security assessment is insufficient for a component that can evolve after approval.

2.5.10 Defense in Depth Architecture

No single security mechanism adequately addresses MCP threats because attacks can cross semantic, protocol, application, operating system, network, and supply-chain boundaries. A secure MCP architecture should therefore combine identity verification, narrowly scoped authorization, least-privilege, metadata validation, sandboxing, network restriction, provenance controls, version pinning, runtime monitoring, dependency scanning, and incident response. The broader agent security literature supports this layered architecture because autonomous systems can be attacked through both conventional and AI specific mechanisms [10]. MCP research similarly demonstrates that different attack pathways can exploit different trust assumptions, making single control strategies inherently incomplete [12,13]. A practical secure lifecycle begins before a server is connected. At registration, publisher identity and artifact provenance should be established. Before connection, package integrity, dependencies, permissions, and expected network destinations should be reviewed. During discovery, metadata should be treated as untrusted input until validated. Before execution, authorization and policy checks should determine whether the requested action is permissible. During execution, sandboxing and network controls should restrict the blast radius. After execution, logs and behavioral analytics should support detection and attribution.

Table 5 summarizes the defensive mechanisms examined in Section 2.5 across the MCP security lifecycle. It links each control to the threat addressed, enforcement point, evidence status, and principal limitation, thereby distinguishing established security-engineering practices from controls that still require broader empirical validation.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Table 5. Defensive Controls for the MCP Security Lifecycle

Security layer	Threat addressed	Defensive mechanism	Enforcement point	Evidence status	Key limitation	Citation(s)
Identity	Impersonation	Server identity and provenance verification	Registry, host or client	Emerging	Ecosystem standardization remains incomplete	[11]
Metadata	Tool poisoning	Structural and semantic metadata scanning	Client or gateway	Empirical and emerging	False positives and adaptive attacks	[17]
Authorization	Confused deputy	OAuth based resource bound authorization	Authorization server and MCP server	Strong protocol basis	Implementation complexity	[10]
Privilege	Capability abuse	Least privilege and narrow scopes	Host and server	Strong security principle	Fine grained policies can be difficult	[10]
Execution	RCE and local compromise	Sandboxing and process isolation	Server runtime	Established engineering control	Compatibility and performance overhead	[26]
Network	Exfiltration	Egress filtering and segmentation	Gateway and network	Practical	Encrypted legitimate traffic can complicate inspection	[20]
Runtime	Tool abuse	Behavioral monitoring and action policies	Agent runtime	Emerging	Detection latency and false alarms	[27]
Supply chain	Malicious server or dependency compromise	Signing, SBOM, pinning and attestation	Registry and deployment pipeline	Established software security basis	Registry heterogeneity	[22],[23]
Code security	Implementation vulnerabilities	Static and dynamic security scanning	Development pipeline	Empirically supported	Coverage gaps remain	[24]
Privacy	Data leakage	Data minimization and context isolation	Host, client and tool	Established principle	Excessive restriction may reduce utility	[25]

Security layer	Threat addressed	Defensive mechanism	Enforcement point	Evidence status	Key limitation	Citation(s)
Defense in depth	Multistage attacks	Combined identity, policy, isolation and monitoring	Full lifecycle	Strong architectural recommendation	Increased operational complexity	[10],[26],[27]

MCP, Model Context Protocol; OAuth, Open Authorization; RCE, remote code execution; SBOM, Software Bill of Materials. Evidence status indicates whether a control is established, strongly supported, empirical, or still emerging.

III. RESULTS AND DISCUSSION

This section forms the principal analytical core of the review by integrating the architectural issues, threat mechanisms, empirical observations, and defensive considerations established in the preceding sections. The findings indicate that MCP security is not adequately described as a problem of malicious prompts alone. Instead, the major risks arise when natural language instructions, tool metadata, model-based selection, delegated authority, credentials, software dependencies, and runtime execution are connected within one operational workflow.

The evidence also shows that attacks against MCP can occur at several stages. Some attacks attempt to manipulate the model before a tool is called. Others exploit the authorization relationship between a server and an upstream service. Some take advantage of the ability of one trusted tool to pass information to another tool. Others enter through malicious packages, compromised repositories, vulnerable dependencies, or unsafe server implementations. The common feature across these categories is a failure to preserve trust as information and authority move through the system.

Recent peer-reviewed studies provide particularly important evidence for this interpretation. The 2026 systematic MCP study describes the protocol across creation, deployment, operation, and maintenance stages, emphasizing that security must be considered throughout the complete MCP server lifecycle [28]. A separate 2026 experimental study demonstrated that malicious remote MCP servers can exploit more than the tool description layer and can create multiple attack paths across different LLM platforms [29]. The confused deputy study further demonstrated that manipulated metadata can bias tool selection and allow an adversarial server to overshadow a legitimate alternative [30]. An IEEE Transactions on Software Engineering study extended the evidence into the broader MCP ecosystem by experimentally placing malicious MCP servers on aggregation platforms and demonstrating harmful actions against local environments [31].

3.1 Tool Poisoning, Tool Impersonation, and Manipulation of Tool Selection

3.1.1 Anatomy of MCP Tool Poisoning Attacks

Tool poisoning represents one of the clearest demonstrations of how MCP changes the traditional relationship between software documentation and system execution. In ordinary software, a function description is generally intended to help a programmer understand how to call an interface. In an LLM agent, however, descriptions can become part of the model's active reasoning context. A sentence that appears to be documentation to a human developer can therefore function as an instruction to the model.

The underlying attack has a simple structure. A malicious or compromised server publishes a legitimate looking tool. The server supplies a description that contains additional instructions or behavioral cues. The MCP client retrieves the description and makes it available to the LLM. The LLM interprets the malicious content as part of the context for performing the user's task. The model may then invoke the poisoned tool or another tool in an attacker preferred sequence. If the connected capabilities are sufficiently privileged, the attack can

progress from semantic manipulation to data access, credential use, external communication, or another unauthorized action.

The 2026 MCP landscape study places this problem within the wider server lifecycle and shows why tool poisoning should not be analyzed only at execution time. Security weaknesses can be introduced during server creation, deployment, registration, operation, or maintenance [28]. This observation is important because an organization that checks an MCP server only when it is initially installed may overlook later changes in its metadata or implementation.

The attack also demonstrates a fundamental mismatch between semantic trust and execution authority. The LLM is expected to reason about whether a tool is relevant, but relevance does not establish trustworthiness. A tool may be highly relevant to the task while still being malicious. Conversely, a trusted tool may be less semantically attractive than a malicious one. A secure design therefore needs a separate policy mechanism to decide which tool is actually permitted to act.

3.1.2 Hidden Instructions in Tool Descriptions and Schemas

Hidden instructions can be inserted into tool descriptions in several ways. The most obvious form is explicit text telling the model to perform an unrelated action. More subtle approaches may exploit descriptions that appear operationally reasonable but contain a secondary instruction. An attacker may also manipulate parameter descriptions, examples, error messages, annotations, or other contextual fields that influence how the LLM constructs a tool call.

The danger is amplified by the ability of LLMs to interpret natural language flexibly. A malicious instruction does not necessarily need to use an obvious phrase such as 'ignore previous instructions.' It may instead be framed as a security recommendation, compatibility requirement, diagnostic procedure, or necessary preprocessing step.

The malicious remote server evidence is important here because it demonstrates that the server itself can become an active adversarial component rather than merely serving as a passive source of metadata [29]. This distinction broadens the security problem. A server may attack through its descriptions, its returned data, its runtime implementation, or its interaction with external systems.

Another concern is schema manipulation. A parameter may appear to be an ordinary input while its description encourages the model to place sensitive information into it. A malicious tool may also provide apparently valid but unusually broad parameters. The result is that schema validation should include semantic review and permission analysis rather than checking only whether the JSON structure is syntactically correct.

3.1.3 Tool Shadowing and Cross Origin Escalation

Tool shadowing occurs when an attacker introduces a capability that resembles a legitimate tool closely enough to compete successfully for the model's attention. The malicious tool may have a similar name, an almost identical description, overlapping parameters, or a claim that it provides a faster or more complete implementation.

The confused deputy research provides strong empirical support for this threat. The authors demonstrated that manipulated descriptions and name-based preferences can cause a malicious server to overshadow a benign server, allowing tool invocations to be redirected without necessarily producing an obviously malicious interaction [30]. The finding is important because it challenges the assumption that an authenticated tool list is necessarily a trustworthy tool list.

Cross origin escalation becomes possible when the selected tool is connected to a different trust domain. For example, an apparently harmless weather or document utility could gain access to information through another tool and then communicate that information to an external system. The attack therefore crosses boundaries that were not visible when each capability was evaluated separately.

Tool shadowing is particularly concerning in environments with large tool sets. As the number of candidates increases, the model must discriminate

among more semantically similar descriptions. The resulting selection process becomes an increasingly important security control point.

3.1.4 Tool Impersonation, Name Collision, Typosquatting, and Identity Spoofing

Tool impersonation exploits the tendency of users and models to infer identity from names and recognizable descriptions. A malicious publisher can adopt a name that resembles an official integration, create a package with a near identical spelling, or use a description that strongly resembles a known provider.

The problem becomes more difficult when tools are obtained from public aggregation platforms. The IEEE TSE study published in 2026 experimentally investigated this ecosystem level problem. The researchers were able to upload malicious MCP servers to three widely used aggregation platforms, observed limitations in current auditing mechanisms, and found through a study involving 20 participants that users frequently had difficulty identifying malicious servers and could unknowingly install them [31].

These findings indicate that name-based trust is inadequate. The fact that a tool appears beside legitimate tools in a catalog does not establish its provenance. Likewise, a publisher name does not provide cryptographic evidence that the software is controlled by that publisher.

Typosquatting is especially relevant to package based MCP installation. A malicious package can differ from a trusted package by one character and still appear plausible to a user. The risk increases when installation is automated or when a developer copies package identifiers without independently checking publisher identity. In such cases, secure MCP deployment requires provenance, package integrity verification, trusted registry information, and preferably cryptographically anchored identity.

3.1.5 Preference Manipulation and LLM Tool Selection Bias

Tool selection is a central decision point in an MCP workflow because the model determines which capability is used to satisfy the user's request. Preference manipulation attempts to influence this ranking process rather than directly forcing an obviously harmful action.

The practical concern is that a malicious tool can be designed to look more relevant than the legitimate alternative. Descriptions may emphasize keywords that closely match common user requests. Tool names may be optimized for semantic prominence. Capability claims may be written in ways that encourage the model to prefer one option even when the actual implementation is less trustworthy.

The confused deputy evidence is important because it demonstrates that this problem is not merely conceptual [30]. The study reported selection hijacking rates as high as 90.89% under its tested conditions, with malicious payload execution reaching 86.46%. These figures should not be interpreted as universal attack probabilities because they depend on the evaluated models, prompts, tools, hosts, and attack construction. They nevertheless establish that model mediated tool selection can be a highly effective attack surface under realistic experimental conditions.

The implication is that tool selection should not be delegated entirely to semantic similarity. A security-aware selection layer should consider provenance, publisher identity, capability risk, permissions, destination, previous behavior, and organizational policy. A tool should not be selected merely because its description gives the strongest textual match.

3.1.6 Effect of Tool Description Complexity and Tool Set Size

Tool descriptions create an information problem as well as a security problem. When a tool set contains only a few simple capabilities, the model may have a relatively clear distinction among them. As the number and complexity of tools increase, the model must reason across a larger set of overlapping descriptions.

Complex descriptions can create additional opportunities for malicious content to be hidden. They also make manual review more difficult because users may not inspect every field supplied by every server. This creates a practical asymmetry: legitimate

developers may attempt to provide detailed descriptions to improve usability, while attackers can use the same descriptive surface to manipulate selection.

The current MCP literature suggests that security evaluations should therefore report tool set size and description structure rather than merely stating which model was tested [28]. The same model may behave differently when exposed to a small, curated tool set compared with a large ecosystem containing many semantically overlapping capabilities.

This problem also interacts with model context limits. A larger tool set can increase contextual competition and reduce the relative prominence of legitimate policy instructions. Consequently, tool inventory management itself should be treated as a security measure. Organizations should not connect every available server simply because the protocol makes integration convenient.

3.1.7 Model and Client Specific Susceptibility

Attack performance is not constant across LLMs or clients. The malicious remote MCP server study found meaningful platform differences in attack behavior [29]. This is important because a mitigation that works with one host may not provide equivalent protection in another.

The same principle applies to model versions. Tool calling behavior, instruction following, context prioritization, safety filters, and tool selection strategies can all change between models. Consequently, a security claim about one model configuration should not automatically be generalized to an entire product family.

The model dependence of these attacks has an important practical consequence. Security testing should be conducted against the actual model and client combination used in deployment. Substituting a model with another model of similar capability is not necessarily a valid security equivalence test.

It is also important not to interpret greater model capability as automatic security improvement. Stronger instruction following can improve legitimate tool use while simultaneously giving an attacker a more responsive target when the malicious instruction enters an apparently trusted part of the context. The recent agent security literature consistently emphasizes this duality between capability and exposure [32,33].

3.1.8 Comparative Evaluation of Detection Approaches

Detection mechanisms can operate at several levels. Structural scanners can inspect names, schemas, permissions, package metadata, and configuration. Semantic scanners can look for suspicious instructions or inconsistencies between descriptions and capabilities. Provenance systems can verify publisher identity and artifact integrity. Runtime systems can monitor tool selection and behavior.

The important finding from recent work is that no single layer is sufficient. A metadata scanner may identify suspicious instructions but miss a malicious dependency. A package scanner may detect a vulnerable library but miss an attack that is carried entirely through a description. Runtime monitoring may identify an anomalous action after the attack has already occurred.

The 2026 research literature therefore favors layered security controls and more rigorous evaluation metrics [32,34]. Defense quality should be measured not only by reduction in attack success but also by false positives, legitimate task performance, response latency, action availability, and the ability to withstand adaptive attacks.

Table 6 compares the principal empirical results discussed in Section 3.1, including tool-poisoning and tool-selection studies. It separates benchmark-derived measurements from peer-reviewed experimental evidence and records tool-set scale, attack outcomes, detection characteristics, latency considerations, and mitigation approaches.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Table 6. Comparative Empirical Performance of MCP Tool Poisoning and Selection Attacks

Study	Model or client	Tool set size	Attack type	Attack success or unsafe invocation	Detection	Latency	Mitigation	Citation(s)
MCPTox	Multiple LLM agent configurations	353 authentic tools from 45 live servers	Tool poisoning	72.8% ASR reported for GPT o1 mini under the published benchmark condition	Benchmark refusal and attack outcome analysis	Reported by benchmark; not independently standardized across models	Comparative defenses and refusal analysis	[28]
MCPSecBench	Multiple providers	Multiple MCP tool sets	Multiple attack classes	Broad failures across discovery, invocation, parameter and response stages	Benchmark based	Not standardized across platforms	Attack and defense comparison	[28]
MSB	9 agents	405 tools	12 attack classes and mixed chains	2,000 attack instances; model vulnerability varied by attack stage	Net Resilient Performance and attack outcomes	Benchmark dependent	Robustness evaluation	[32]
MPMA	Multiple model and tool conditions	Variable	Preference manipulation	Effective manipulation of tool preference demonstrated experimentally	Selection comparison and stealth analysis	Attack dependent	Preference manipulation defenses proposed	[30]
Confused deputy experiment	14 models, 6 providers, 2 MCP hosts	Competing legitimate and adversarial tools	Tool selection hijacking	Up to 90.89% selection hijacking; up to 86.46% malicious execution	Security evaluation framework	Configuration dependent	Automated evaluation and security scanning	[30]
Remote MCP server experiment	ChatGPT, Claude Desktop and	Multiple malicious remote servers	Server level attack pathways	Platform dependent attack success across five	Host and server control	Platform dependent	Filtering, auditing and hardening approaches	[29]

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Study	Model or client	Tool set size	Attack type	Attack success or unsafe invocation	Detection	Latency	Mitigation	Citation(s)
	Gemini CLI			attack scenarios	s considered			
Current indirect injection defense evaluation	GPT 5.4, GPT 5.4 mini and Claude Sonnet 4.6	AgentDojo banking tasks	Indirect prompt injection	Undefended ASR was 0/288 for GPT 5.4, 11/288 for GPT 5.4 mini and 1/288 for Claude Sonnet 4.6 under the evaluated setup	Four defenses evaluated	Operational metrics included	Security utility trade off analysis	[34]

MCP, Model Context Protocol; LLM, large language model; ASR, attack success rate; MSB, MCP Security Bench; MPMA, the manuscript's benchmark label for preference manipulation research. Benchmark results are dependent on the evaluated models, tools, and experimental conditions.

Figure 2 complements Section 3.1 by depicting the attack lifecycle from poisoned metadata and semantic manipulation through model-mediated tool selection and into downstream system compromise or data exfiltration. The schematic highlights the point at which a semantic attack can cross into conventional execution and infrastructure risks.

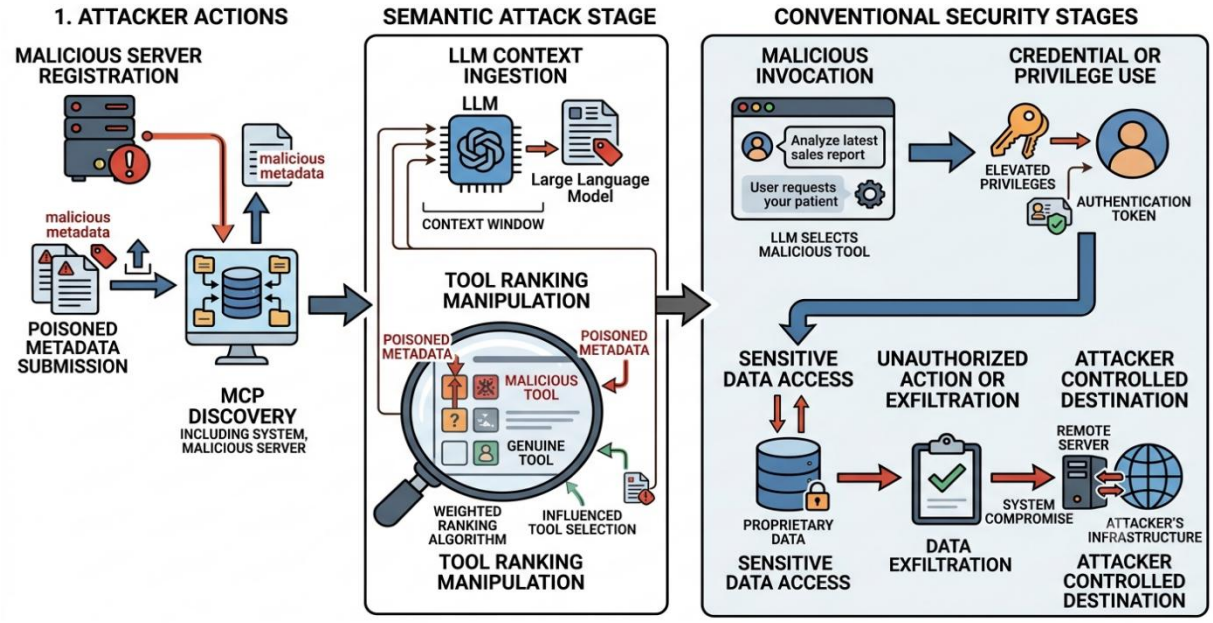

Figure 2. End-to-end lifecycle of MCP tool poisoning and selection manipulation [28-31].

Sequential schematic detailing the transition from semantic context manipulation (poisoned metadata ingestion and tool ranking bias) to conventional system compromise and exfiltration. MCP, Model Context Protocol; LLM, large language model; ID, identifier; API, application programming interface; URL, uniform resource locator.

3.2 Confused Deputy Attacks, Credential Abuse, and Privilege Escalation

3.2.1 MCP as an Authorization Intermediary

MCP servers increasingly operate as intermediaries between users, LLM agents, and external services. This makes the server more than a technical transport component. It becomes an authorization intermediary that can potentially act with privileges greater than those available to the user through the immediate interface.

This intermediary role creates a confused deputy condition when the downstream service authenticates the MCP server but cannot determine whether the particular action was actually authorized by the original user. The agent may request an operation that is technically possible because the server's credentials allow it, even though the user intended a much narrower action.

The security literature on LLM based agents emphasizes that authorization cannot be reduced to authentication because autonomous tool use introduces a distinction between what an agent can technically do and what it should be permitted to do [32,33]. That distinction becomes especially important where an MCP server aggregates several upstream capabilities.

3.2.2 Confused Deputy Attack Mechanics

A typical confused deputy sequence begins with a legitimate user request. The user interacts with an MCP enabled host. The LLM determines a task and chooses a server. The server possesses a token or service credential that gives it authority over an external resource. An attacker then manipulates the model context, tool metadata, request parameters, or authorization flow so that the server performs a broader operation than the user intended.

The server has not necessarily been compromised at the code level. It may be behaving exactly as programmed. The security failure occurs because it is unable to distinguish legitimate delegated authority from attacker induced authority.

This problem is particularly difficult for automated systems because the request may look syntactically correct. A downstream API may see a valid token, a valid endpoint, and a valid request body. Nothing in the network exchange necessarily reveals that the semantic decision leading to the request was manipulated.

3.2.3 Token Audience Confusion and Token Passthrough

A bearer token should not automatically be reusable across multiple resource servers. If a token issued for one audience is accepted by another service, the security boundary between those services is weakened.

Token passthrough creates another related risk. An MCP server may receive a token intended for one upstream service and simply forward it to a downstream service without ensuring that the token was intended for that destination. Such behavior can enlarge the effective privilege associated with the original authorization.

The emerging MCP authorization literature increasingly emphasizes resource specific binding and proper audience validation. Peer reviewed work examining OAuth based AI automation likewise demonstrates the importance of contextual identity verification and policy driven authorization when AI systems act on behalf of users [39].

The principle is straightforward: a token should represent a narrowly defined authorization relationship, not a general purpose key that any connected component can reuse.

3.2.4 Authorization Code Theft and Dynamic Client Registration Risks

Authorization code based workflows create a separate attack surface. If an attacker can intercept, redirect, or

misuse an authorization code, the attacker may be able to obtain access to a downstream resource.

Dynamic client registration can further complicate trust because clients may be created or registered without the same administrative controls used for conventional enterprise applications. In an MCP ecosystem containing many third-party servers, organizations need to know whether a newly registered client is authentic, what software it represents, who controls it, and what permissions it requests.

The policy driven OAuth literature supports the broader principle that AI enabled automation requires contextual risk evaluation and reliable identity association rather than relying only on a successful authentication exchange [39].

For MCP, the practical implication is that authorization server discovery should not be treated as a security endpoint in itself. The authorization flow must also establish who requested access, which resource is being accessed, which tool is acting, what scope was granted, and whether the resulting token can be used anywhere else.

3.2.5 Excessive MCP Server Privileges

Excessive privilege is one of the most consequential design weaknesses because it converts a successful semantic attack into a potentially high-impact system compromise. A server connected to a filesystem with unrestricted read and write permission presents a very different security profile from a server restricted to a single application directory.

The same principle applies to cloud credentials. A tool requiring access to a single calendar might receive a credential with broad access to the entire enterprise workspace. If the server is compromised, the attacker acquires authority that far exceeds the original task.

Recent research on agentic security repeatedly identifies least-privilege as a core protection because autonomous systems can chain otherwise legitimate capabilities into harmful workflows [32,33]. The security value of least-privilege is therefore not limited to preventing direct exploitation. It also restricts the consequences of successful tool poisoning, confused deputy behavior, and cross-server attacks.

3.2.6 Credential Storage and Secret Exposure

Credential storage remains a practical weakness across many tool-enabled systems. Static API keys may be stored in environment variables, configuration files, source code, container images, secret stores, deployment scripts, or local user directories.

Static secrets are particularly dangerous in MCP because a server may have persistent access even after a model session has ended. If that server is compromised, the attacker may obtain a credential that can later be used outside the original agent interaction. The broader LLM security literature has emphasized that sensitive information can leak through model contexts, tool inputs, application logs, or connected services [9,36]. In an MCP deployment, the credential should therefore remain outside the model context whenever possible. The model should request an operation, while the host or secure execution layer supplies the authorization material without exposing the raw secret.

3.2.7 Cross Server Credential and Permission Propagation

Cross server propagation creates a lateral privilege problem. One server may obtain a credential, data object, or privileged result and then provide it to another server through the model context or tool arguments.

This means that security policy should not stop at the server boundary. The architecture must control how capabilities and sensitive information move between servers. A token that is safe within one server can become dangerous when transferred into another trust domain.

Capability oriented approaches offer a promising direction because they can make permissions explicit and potentially attenuate them as work is delegated. Recent capability token research for multi agent LLM systems proposes cryptographically bound permissions and provenance tracking to control which agents can access which fields or resources [40].

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Although such approaches remain emerging in the MCP context, their architectural principle is highly relevant: delegated authority should become narrower, not broader, as a task moves through multiple components.

3.2.8 Least Privilege and Fine Grained Tool Authorization

Fine grained authorization should operate at the level of individual actions rather than only at the level of servers. A server may legitimately provide ten tools, but a particular user or workflow may be authorized to use only two.

The ideal model is therefore: identity → task → capability → resource → action → destination → time limit. Each link should be independently enforceable. This creates an explicit relationship between what the user requested and what the tool can do.

Capability based research supports this direction. CapChain, for example, proposes capability tokens that bind permissions to an agent, resource, field, and data-flow, while maintaining tamper evident provenance records [40]. Although this is not yet a universal MCP standard, it illustrates a promising pathway for reducing broad delegated authority.

3.2.9 Current OAuth 2.1 Based MCP Security Requirements

Current MCP deployments increasingly require stronger OAuth practices because token based authorization is one of the main interfaces between an MCP server and protected external resources. Important requirements include authorization server discovery, protected resource metadata, PKCE, explicit audience validation, resource indicators, narrow scopes, short token lifetimes, secure refresh handling, and avoidance of token passthrough.

The key principle is that authentication should answer 'who is this?' while authorization should answer 'what can this actor do to this particular resource under this particular task?' A token should not be interpreted as a blanket statement of authority.

Table 7 summarizes the authorization, credential, and privilege risks examined in Section 3.2. It connects each vulnerability to its attack mechanism, credential type, trust boundary, potential privilege gained, current mitigation, and remaining implementation gap.

Table 7. MCP Authentication, Authorization, Credential, and Privilege Risks

Vulnerability	Attack mechanism	Credential involved	Trust boundary	Potential privilege gained	Current mitigation	Remaining gap	Citation(s)
Confused deputy	Authorization delegation abuse	OAuth access token or authorization code	Client, server and authorization server	Downstream resource access	Audience validation and resource binding	Implementation consistency	[32]
Token passthrough	Forwarding token to an unintended downstream service	Bearer access token	Server and resource server	Unintended API authority	Destination and audience validation; prohibit passthrough	Legacy and inconsistent implementations	[39]
Authorization code theft	Code interception or redirect abuse	Authorization code	Client and authorization server	Token acquisition	PKCE and secure redirect handling	Complex multi component flows	[39]

Vulnerability	Attack mechanism	Credential involved	Trust boundary	Potential privilege gained	Current mitigation	Remaining gap	Citation(s)
Credential exposure	Static secret leakage	API key or long-lived token	Runtime environment and server	Service impersonation	Secret management and short-lived credentials	Persistent static secret use	[36]
Excessive privilege	Broad tool scope	User or service credential	Host and server	High impact actions	Least privilege and scoped authorization	Fine grained capability models	[32]
Cross server propagation	Delegated authority transferred between tools	OAuth or service token	Server to server boundary	Lateral privilege	Scoped tokens and isolation	Complex orchestration	[40]
Context credential leakage	Secret enters model context or logs	API key, token or password	Host and LLM context	Credential theft	Secret isolation and redaction	Tool implementation variation	[36]
Over persistent authorization	Long lived authorization survives the task	Refresh token or static secret	Session and resource boundary	Reuse after intended task	Short expiry and revocation	Operational complexity	[39]

MCP, Model Context Protocol; OAuth, Open Authorization; PKCE, Proof Key for Code Exchange; API, application programming interface; LLM, large language model. Credential and authorization entries distinguish access tokens, API keys, static secrets, and other authorization materials according to their role in the attack pathway.

Figure 3 complements Section 3.2 by mapping the principal trust relationships among the LLM, MCP host, client, server, and OAuth authorization ecosystem. It identifies attack points involving authorization-code theft, token-audience confusion, token passthrough, and excessive scope assignment.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

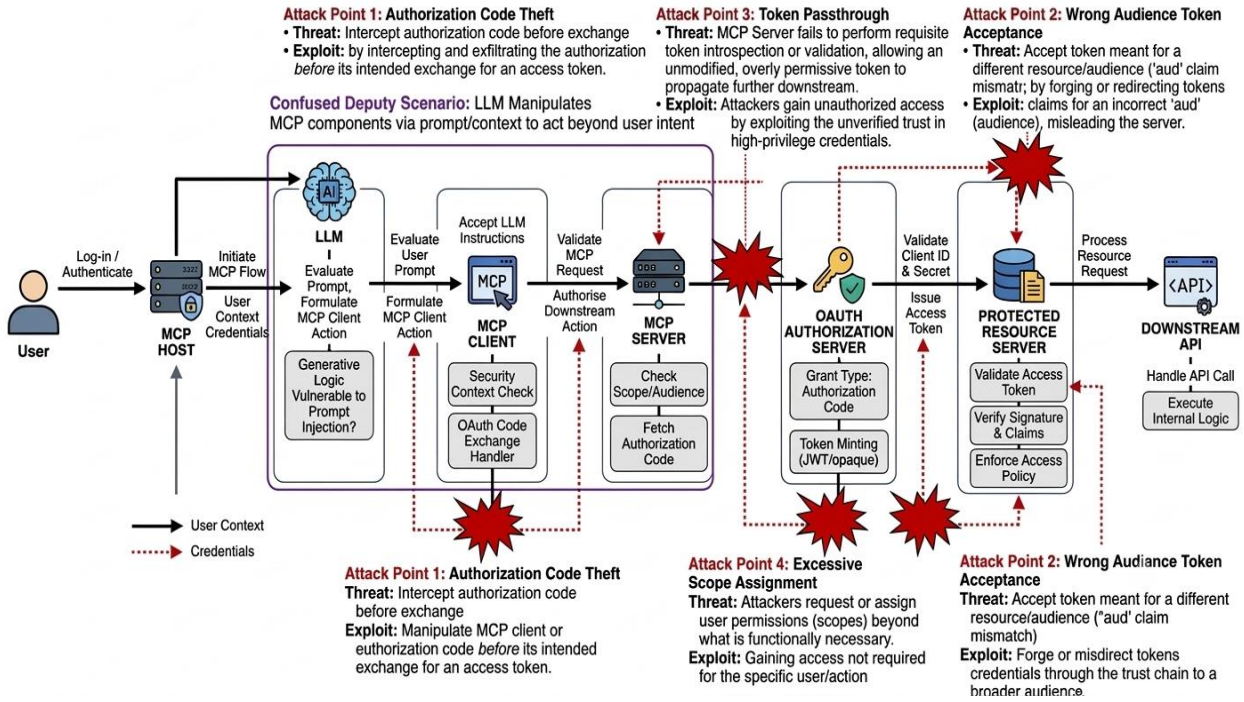

Figure 3. Comprehensive threat model of a multilayer MCP architecture: from prompt injection to token abuse

This comprehensive diagram details a multi-layered Model Context Protocol (MCP) architecture and analyzes potential attack points. It visualizes the flow of user context and credentials, highlighting critical vulnerabilities such as authorization code theft (Attack Point 1), wrong audience token acceptance (Attack Point 2), token passthrough (Attack Point 3), and excessive scope assignment (Attack Point 4).

MCP, Model Context Protocol; LLM, large language model; AI, artificial intelligence; OAuth, Open Authorization; API, application programming interface; JWT, JSON Web Token; aud, audience.

3.3 Cross Tool Data Exfiltration and Multi Server Attack Chains

3.3.1 Shared Context as an Exfiltration Enabler

The shared context of an LLM agent is one of MCP's greatest functional advantages and simultaneously one of its most important security risks. A model can combine information from different tools and use it to perform complex tasks. However, the same capability can allow data retrieved from one trust domain to be passed silently into another.

The key issue is that the model often acts as the information broker. The source tool may provide sensitive information to the model, and the model may subsequently include that information in arguments to another tool. No direct network connection between the two servers is necessary.

Recent agent security research describes information leakage as a major class of agent threat because agents continuously move information between memory, retrieval systems, external tools, and action channels [33,37]. MCP intensifies this concern by making heterogeneous tools easier to connect.

3.3.2 Malicious to Trusted Tool Chaining

Tool chaining occurs when one tool's output becomes another tool's input. This can be beneficial when the sequence is intentional, but it creates a serious confidentiality problem when the first tool is malicious or when the second tool is manipulated.

Consider an agent with a document search tool and an email tool. The search capability may retrieve confidential information from a private repository. The email capability may be permitted to send

messages externally. If a malicious server can cause the model to combine the two capabilities, confidential information can leave the organization without either individual tool appearing obviously defective.

The agent literature shows that composed and chained attacks are often more difficult to detect than isolated attacks because the malicious objective can be distributed across multiple otherwise legitimate operations [33].

3.3.3 Cross Server Data Discovery

A malicious server may not request sensitive information directly. Instead, it can encourage the model to search for relevant files, query a database, inspect memory, or access an application. Once the information is available in context, the attacker can attempt to redirect it to an external destination.

The 2026 MCP malicious server research is important because it demonstrated that remote servers can attack through broader server level pathways rather than only through tool descriptions [29]. This means that data acquisition and exfiltration should be modeled together.

A secure MCP architecture should therefore distinguish between data discovery and data release. The fact that a tool may read a resource does not mean that the resulting information can be given to another tool or external service.

3.3.4 Sensitive File, Credential, Memory, and Application Data Extraction

Sensitive data can enter the model context from many locations. Local files may contain financial records, passwords, application configurations, SSH credentials, database credentials, or private research materials. Memory systems may contain user preferences or confidential conversation history. Business applications may expose customer records, documents, messages, or transaction data.

A malicious tool can therefore target several classes of information without possessing direct access to each source. It may simply manipulate the agent into retrieving the information through legitimate capabilities.

This is why MCP data security should use classification and sensitivity labels. A confidential file should not become an unrestricted input to a tool merely because that tool is connected. Similarly, secrets should be marked as non-exportable wherever practical.

3.3.5 Covert Data Transfer through Legitimate Tools

Attackers do not necessarily need an obviously malicious network tool to exfiltrate information. A legitimate communication tool can be abused to send data to an attacker controlled address. A file conversion tool may encode information into a generated document. A logging tool may be instructed to record sensitive content in an externally visible location.

The important distinction is between tool legitimacy and action legitimacy. A tool may be perfectly legitimate while a particular use of it is not.

This insight is reflected in recent runtime enforcement research. Wang and colleagues implemented a policy enforcement point that operates at the tool call boundary and combines cross step information flow labels with policy decisions. Under the evaluated controlled conditions, the full system reduced attack success from 40.0% to 5.0%, compared with 35.0% for the strongest prompt only baseline [41].

3.3.6 Cross Prompt Injection and Toxic Flows

A toxic flow occurs when malicious information introduced at one point in an agent workflow influences later actions. For example, a malicious webpage can inject instructions into the model, which then selects an MCP tool that accesses local files. The resulting data can then be transferred through another tool.

The significance of this pattern is that the original attacker controlled input may be several steps removed from the final security impact. Traditional monitoring focused only on the final action may fail to identify the source of the compromise.

Recent indirect prompt injection research shows why realistic evaluation must examine the entire agent

trajectory rather than testing only direct user prompts [34]. The 2026 defense evaluation further found that security measurements should report attack outcomes together with benign utility, action availability, operational behavior, and run to run variability rather than relying solely on attack success rate [34].

3.3.7 Data Flow Security and Taint Tracking

Information flow control provides a more principled response to cross-tool exfiltration. Data can be labeled according to sensitivity and origin, and policies can determine which tools are allowed to receive or release particular classes of information.

The challenge is that conventional data-flow mechanisms operate on structured program states, while LLM agents manipulate information semantically. The system may need to determine that a paragraph returned by one tool contains confidential financial information even when the exact string is transformed by the model.

Recent capability and semantic data-flow research attempts to bridge this gap. CapAgent proposes a governance layer that associates user intent, data objects, actions, destinations, and semantic release decisions with enforceable capability controls [40]. The importance of this research for MCP is that it shifts attention from prompt instructions to actual information release conditions.

Similarly, recent empirical work using data-flow diagrams for security threat validation suggests that information flow representations can help structure the identification and analysis of security boundaries in complex software systems [47].

3.3.8 Egress Filtering and Destination Control

Egress filtering restricts where an MCP server or tool can send data. It is especially useful because it remains effective even when the LLM has already been manipulated.

A tool that is authorized to access private files but is unable to establish connections to unapproved external domains has a smaller exfiltration surface than an otherwise identical tool with unrestricted Internet access.

Destination control should ideally be contextual. A weather tool may legitimately require access to one weather API, while a file management tool may require no external Internet access at all. Network policies should therefore be derived from actual capability requirements rather than simply allowing general HTTPS connectivity.

Egress control also supports incident investigation. Unexpected connections to new domains can become behavioral indicators of compromise, particularly when combined with tool invocation logs and data sensitivity information.

3.3.9 Runtime Detection of Sensitive Data Flows

Runtime detection should identify suspicious relationships among the source of information, the selected tool, the destination, and the requested action. Static scanning cannot predict every possible tool sequence, particularly in systems whose behavior changes dynamically with user requests.

Recent MCP runtime enforcement research demonstrates the potential value of an interception layer that tracks information flow across multiple steps rather than evaluating each tool call independently [41]. This is particularly relevant for cross-server workflows because the security boundary often lies between calls rather than inside one call.

Table 8 summarizes the cross-tool and cross-server exfiltration pathways discussed in Section 3.3. The matrix traces sensitive data from its source through intermediate capabilities to the release mechanism, required privileges, impact, and supporting evidence, highlighting how individually plausible tool actions can form harmful composite workflows.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Table 8. Cross Tool and Cross Server Data Exfiltration Pathways

Attack chain	Source of sensitive data	Intermediate tool	Exfiltration mechanism	Required privilege	Impact	Demonstrated evidence	Citation (s)
Banking data → weather utility → external destination	Financial or banking data	Malicious weather style tool	Legitimate outbound tool or API request	Tool invocation and contextual data access	Financial data disclosure	MCP and agent attack literature	[29]
Local file → malicious or compromised tool → network service	Confidential local files	Compromised MCP tool	Network or API request	Filesystem read and network access	Confidentiality loss	Empirical MCP attack studies	[31]
Credential store → malicious tool → external service	Environment variables or token store	Malicious MCP tool	Web or API request	Secret read access	Account impersonation	Agent and MCP security research	[33]
Trusted server output → malicious server	Sensitive tool output	Second MCP server	Prompt or tool argument propagation	Cross tool invocation	Sensitive information leakage	Information flow research	[40]
Private document → transformation tool → communication tool	Confidential document	Conversion or processing tool	Legitimate communication channel	Read access plus communication authority	Data disclosure	Tool chain security literature	[33]
Database record → summarization tool → external endpoint	Structured enterprise data	LLM summarization tool	Generated text transmitted through another tool	Database read plus export capability	Privacy breach	Agent security literature	[34]
Secret context → authorized export tool	Credential or confidential context	Communication or storage tool	Approved but attacker induced export	Context access plus export right	Credential compromise	Runtime enforcement evidence	[41]

MCP, Model Context Protocol; LLM, large language model; API, application programming interface. The pathways emphasize that sensitive data may be disclosed through individually legitimate tools when their capabilities are improperly combined.

© SEP 2026 | IRE Journals | Volume 10 Issue 3 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I3-1723420

Figure 4 illustrates the cross-tool and cross-server data exfiltration pathways discussed in section 3.3, showing how sensitive information can move from trusted resources through the model context and connected tools to an attacker-controlled endpoint. The figure highlights how legitimate tool combinations can be exploited to bypass trust boundaries and facilitate unauthorized data disclosure.

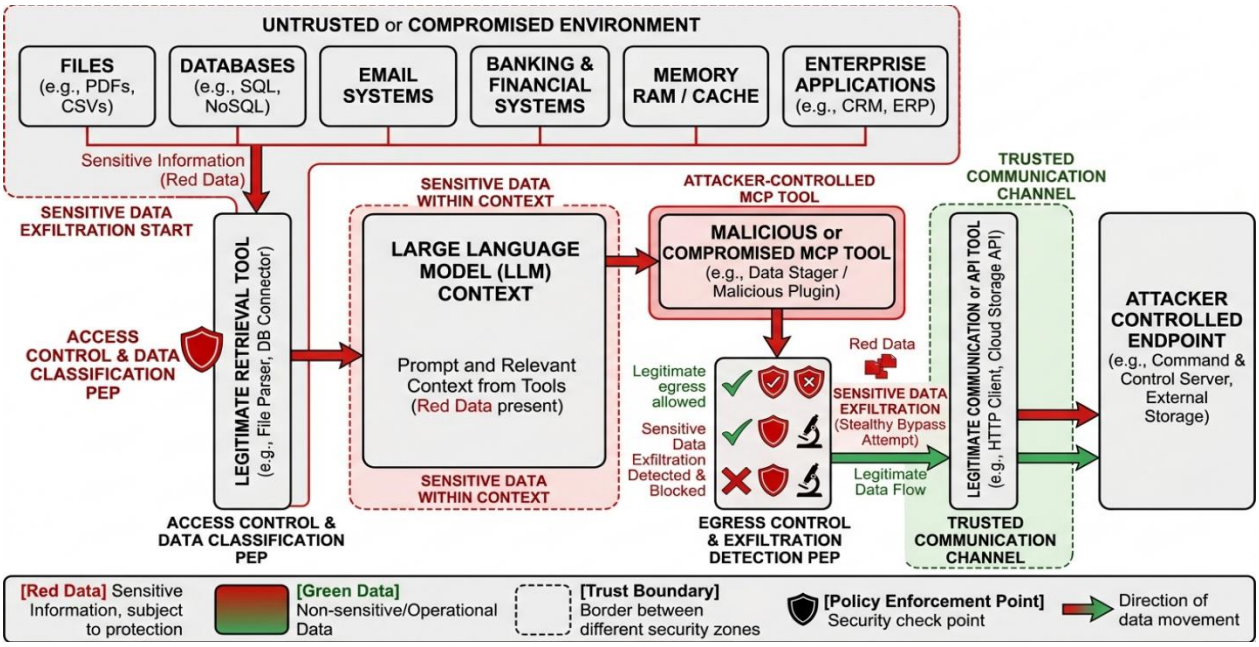

Figure 4. Cross-tool data exfiltration pathway in an MCP agent

The diagram illustrates the cross-tool data-flow problem, showing how sensitive information can pass from trusted resources through model context to a malicious or manipulated capability and then to an attacker-controlled endpoint. It also distinguishes legitimate flows, trust boundaries, and policy-enforcement points. CSV, comma-separated values; CRM, customer relationship management; DB, database; ERP, enterprise resource planning; HTTP, Hypertext Transfer Protocol; LLM, large language model; MCP, Model Context Protocol; PDF, Portable Document Format; PEP, policy enforcement point; RAM, random-access memory; SQL, Structured Query Language.

3.4 MCP Supply Chain Risks, Malicious Servers, and Implementation Vulnerabilities

3.4.1 MCP Servers as Software Supply Chain Dependencies

Every third-party MCP server should be treated as a software supply-chain dependency. The user may perceive the server as simply an additional capability, but technically it represents executable software with dependencies, configuration, update mechanisms, credentials, network permissions, and potentially access to sensitive resources.

The MCP lifecycle perspective is important because supply-chain risk begins before installation [28]. A server can be compromised during development, packaging, publication, deployment, update, or operation.

This means that organizations should ask where the code originated, who maintains it, how it was built, which dependencies it uses, whether its release artifact is authentic, and what changed between versions.

3.4.2 Malicious Packages and Registry Abuse

Public package registries and aggregation platforms lower the barrier to MCP adoption but can also create opportunities for malicious publishers. The 2026 IEEE TSE study demonstrated this directly by placing malicious MCP servers on three widely used

aggregation platforms and observing limitations in current audit mechanisms [31]. The researchers also found that users had substantial difficulty distinguishing malicious servers from legitimate ones during their user study.

The finding suggests that registry presence should not be treated as equivalent to security endorsement. A registry can provide discoverability without providing sufficient assurance about code quality, identity, or behavior.

Registry security should therefore include publisher verification, package scanning, vulnerability status, dependency analysis, artifact signing, reputation information, update history, and potentially sandboxed pre deployment execution.

3.4.3 Repository Impersonation and Fake MCP Servers

Repository impersonation may involve copying project names, logos, documentation, descriptions, or repository structures. A malicious server can therefore appear legitimate to both users and models.

The problem is particularly challenging where discovery is driven by natural language queries. A model may choose among several search results based on semantic relevance without establishing cryptographic provenance.

The IEEE TSE findings suggest that human users themselves may have difficulty recognizing malicious servers [31]. This means an LLM should not be expected to solve the problem purely through semantic reasoning. Identity requires external evidence.

3.4.4 Typosquatting and Malicious Forks

Typosquatting creates malicious packages whose names differ slightly from popular packages. Malicious forks create another challenge because legitimate open source projects commonly allow third parties to modify and redistribute code.

The problem is not simply identifying whether a fork exists. The question is whether the fork's publisher, changes, dependencies, and release pipeline are trusted.

Recent empirical work on software supply-chains shows that dependency relationships can cause vulnerability risks to propagate across multiple software components [43]. For MCP, the effect can be more severe because the dependent component may have direct access to an agent's tools and credentials.

3.4.5 Dependency and Transitive Package Risks

An MCP server may depend on dozens or hundreds of external packages. Some of those dependencies may themselves depend on additional packages.

This creates a transitive trust problem. The MCP server developer may have reviewed the direct dependencies while having little visibility into deeper components. A compromised dependency can therefore enter a trusted MCP server without obvious modification to the server's own code.

An empirical study of real-world software supply-chain vulnerabilities found that dependencies influence vulnerability propagation, localization, repair cost, and the broader exposure of client programs [43]. The result strongly supports the use of dependency inventories and continuous vulnerability monitoring for MCP servers.

3.4.6 Credential Exposure in MCP Server Code

Supply chain compromise is especially dangerous when servers contain embedded credentials. A malicious package can search configuration files, environment variables, credential stores, or deployment metadata for secrets.

The preferred design is to keep credentials outside the server code and inject narrowly scoped authorization material only when required. Secrets should not be hard coded, logged, returned through tool responses, or included in model visible context.

The current agent security literature supports this separation because data and credential leakage are major components of the broader agent threat surface [32,33].

3.4.7 Post Approval Modification and MCP Rug Pulls

A rug pull exploits the assumption that a trusted component remains trustworthy after approval. A server may be benign when installed but later receive an update that changes its executable code, dependencies, tool descriptions, permissions, or external destinations.

The supply-chain literature demonstrates that software compromise can occur through trusted update mechanisms [22]. In the MCP environment, the impact extends beyond executable code because the updated metadata may also alter the way the LLM interprets the tool.

Version pinning and artifact hashes can reduce the risk, but they create an operational trade off. Very rigid pinning can delay security patches, while automatic updates can introduce unreviewed behavior. Organizations therefore need controlled update pipelines in which changes trigger revalidation.

3.4.8 Path Traversal, Argument Injection, Symlink and Filesystem Vulnerabilities

Conventional software vulnerabilities become particularly consequential when present in MCP servers. A path traversal flaw can expose files outside the intended directory. An argument injection flaw can allow a user supplied value to alter command execution. Symlink handling weaknesses can cause a supposedly restricted path to resolve to a sensitive target.

The 2026 comparative study of LLM agent deployment paradigms is valuable because it directly compares MCP with function calling and evaluates both AI specific threats and conventional software vulnerabilities. Across 3,250 attack scenarios and seven models, the study reported an overall attack success rate of 62.6% for MCP compared with 73.5% for function calling [42].

The result should not be interpreted as MCP being intrinsically secure. Instead, it demonstrates that MCP and function calling exhibit different distributions of security risk and that both AI specific and conventional software vulnerabilities need to be evaluated.

3.4.9 SDK Level and Stateless Deployment Vulnerabilities

Security failures can also originate in SDKs and protocol implementations. Incorrect state management, improper request isolation, shared caches, insufficient validation, or weak handling of authorization context can allow information to cross sessions or tenants.

Stateless deployment does not remove these risks. It changes where state and security context have to be reconstructed and validated. A request must remain associated with the correct identity, resource, permission, and security policy even when the server does not maintain a conventional conversational session.

The MCP landscape research highlights lifecycle and implementation security as distinct dimensions, reinforcing the need to examine server behavior rather than assuming that a formally specified protocol guarantees secure deployment [28].

3.4.10 Software Bill of Materials, Signing, Hashing, Version Pinning, and Reproducible Builds

SBOMs provide visibility into the components used to construct an MCP server. They can help identify vulnerable libraries, outdated dependencies, unexpected transitive packages, and changes between releases.

However, SBOMs are not automatically trustworthy. Recent empirical research on open source projects found that SBOM adoption remains limited, fewer than half of studied projects included SBOMs consistently in version control or release artifacts, and many SBOMs lacked fields such as hashes, suppliers, and vulnerability information [45].

This means that MCP registries should not simply require 'an SBOM.' They should establish minimum information requirements, validate SBOM generation, protect SBOM integrity, and associate the SBOM with a verifiable build artifact.

Signing complements SBOMs by helping establish that the artifact examined is the artifact actually executed. Hashes and version pinning further support change detection. Reproducible builds offer an additional assurance mechanism by making it possible to verify whether independently generated artifacts correspond to the expected source.

Table 9 consolidates the supply-chain and implementation evidence discussed in Section 3.4, focusing on documented attack demonstrations, ecosystem findings, and empirically observed implementation risks. It links the affected component and attack vector to severity, consequence, and available detection or mitigation measures.

Table 9. Documented MCP Supply Chain and Implementation Security Incidents and Evidence

Component	Vulnerability or attack	Attack vector	Severity	Consequence	Detection or mitigation	Citation(s)
MCP server lifecycle	Malicious or compromised server introduction	Server registration and deployment	High	Unauthorized capability exposure	Lifecycle governance and provenance verification	[28]
Remote MCP server	Malicious server attack pathways	Remote server interaction	High	Data access, unauthorized actions and other server mediated compromise	Host filtering, auditing and runtime hardening	[29]
MCP tool selection	Confused deputy through adversarial metadata	Tool description and name manipulation	High	Tool interception and malicious execution	Selection auditing and provenance controls	[30]
MCP aggregation ecosystem	Malicious servers uploaded to public aggregators	Repository or registry distribution	High	Local file access and harmful device actions	Registry auditing, user verification and sandboxing	[31]
LLM agent software	JSON injection, prompt injection and denial of service	MCP or function calling interfaces	High	Agent compromise or degraded availability	Unified software and AI security testing	[42]
Software supply-chain	Dependency based vulnerability propagation	Transitive dependencies	High	Risk propagation across client applications	Dependency inventories and continuous monitoring	[43]
Open source package ecosystem	Incomplete supply-chain visibility	Weak SCA coverage	Medium to high	Missed dependency or vulnerability detection	Software composition analysis	[44]
Open source projects	Incomplete SBOM adoption	Missing or incomplete component records	Medium	Reduced traceability and delayed response	Improved SBOM generation and release integration	[45]

Component	Vulnerability or attack	Attack vector	Severity	Consequence	Detection or mitigation	Citation(s)
Agent deployment	Tool mediated software vulnerabilities	Structured input and runtime interfaces	High	Unauthorized actions or application compromise	Combined AI and conventional vulnerability assessment	[42]

MCP, Model Context Protocol; LLM, large language model; JSON, JavaScript Object Notation; AI, artificial intelligence; SBOM, Software Bill of Materials; SCA, software composition analysis. These abbreviations describe the protocol, model-based systems, structured software interfaces, and supply-chain security controls represented in the table.

3.5 Secure MCP Architecture, Provenance, Defense in Depth, and Research Gaps

3.5.1 Secure MCP Reference Architecture

The preceding results point toward a layered reference architecture in which the LLM is treated as a decision-making component rather than as the final authority over system actions.

The architecture should contain at least seven security layers: identity and provenance; authorization and capability control; metadata inspection; isolated execution; data-flow enforcement; network and egress control; runtime monitoring and human authorization. The central principle is separation of concerns. The LLM can determine that a tool appears relevant, but an independent policy layer should determine whether that tool is trusted and authorized.

Recent reviews of LLM agent security emphasize the need for system level defenses because attacks can enter through prompts, retrieval, memory, tools, external services, and action interfaces [32,33]. The 2026 MCP literature reaches a similar conclusion, with malicious server experiments demonstrating that several attack channels can coexist within a single server interaction [29].

3.5.2 Strong Server Identity and Provenance Verification

Server identity should be established through evidence stronger than a name, URL, or registry listing. Ideally, an MCP client should be able to determine the publisher, artifact identity, version, signing status, ownership history, and relationship between the registered server and the executed artifact.

The IEEE TSE findings make this requirement particularly compelling because malicious servers were successfully placed on aggregation platforms and users had difficulty distinguishing them from legitimate servers [31].

A provenance system should therefore bind at least four elements: publisher identity → software artifact → server endpoint → registered tool metadata. Breaking this chain creates opportunities for impersonation, repository substitution, and rug pulls.

3.5.3 Cryptographic Tool and Manifest Signing

Cryptographic signatures can establish artifact integrity and publisher authenticity. A signed tool manifest can also create a stable representation of approved metadata.

This is especially important for rug pull detection. An organization can compare the current manifest against a previously approved version and require review when the signature, hash, permission set, or metadata changes.

However, signing is not equivalent to security. A compromised publisher can sign malicious software. An organization may also mistakenly trust the wrong publisher. Signing must therefore be combined with identity verification, provenance records, code scanning, and behavioral monitoring.

3.5.4 Capability Attestation

Capability attestation would allow a server to state, and preferably prove, what resources and operations it is capable of accessing. The assertion should be

independently verifiable rather than being based only on the server's own description.

This could provide an important bridge between MCP metadata and conventional access control. Instead of saying that a server 'can access files,' the server could present a verifiable capability record stating precisely which directory, operations, network destinations, or APIs are authorized.

Capability oriented research already demonstrates the feasibility of associating cryptographic permissions with agents and data fields [40]. This area deserves further development specifically for MCP.

3.5.5 Least Privilege Authorization and Fine Grained Tool Scopes

Fine grained authorization should restrict individual tool operations, resource classes, data categories, and network destinations.

A high-impact tool should require a stronger authorization condition than a low-risk read only tool. For example, reading a public webpage should not require the same policy as modifying an enterprise database or sending a financial transaction.

This graduated model also provides a basis for human authorization. A user does not need to approve every harmless operation, but high-impact operations can be routed through explicit consent mechanisms.

3.5.6 Sandboxing and Capability Restriction

Sandboxing remains an important containment layer because some attacks will inevitably bypass earlier defenses. A malicious MCP server should not automatically inherit the full permissions of the host. File systems, operating system processes, environment variables, device interfaces, and network connectivity should be restricted to the minimum required capability set.

Recent research on agent security consistently identifies execution isolation and least-privilege as important controls [32,33]. Sandboxing is therefore best viewed as a consequence limiting mechanism: it does not guarantee that an attack will fail, but it can prevent a successful attack from producing catastrophic system wide effects.

3.5.7 Secure Secret Management and Credential Isolation

Secrets should preferably remain outside the LLM context and should be injected into tool execution only when a permitted operation requires them.

A secure secret management architecture should provide short-lived tokens, narrow scopes, resource specific binding, rapid revocation, automatic rotation, audit logging, separation of development and production secrets, and protection against accidental model exposure.

The credential should be associated with the tool and policy decision, not simply handed to the model as context.

3.5.8 Static Metadata Analysis

Static analysis should examine tool names, descriptions, schemas, permissions, examples, publisher information, and dependency relationships. The objective should not be to identify malicious words alone. A stronger system would detect relationships that are unusual for the declared function. A weather tool requesting filesystem access, for instance, would warrant investigation even if its description contains no obviously malicious language. The current MCP literature indicates that semantic manipulation is a core risk [28,30]. Metadata analysis should therefore become a standard stage of MCP onboarding.

3.5.9 Decision Level Monitoring

Decision level monitoring observes the relationship between the user's objective and the tool the model chooses. This is different from ordinary logging because the aim is to determine whether the chosen action is plausible under the declared task.

For example, a user may ask for a summary of a document while the model selects an email sending tool. Such a selection should produce a security signal even if the tool invocation is syntactically valid.

This type of monitoring is increasingly important because tool selection manipulation does not always result in visibly malicious output. The confused deputy findings demonstrate that an agent can be induced to select an attacker controlled tool while appearing to perform the intended task [30].

3.5.10 Runtime MCP Security Gateways

A security gateway positioned between the LLM agent and MCP servers can provide a deterministic enforcement point. It can validate the destination server, inspect tool metadata, evaluate authorization, apply data-flow rules, enforce network restrictions, and log the result.

The 2026 runtime policy enforcement study provides direct empirical support for this architecture. Its policy enforcement point reduced the reported ASR from 40.0% to 5.0% under the controlled experimental conditions [41].

This result is encouraging because it suggests that some risks can be mitigated without expecting the LLM itself to correctly recognize every malicious instruction.

3.5.11 Network and Egress Controls

Network controls should be derived from tool capability requirements. Each server should ideally have a documented list of permitted destinations and protocols.

Unexpected outbound traffic should trigger investigation. Egress restrictions are especially important for exfiltration because they can limit the success of attacks that have already compromised the model's decision process.

The control should be complemented by DNS monitoring, proxy enforcement, domain reputation, traffic volume analysis, and sensitive data-flow monitoring.

3.5.12 Human in the Loop Authorization and Consent

Human authorization is most valuable for actions with irreversible or high-consequence effects. These may include deleting information, transferring money, sending external communications containing sensitive information, changing production infrastructure, modifying access controls, or executing privileged commands.

Human approval should not become a meaningless confirmation prompt for every action. Excessive prompts can lead to approval fatigue, causing users to approve actions without meaningful review.

Recent agent security research highlights human oversight as an important component of high-impact action control, but also emphasizes the need to balance security with usability [32,33].

3.5.13 Secure MCP Deployment for Local and Remote Servers

Local MCP servers should be isolated from the host with strict filesystem and process permissions. Remote servers should additionally be subjected to identity verification, transport security, authorization validation, tenant isolation, and endpoint monitoring.
The malicious remote server study demonstrates why remote deployments require special attention [29]. The user may not have direct knowledge of the infrastructure operating the server, making provenance and continuous monitoring more important than in a tightly controlled local environment.

Nevertheless, remote deployment is not automatically less secure. A properly isolated remote service can have stronger controls than a local server running with extensive operating system privileges.

3.5.14 Security Implications of the 28 July 2026 Stateless MCP Architecture

The stateless direction of MCP increases the importance of request level security binding. Security context should accompany each operation rather than being assumed from a previous conversational session. This means that request processing should validate: principal → resource → capability → tool → destination → expiry → policy.

Headers, routing information, authorization metadata, cached tool lists, and task identifiers should all be treated as security-relevant fields.

The challenge is particularly significant for asynchronous operations. A task authorized at one-time may complete after the user's permission has expired. Systems should therefore revalidate security context at execution time where appropriate rather than assuming that authorization is permanently inherited from task creation.

The MCP lifecycle study supports treating protocol evolution as part of security engineering rather than as a purely functional update [28].

3.5.15 Standardization and Interoperability Challenges

MCP's major strength is interoperability, but interoperability can become a security weakness when implementations interpret security requirements differently.

Different hosts may handle metadata, authorization, caching, task execution, or user confirmation differently. A server secure in one environment may therefore produce a different risk profile in another.

The comparative MCP study demonstrates this broader challenge by showing that attack outcomes depend on deployment paradigm and model configuration [42].

Security standards should therefore define not only message formats but also interoperable security semantics. A client should be able to determine what a tool is authorized to do, what credentials it may use, which resources it can reach, and which provenance assertions it satisfies.

3.5.16 Major Unresolved Research Gaps

The evidence reviewed in this section reveals several important unresolved research problems.

The first is cryptographic server identity. The current ecosystem still relies heavily on names, URLs, publisher claims, and registry placement. A broadly interoperable cryptographic identity model is needed.

The second is capability attestation. Tools need a machine verifiable way of stating what they can do and what resources they can reach.

The third is provenance standardization. The ecosystem needs consistent representations for publisher identity, software artifacts, metadata fingerprints, signatures, dependencies, versions, and update history.

The fourth is standardized malicious tool benchmarks. Existing benchmarks differ substantially in attack categories, models, clients, tool sets, and evaluation metrics. This makes results difficult to compare directly.

The fifth is explainable tool selection. A security system should ideally be able to explain why a particular tool was chosen and whether the choice was consistent with policy, provenance, and capability requirements.

The sixth is formal MCP security modeling. Most existing work remains empirical or threat modeling based. Formal models could help establish conditions under which tool selection, authorization, or data-flow are provably constrained.

The seventh is cross-server authorization. Current systems need stronger methods for controlling delegated authority when a task moves through several servers.

The eighth is real time data-flow enforcement. Recent runtime policy research provides encouraging evidence [41], but broader testing is required across more models, tools, data types, and adversarial strategies.

The ninth is secure tool registries. The 2026 IEEE TSE evidence shows that current aggregation platforms can admit malicious servers and that users can have difficulty recognizing them [31].

The tenth is standardized security metrics. Attack success alone is insufficient. Security evaluation should include benign utility, harmful action completion, latency, false positives, action availability, detection time, privilege reduction, and data exposure.

The eleventh is broader model coverage. Current evidence is still concentrated around a relatively small group of commercially important and research accessible models. The generalizability of findings across open source, proprietary, multilingual, multimodal, and smaller models remains insufficiently established.

The twelfth is secure governance of MCP extensions. Future extensions may introduce additional capabilities and new state models. Security review must occur before such extensions become widely deployed.

Table 10 synthesizes the architecture and research gaps discussed in Section 3.5 by linking each principal threat to current controls, empirical support, residual weaknesses, and corresponding research directions. The matrix emphasizes where interoperable identity, authorization, information-flow, provenance, and supply-chain mechanisms remain underdeveloped.

Table 10. Secure MCP Architecture: Threat to Control to Gap Matrix

Threat	Current defensive control	Empirical support	Residual weakness	Recommended research direction	Citation(s)
Tool poisoning	Metadata scanning and validation	Emerging and empirical	Adaptive semantic manipulation	Semantic provenance and intent aware metadata validation	[28],[30]
Tool impersonation	Registry verification and publisher checks	Limited to emerging	Weak cryptographic identity binding	Cryptographically verifiable MCP identity	[31]
Confused deputy	OAuth audience and resource validation	Strong security basis	Implementation variation	End to end authorization binding	[39]
Credential abuse	Short lived and scoped tokens	Moderate	Persistent static secret dependence	Secretless MCP architectures	[39],[40]
Cross tool exfiltration	DLP, information flow and egress controls	Emerging empirical	Covert and transformed data-flows	Taint aware semantic runtime enforcement	[41]
Rug pulls	Hashing, signing and version pinning	Practical software security basis	Key management and update governance	Signed manifests with revalidation	[45]
Supply chain attacks	SBOM, signing, scanning and dependency monitoring	Empirically supported	Registry and dependency heterogeneity	Trusted MCP registry architecture	[43],[44],[45]
Capability abuse	Sandboxing and least-privilege	Strong security principle	Performance and compatibility trade offs	Fine grained capability systems	[40]
Tool selection manipulation	Selection monitoring and metadata controls	Strong emerging MCP evidence	Model specific behavior	Explainable and policy constrained tool selection	[30]

Threat	Current defensive control	Empirical support	Residual weakness	Recommended research direction	Citation(s)
Malicious remote servers	Server verification, gateway inspection and isolation	Direct empirical support	Unknown operators and changing behavior	Continuous remote server attestation	[29]
Implementation vulnerabilities	Static and dynamic scanning	Strong software security basis	Detection and coverage gaps	MCP specific vulnerability datasets and automated testing	[42]
Information flow violations	Runtime labels and policy enforcement	Early empirical support	Semantic transformations	Formal semantic information flow models	[41],[47]
SBOM and provenance weakness	Component inventories	Empirical ecosystem evidence	Incomplete metadata	Verifiable machine readable MCP manifests	[45]
Registry compromise	Publisher verification and package scanning	Emerging	Human verification remains difficult	Trusted and auditable MCP registries	[31]
High impact autonomous actions	Human authorization	Strong security principle	Approval fatigue	Risk adaptive human-in-the-loop control	[33]

MCP, Model Context Protocol; OAuth, Open Authorization; DLP, data loss prevention; SBOM, Software Bill of Materials. The matrix distinguishes existing controls from remaining gaps and identifies research directions needed to strengthen identity, authorization, information flow, provenance, and supply-chain security.

Metadata validation reduces the probability that semantic manipulation will succeed. Strong identity reduces impersonation. Authorization limits what an authenticated component may do. Least privilege reduces the consequences of compromise. Sandboxing limits operating system exposure. Egress controls restrict exfiltration pathways. Runtime monitoring detects actions that escape earlier controls. Supply chain controls reduce the probability that malicious components enter the ecosystem.

The remaining research gap is therefore largely compositional. Individual mechanisms exist, but they are not yet integrated into a common, interoperable security architecture for MCP. This is consistent with the recent systematic literature, which notes that agent security research remains fragmented across attack types, evaluation approaches, and defense layers [32,33].

A further challenge is that security controls themselves can interfere with agent utility. A metadata scanner that blocks too many tools can make the agent ineffective. A network policy that is too restrictive can prevent legitimate integrations. A human approval mechanism that interrupts every action can create approval fatigue. A runtime policy that performs expensive analysis on every call may introduce unacceptable latency.

The strongest future designs will therefore need risk-adaptive security. Low risk operations can proceed automatically under predefined controls. Medium risk operations can receive additional automated inspection. High impact operations can require explicit human authorization.

Recent empirical work on indirect prompt injection defenses demonstrates why security and utility must be measured together. In the September 2026 study of GPT 5.4, GPT 5.4 mini, and Claude Sonnet 4.6 on AgentDojo, some defensive mechanisms reduced observed attack outcomes but also affected benign utility and available actions [34]. This provides an important methodological lesson for MCP research: a defense should not be considered successful simply because it reduces attacks if it simultaneously destroys the legitimate functionality that motivated MCP adoption.

Another major finding is that data-flow security should become an explicit architectural principle. Recent CapAgent work argues that prompt level instructions do not provide a cryptographically enforceable relationship between user intent, data object, action, output channel, and destination [40]. The runtime MCP policy study takes a related approach by placing enforcement at the tool call boundary and tracking information flow across multiple steps [41]. These findings suggest a transition from prompt-centric security to policy-centric agent security.

The supply-chain dimension requires a similarly integrated approach. SBOMs, dependency scanning, signing, package verification, and version pinning are valuable, but they address different aspects of the trust problem. The 2025 empirical work on SBOM adoption shows that the technology itself is not uniformly implemented and that many project SBOMs remain incomplete [45]. Thus, merely requiring an SBOM is insufficient. The ecosystem needs verifiable, standardized, machine readable security manifests whose content can itself be validated.

The broader evidence therefore supports a secure MCP reference architecture built around zero implicit trust. No server should be trusted solely because it is connected. No tool should be trusted solely because its description appears relevant. No token should be considered safe solely because it is valid. No package should be considered secure solely because it was downloaded from a recognized registry. No action should be considered authorized solely because the LLM selected the corresponding tool. Instead, trust should be continuously established from evidence.

The reference security sequence can therefore be summarized as: verify identity → verify provenance → inspect capability → authorize narrowly → isolate execution → control information flow → restrict egress → monitor behavior → obtain human approval for high-impact actions → revalidate after change.

This layered sequence directly addresses the central failure observed throughout the review: the unsafe propagation of trust and authority across MCP boundaries.

The evidence also suggests that future MCP security standards should distinguish more clearly between identity trust, software trust, semantic trust, and action trust. A server can be authentically operated by a known publisher while still containing a vulnerability. A tool can be benign while its description is manipulated. A model can correctly interpret the user's task while selecting a malicious server. An action can be technically possible while remaining unauthorized. Treating these different forms of trust as separate security properties would produce a more precise architecture and a more meaningful evaluation framework.

Finally, the unresolved research agenda should move beyond demonstrating that attacks are possible. The field now needs reproducible, interoperable, deployment realistic evaluations that measure how well complete MCP systems prevent, contain, detect, and recover from attacks. Benchmarks should include realistic local and remote servers, heterogeneous tool sets, malicious metadata, compromised dependencies, delegated credentials, cross-server flows, asynchronous tasks, changing tool versions, and high-impact actions.

Security evaluation should also become longitudinal. A server that is safe on day one may become unsafe after a dependency update, metadata change, credential rotation, ownership transfer, or protocol extension. Security should therefore be measured as a continuous property of the MCP lifecycle rather than as a one-time certification.

Overall, the results presented in Section 3 support a fundamental shift in the way MCP security should be conceptualized. The central security challenge is not simply that an LLM may follow a malicious instruction. The deeper issue is that MCP connects a probabilistic decision maker to powerful external capabilities through trust relationships that can be manipulated before, during, and after tool execution. Tool poisoning attacks the semantic layer. Tool impersonation attacks identity. Confused deputy attacks delegated authority. Cross tool exfiltration attacks information flow. Supply chain attacks compromise the software foundation. Implementation vulnerabilities attack the execution layer.

The appropriate response is therefore architectural rather than purely prompt-based. Secure MCP systems require verified identity, provenance aware tool registration, fine-grained authorization, least-privilege, sandboxed execution, secret isolation, data-flow controls, egress restrictions, runtime monitoring, supply-chain assurance, and human authorization for high-consequence actions.

The most important future direction is to make these controls interoperable and enforceable outside the LLM's reasoning context. The model should remain capable of planning and selecting tools, but security policy should determine what the model is actually allowed to cause. This separation between reasoning authority and execution authority provides the strongest foundation for building MCP ecosystems that remain useful while resisting increasingly sophisticated attacks.

IV. CONCLUSION

The evidence reviewed demonstrates that MCP is not inherently insecure, but protocol compliance alone cannot guarantee a secure deployment. Its security ultimately depends on the coordinated behavior of the host, client, model, MCP server, authorization infrastructure, runtime environment, registry, and downstream services. The principal threats include tool poisoning, tool selection manipulation, shadowing and impersonation, confused deputy behavior, credential abuse, cross-tool data exfiltration, supply-chain compromise, and implementation vulnerabilities.

These risks can become more severe when semantic manipulation is combined with delegated authority, excessive privileges, persistent credentials, and interconnected tools.

Secure MCP adoption should therefore follow a defense in depth approach incorporating verified identity, trusted provenance, least-privilege, strong authorization, metadata inspection, sandboxing, runtime monitoring, and controlled network egress. Supply chain protections, including secure registries, dependency assessment, component verification, and controlled updates, should form an integral part of MCP security throughout its lifecycle. High impact operations should also require appropriate human authorization to prevent autonomous decisions from producing irreversible or consequential actions. Future research should prioritize standardized security benchmarks, cryptographic server and tool identity, capability attestation, provenance aware discovery, fine-grained authorization, decision level attack detection, and cross-tool data-flow controls. Formal verification of critical MCP security properties and empirical evaluation across diverse models, clients, servers, and deployment environments are also necessary to establish more reliable security guarantees. Ultimately, MCP security should be treated as an ecosystem security problem in which conventional software vulnerabilities and LLM specific semantic attacks are addressed together rather than as isolated API security concerns.

ACKNOWLEDGEMENT

The author acknowledges the researchers, scholars, and scientific communities whose published work provided the evidence base for this comprehensive review of MCP security and its evolving threat landscape. Appreciation is also extended to the broader open research community whose advances in artificial intelligence, cybersecurity, software security, and agentic systems continue to inform the understanding of secure MCP deployment.

FUNDING STATEMENT

This comprehensive review received no specific financial support, research grant, or institutional funding from any public, private, commercial, or nonprofit organization. The work was undertaken independently using publicly available scholarly literature and other relevant sources reviewed for the purposes of evidence synthesis and analysis.

CONFLICT OF INTEREST STATEMENT

The author declares that there are no financial, professional, personal, or other interests that could have influenced the interpretation, analysis, or conclusions presented in this comprehensive review. The manuscript was prepared independently, and no external relationship or affiliation is considered to have compromised the objectivity or academic integrity of the work.

REFERENCES

[1] M. Abou Ali, F. Dornaika, and J. Charafeddine, "Agentic AI: a comprehensive survey of architectures, applications, and future directions," *Artif Intell Rev.*, vol. 59, Art. no. 11, 2026. Springer

[2] A. Milani, V. Franzoni, and E. Florindi, "Indirect prompt injection in large language models," *Neural Comput Appl.*, vol. 38, Art. no. 530, 2026. Springer

[3] B. M. Reichert and R. R. Obelheiro, "Software supply chain security: a systematic literature review," *Int J Comput Appl.*, vol. 46, no. 10, pp. 853–867, 2024. Taylor & Francis

[4] Y. Xu, Y. Zhuang, X. Liu, T. Zhang, B. Xiao, X. Xu, *et al.*, "LLM agents security duality: a comprehensive survey of self-security and empowered cybersecurity," *Artif Intell Rev.*, vol. 59, Art. no. 174, 2026. Springer

[5] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (MCP): landscape, security threats, and future research directions," *ACM Trans Softw Eng Methodol.*, 2026. ACM

[6] C. Huang, X. Huang, N. P. Tran, and A. Milani Fard, "Model Context Protocol threat modeling and analysis of vulnerabilities to prompt injection with tool poisoning," *J Cybersecur Priv.*, vol. 6, no. 3, Art. no. 84, 2026. MDPI

[7] Z. Li, J. Wu, Y. Peng, T. Luo, X. Cui, and X. Ling, "Confused deputy attack against Model Context Protocol," *ACM Trans Softw Eng Methodol.*, 2026. ACM

[8] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, *et al.*, "The rise and potential of large language model based agents: a survey," *Sci China Inf Sci.*, vol. 68, Art. no. 121101, 2025. Springer

[9] B. C. Das, M. H. Amini, and Y. Wu, "Security and privacy challenges of large language models: a survey," *ACM Comput Surv.*, vol. 57, no. 6, 2025. ACM

[10] M. A. Ferrag, A. Lakas, N. Tihanyi, and M. Debbah, "Securing LLM agents: from prompt sanitization to autonomous red teaming and beyond," *Internet Things Cyber-Phys Syst.*, vol. 5, pp. 185–209, 2026. ScienceDirect

[11] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, *et al.*, "On protecting the data privacy of Large Language Models (LLMs) and LLM agents: a literature review," *High-Confidence Comput.*, vol. 5, no. 2, Art. no. 100300, 2025. ScienceDirect

[12] J. Park, G. Kim, H. Lee, and J. Park, "Beyond tool poisoning: attack surfaces of malicious remote MCP servers across LLM platforms," *Electronics*, vol. 15, no. 10, Art. no. 2214, 2026. MDPI

[13] T. Geng, Z. Xu, Y. Qu, and W. E. Wong, "Prompt injection attacks on large language models: a survey of attack methods, root causes, and defense strategies," *Comput Mater Contin.*, vol. 87, no. 1, Art. no. 4, 2026. Tech Science Press

[14] M. Leo, F. Tan, T. Miao, and G. Anand, "From threat to trust: assessing security risks of agentic AI systems," *Int J Inf Secur.*, vol. 25, Art. no. 23, 2026. Springer

[15] X. Zhang, C. Zhang, T. Li, Y. Huang, X. Jia, M. Hu, *et al.*, "JailGuard: a universal detection framework for prompt-based attacks on LLM

systems," *ACM Trans Softw Eng Methodol.*, vol. 35, no. 1, pp. 8:1–8:40, 2026. ACM

[16] F. He, T. Zhu, D. Ye, B. Liu, W. Zhou, and P. S. Yu, "The emerged security and privacy of LLM agent: a survey with case studies," *ACM Comput Surv.*, vol. 58, no. 6, Art. no. 162, 2025. ACM

[17] G. Sato, S. Egami, Y. Tahara, A. Ohsuga, and Y. Sei, "Addressing prompt injection in large language models via in-context learning," *Comput Mater Contin.*, vol. 87, no. 2, Art. no. 99, 2026. Tech Science Press

[18] Z. Zhang, Q. Li, J. Cao, L. Liu, and J. Ni, "From AI-generated content to agentic action: security and safety threats in generative AI," *J Inf Intell.*, vol. 4, no. 4, pp. 276–297, 2026. ScienceDirect

[19] S. Du, J. Zhao, J. Shi, Z. Xie, X. Jiang, Y. Bai, *et al*., "A survey on the optimization of large language model-based agents," *ACM Comput Surv.*, vol. 58, no. 9, Art. no. 223, 2026. ACM

[20] N. Kshetri, "Transforming cybersecurity with agentic AI to combat emerging cyber threats," *Telecommun Policy*, vol. 49, no. 6, Art. no. 102976, 2025. ScienceDirect

[21] I. Adabara, B. O. Sadiq, A. N. Shuaibu, Y. I. Danjuma, and M. Venkateswarlu, "A review of agentic AI in cybersecurity: cognitive autonomy, ethical governance, and quantum-resilient defense," *F1000Res.*, vol. 14, Art. no. 843, 2025. F1000Research

[22] A. Andreoli, A. Lounis, M. Debbabi, and A. Hanna, "On the prevalence of software supply chain attacks: empirical study and investigative framework," *Forensic Sci Int Digit Investig.*, vol. 44, Art. no. 301508, 2023. ScienceDirect

[23] B. Hammi and S. Zeadally, "Software supply chain security: issues and countermeasures," *Computer*, vol. 56, no. 7, pp. 54–66, 2023. IEEE

[24] Mechri, M. A. Ferrag, and M. Debbah, "SecureQwen: leveraging LLMs for vulnerability detection in Python codebases," *Comput Secur.*, vol. 148, Art. no. 104151, 2025. ScienceDirect

[25] Edemacu and X. Wu, "Privacy preserving prompt engineering: a survey," *ACM Comput Surv.*, vol. 57, no. 10, Art. no. 255, 2025. ACM

[26] Z. Deng, Y. Guo, C. Han, W. Ma, J. Xiong, S. Wen, *et al*., "AI agents under threat: a survey of key security challenges and future pathways," *ACM Comput Surv.*, vol. 57, no. 7, Art. no. 182, 2025. ACM

[27] Hughes, Y. K. Dwivedi, T. Malik, M. Shawosh, M. A. Albashrawi, I. Jeon, *et al*., "AI agents and agentic systems: a multi-expert analysis," *J Comput Inf Syst.*, vol. 65, no. 4, pp. 489–517, 2025.

[28] H. Rebatchi, N. Moha, and T. F. A. Bissyandé, "Dependabot and security pull requests: large empirical study," *Empir Softw Eng.*, vol. 29, no. 5, 2024. Springer

[29] S. Srinivas, B. Kirk, J. Zendejas, M. Espino, M. Boskovich, A. Bari, *et al*., "AI-augmented SOC: a survey of LLMs and agents for security automation," *J Cybersecur Priv.*, vol. 5, no. 4, Art. no. 95, 2025. MDPI

[30] N. Dzhaliuk, D. Sabodashko, V. Khoma, Y. Khoma, V. Kolchenko, and M. Podpora, "Comparative evaluation of machine learning methods for protecting LLMs from prompt injection attacks," *Int J Inf Secur.*, vol. 25, Art. no. 109, 2026. Springer

[31] H. G. Moia, I. J. Sanz, G. A. F. Rebello, R. D. Meneses, B. Hitaj, and U. Lindqvist, "LLM in the middle: a systematic review of threats and mitigations to real-world LLM-based systems," *Comput Sci Rev.*, vol. 61, Art. no. 100916, 2026. ScienceDirect

[32] H. Jing, F. Li, Y. Dong, W. Zhou, and R. Liu, "Memory poisoning attacks on retrieval-augmented large language model agents via deceptive semantic reasoning," *Eng Appl Artif Intell.*, vol. 167, Art. no. 113968, 2026. ScienceDirect

[33] Y. Tang, Y. Liu, J. Lan, Z. Yan, and E. Gelenbe, "Security of LLM-based agents regarding attacks, defenses, and applications: a comprehensive survey," *Inf Fusion*, vol. 127, Art. no. 103941, 2026. ScienceDirect

[34] A. Khan, K. AlKhanbashi, and A. Mohamed, "Evaluating indirect prompt injection defenses in tool-using LLM agents: security, utility, and

replication," *Computers*, vol. 15, no. 9, Art. no. 570, 2026. MDPI

[35] J. Zhang, H. Bu, H. Wen, Y. Liu, H. Fei, R. Xi, *et al*., "When LLMs meet cybersecurity: a systematic literature review," *Cybersecurity*, vol. 8, Art. no. 55, 2025. Springer

[36] E. S. Mathew, "Enhancing security in large language models: a comprehensive review of prompt injection attacks and defenses," *J Artif Intell.*, vol. 7, no. 1, pp. 347–363, 2025. Tech Science Press

[37] R. B. Hadiprakoso, W. Wilujengning, and A. Amiruddin, "Adaptive multi-layer framework for detecting and mitigating prompt injection attacks in large language models," *J Inf Syst Eng Bus Intell.*, vol. 11, no. 3, pp. 473–487, 2025. Journal of Information Systems Engineering and Business Intelligence

[38] T. Gokcimen and B. Das, "A novel system for strengthening security in large language models against hallucination and injection attacks with effective strategies," *Alexandria Eng J.*, vol. 123, pp. 71–90, 2025. ScienceDirect

[39] Böhme, E. Bodden, T. Bultan, C. Cadar, Y. Liu, and G. Scanniello, "Software security analysis in 2030 and beyond: a research roadmap," *ACM Trans Softw Eng Methodol.*, vol. 34, 2025. ACM

[40] H. Hou, Y. Ma, and J. Miao, "CapAgent: semantic data-flow governance for LLM agents in big-data cognitive computing," *Big Data Cogn Comput.*, vol. 10, no. 9, Art. no. 293, 2026. MDPI

[41] S. Wang, S. Zhu, and R. Li, "Runtime policy enforcement for MCP-based LLM agents," *Electronics*, vol. 15, no. 13, Art. no. 2829, 2026. MDPI

[42] Nageshwaran and S. Ezekiel, "Agentic AI and large language models for autonomous IoT cybersecurity: a systematic survey, taxonomy, and research roadmap," *Electronics*, vol. 15, no. 12, Art. no. 2740, 2026. MDPI

[43] Y. Shen, X. Gao, H. Sun, and Y. Guo, "Understanding vulnerabilities in software supply chains," *Empir Softw Eng.*, vol. 30, Art. no. 20, 2025. Springer

[44] Shu, W. Chen, G. Fan, H. Yu, Z. Huang, and Y. Liang, "Tool or toy: are SCA tools ready for challenging scenarios," *Comput Secur.*, vol. 158, Art. no. 104624, 2025. ScienceDirect

[45] S. Nocera, S. Romano, M. Di Penta, R. Francese, and G. Scanniello, "On the adoption of software bill of materials in open-source software projects," *J Syst Softw.*, vol. 230, Art. no. 112540, 2025. ScienceDirect

[46] F. M. Teichmann, "Agentic AI systems as a responsibility-attribution problem in autonomous cyber operations," *Law Innov Technol.*, 2026. Taylor & Francis

[47] W. B. Mbaka and K. Tuma, "Less is more: usefulness of data flow diagrams and large language models for security threat validation," *Empir Softw Eng.*, vol. 31, Art. no. 122, 2026. Springer