计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(5): 1177−1190, 2026

针对大语言模型 MCP 服务器中权限使用的实证研究

张 泉[1] 冯诀宵[2] 周炽金[1] 姜 宇[3]
[1]（上海市高可信计算重点实验室（华东师范大学） 上海 200062）
[2]（中国工业互联网研究院 北京 100015）
[3]（清华大学软件学院 北京 100084）
（quanzhang@sei.ecnu.edu.cn）

An Empirical Study of Privilege Usage in Large Language Model MCP Servers

Zhang Quan[1], Feng Juexiao[2], Zhou Chijin[1], and Jiang Yu[3]
[1]（*Shanghai Key Laboratory of Trustworthy Computing* (*East China Normal University*), *Shanghai* 200062）
[2]（*China Academy of Industrial Internet*, *Beijing* 100015）
[3]（*School of Software*, *Tsinghua University*, *Beijing* 100084）

Abstract The evolution of large language models (LLMs) and Model Context Protocol (MCP) has facilitated the development of various MCP servers and tools that extend LLM capabilities, allowing LLMs to interact with external services and content. However, these servers also introduce new security threats. Specifically, they are granted privileges to automate tasks such as email processing and cloud infrastructure management. Due to the inherent instability of LLM outputs and their susceptibility to manipulation, attackers can exploit LLMs to illicitly invoke these tools, causing severe damage. Therefore, the analysis and control of privileges for LLM servers are of paramount importance. This research designs an automated analysis framework to analyze MCP servers and their associated tools. Through static analysis, the framework identifies the privileged APIs called by these tools and their invocation methods. It then categorizes these privileges based on their sensitivity and corresponding behaviors to investigate the current state of privilege usage. We collect 1 609 MCP servers and conduct a detailed analysis of 200, revealing that each server requests an average of 40.74 privileged APIs. Furthermore, a significant 52% of these privilege validations rely on API Tokens and passwords, which fail to adhere to the principle of least privilege. This study calls for developers to implement stricter privilege controls for MCP servers and actively explore more secure authorization mechanisms.

Key words large language model; Model Context Protocol; LLM tools; privilege analysis; static analysis

摘 要 当前大语言模型（LLM）允许用户通过模型上下文协议（Model Context Protocol, MCP）调用 MCP 服务器中的多种工具，进而与外部世界进行交互。然而，这些 MCP 服务器带来了新的安全隐患。具体而言，为了自动化完成诸如数据库查询和云服务设施管理等任务，这些工具被赋予了特定的权限。由于 LLM 安全性不足，攻击者可以利用 LLM 非法调用这些服务器所暴露的工具，对用户资产造成损害。因此，对 MCP 服务器和工具进行权限分析和控制至关重要。设计了一个自动化分析框架，旨在对 MCP 服务器及工具进行深入分析。该框架通过静态分析，可以确定工具调用的权限接口及其调用方法，并根据权限的敏感程度和行为进行分类，从而探究当前工具的权限使用现状。收集了 1 609 个 MCP 服务器，

收稿日期：2025-12-05；**修回日期**：2026-03-04
基金项目：CCF-华为胡杨林基金资助（CCF-HuaweiTC202510）
This work was supported by CCF-Huawei Populus Grove Fund (CCF-HuaweiTC202510).
通信作者：周炽金（tlock.chijin@gmail.com）

并对 200 个 MCP 服务器进行了细致分析,发现平均每个 MCP 服务器调用 40.74 个权限接口。此外,高达 52% 的权限验证依赖于 API Token 和账号密码,未能遵循最小权限原则。旨在呼吁开发者严格控制 MCP 服务器的权限,并积极探索更加安全的权限控制机制。

关键词 大语言模型;模型上下文协议;大模型工具;权限分析;静态分析

中图法分类号 TP311.13; TP309

DOI: 10.7544/issn1000-1239.202550874 **CSTR:** 32373.14.issn1000-1239.202550874

大语言模型(large language model, LLM)目前在自然语言理解、复杂推理和内容创作等方面取得了巨大突破[1-2]。然而,仅凭其固有的静态知识库,LLM 无法获取实时信息或执行外部操作[3]。因此,为了将这些强大的能力应用于现实世界,LLM 必须借助外部工具来打破这一局限,实现与动态、多变的外部世界的交互,例如进行实时天气查询、数据库操作或云服务调用等。

在这一背景下,模型上下文协议(Model Context Protocol , MCP) 被提出。其作为一种标准化的交互协议,使得大模型能够与外部数据、提示命令以及各种工具进行交互[4-5]。这一协议的出现显著地简化了新工具的开发、接入与调用过程,使得开发者可以快速、便捷地为大模型集成各类特定功能。具体来说,可以构建一个 MCP 服务器(MCP server),在服务器中提供相应的 MCP 工具或内容。只要用户将该 MCP 服务器注册到其 Host 应用客户端,则客户端能够根据协议规范,自动获取 MCP 服务器提供的工具和内容,由大模型根据需求进行决策和使用。

这种无缝的集成范式一方面简化了 MCP 工具的开发难度,同时还极大便利了用户接入大模型工具。这一范式推动了 AI 智能体生态的蓬勃发展,促使开源社区与各大企业积极协作,共同构建了庞大的 MCP 服务器社区,其服务范围涵盖了从日常生活辅助到企业级复杂工作流的各类场景。例如,在项目自动化管理领域,GitHub 发布了其官方 MCP 服务器,允许大模型作为智能代理,自动执行诸如创建拉取请求(pull request)或进行代码库安全检查等任务[6]。同时,在云服务设施管理方面,亚马逊公司的 AWS 和微软公司的 Azure 等主流云服务提供商也推出了官方 MCP 服务器,为企业提供了通过自然语言自动化管理计算资源、查看总结资源看板等服务[7]。这标志着 LLM 正在从被动的信息处理工具,转变为能够主动、自主地在真实世界中执行复杂任务的智能代理。

然而,这些 MCP 服务器和工具在带来便利的同时,也引入了全新的安全挑战。MCP 工具作为大模型与外部世界的执行接口,为了完成自动化任务,用户必须赋予其一定的权限。然而,大模型本身固有的“黑箱”特性使其可解释性不足,难以追踪其决策过程。更严重的是,LLM 的输出容易受到恶意输入的攻击,如提示注入(prompt injection)。攻击者可以精心构造恶意输入,劫持 LLM 并恶意调用被赋予高权限的 MCP 工具,对用户的关键设施和资产实施攻击,其后果可能包括未授权的数据访问、关键配置的恶意篡改,乃至严重的财产损失。

在此类攻击过程中,权限的滥用是造成损害的核心原因。因此,为了深入理解这一新型攻击范式并构建有效的防御机制,对 MCP 工具的权限使用进行系统性分析至关重要。这不仅能揭示其攻击面,更能为后续的防护措施提供基础性理解。为此,本文构建了一个自动化的分析系统,旨在对开源社区中的 MCP 服务器和其集成的 MCP 工具进行大规模且深入的探究。本研究围绕 4 个核心问题展开,旨在全面评估当前 MCP 生态系统的权限使用安全状况。

1)当前 MCP 服务器和工具应用到了哪些场景,其中包含哪些安全敏感的场景?该研究首先分析了 MCP 服务器在现实世界中的关键应用场景,以深入理解潜在的权限滥用所能造成的危害。研究发现,这些服务器提供了大量工具,广泛应用于多个高风险领域,包括数据自动访问、开发者工具、应用控制和任务自动化等。这些应用场景直接涉及敏感数据和核心系统操作,一旦权限被泄露或滥用,将会造成数据破坏、资产损失或服务中断等严重后果。这一发现揭示了当前 MCP 生态系统所面临的广阔且极易被利用的攻击面。

2)当前开源 MCP 服务器的功能复杂度如何,使用了多少权限?为了评估 MCP 工具存在的权限冗余问题,本研究通过分析其代码实现,探究了其代码规模与功能复杂度,并统计了所使用的权限数量。分析结果表明,这些 MCP 服务器平均集成了 14.38 个工具,平均代码量达到 8 490 行,并且使用了超过 40.74 个权限。这些数据表明,MCP 工具的内部逻辑高度

复杂，使用了大量权限，这导致权限高度集中，使得攻击者可轻松通过组合其中权限完成攻击，存在严重安全隐患。

3）MCP 工具使用了哪些类别权限，是否存在对敏感权限的使用？该研究问题旨在探究 MCP 工具所使用的权限类型，并着重分析了其中需要严格控制的敏感权限。通过对每个权限及其功能描述进行逐一审查，发现大量 MCP 工具使用了安全攸关的权限，包括任意命令执行、云服务管理、远程资源修改和应用控制等。这些高危权限的使用表明，当前 MCP 工具的执行能力已远超简单的数据查询，它们能够直接对用户的关键设施和资产进行修改或控制，使得攻击行为的潜在危害性成倍增加。

4）MCP 工具在调用权限时是否进行了恰当的授权，其授权机制如何？该研究问题对 MCP 工具调用权限时的授权方法进行了深入分析，旨在分析其是否能够进行精细化的权限控制。结果表明，大量 MCP 工具依赖于粗粒度的 API token 或用户账号密码进行权限管理。这种授权机制普遍存在权限冗余的风险，严重违反了最小权限原则，使得攻击者一旦获取 Token 或密码，便可能拥有远超其所需任务的权限，进而实施更广泛的攻击。这一发现凸显了当前 MCP 生态在权限管理上的粗放与 Token 泄露的严重风险。

通过上述 4 个研究问题，本研究实现了从 MCP 服务器应用场景到其内部工具权限细节的层层递进分析。其中，前 2 个问题主要分析 MCP 服务器的整体功能与当前社区特点，后 2 个问题则聚焦于具体 MCP 工具的权限使用与授权机制。本研究旨在全面评估当前 MCP 生态在权限管理方面的不足，探讨更为安全合理的开发范式，并为后续防护技术的研究提供启示。

研究发现，当前 MCP 生态在权限管理方面存在着系统性的安全缺陷。高风险的应用场景、复杂的内部逻辑、对敏感权限的滥用以及粗放的授权机制共同构成了一个广阔且脆弱的攻击面。在面对提示注入等针对 LLM 的攻击时，这些工具极易被恶意行为者劫持，其冗余的权限将成为攻击者的工具，对用户的数字资产和关键基础设施构成直接且严重的威胁。同时，上述研究也有助于社区开展 MCP 工具漏洞检测分析和攻击防护的技术开发，共同构建安全可靠的大模型智能体系统。

总的来说，本文的贡献有 3 个方面：

1）对 LLM 的 MCP 工具权限使用的安全问题进行了系统的实证研究，并提出了 4 个关键研究问题。

2）搭建了一套自动化的智能体权限分析系统，针对每个 MCP 服务器及其内部集成的工具进行调用图分析和依赖分析，定位权限使用和关键权限。

3）通过对 4 个研究问题的分析，揭示了当前 MCP 生态中的权限风险，为后续检测防护工具的构建和新型授权机制的设计提供了实践依据。

1 相关工作

随着 LLM 在现实世界系统中的深度集成，一个由外部工具、插件和智能体组成的庞大生态系统迅速形成，它们通常通过模型上下文协议等标准化接口与模型进行交互[8-9]。这种扩展虽然显著地增强了 LLM 的功能，使模型功能逐渐从封闭式推理转变为具有外部作用的交互式执行，但其也引入了一个复杂且新颖的攻击面。现有的大量综述性研究从整体层面对 LLM 所面临的安全与隐私挑战进行了系统梳理与归纳[10-14]。总体而言，现有的攻击关注于 2 个方面，一个是模型自身的安全问题[15-16]，另一个则是大模型周边生态中的安全问题[11-12,17]。

1) 模型安全。对模型自身安全的研究揭示了 LLM 固有的脆弱性，这些脆弱性解释了为何在没有严格外部控制的情况下，赋予 LLM 高权限本质上是危险的。现有研究主要集中在安全性对齐、提示注入、越狱攻击以及数据污染等方面。其中，使用基于人类反馈的强化学习（RLHF）是当前主流的安全性对齐技术之一[18]。为了主动发现并修复对齐失效的案例，工业界和学术界广泛采用了“红队测试”（red teaming）的方法，通过模拟对抗性攻击来系统性地探测模型的安全边界[19-20]。

尽管经过严格的安全对齐，大模型仍面临提示注入、越狱攻击等问题。研究者们通过搭建系统性的框架来形式化定义提示注入攻击，并对现有的多种攻击和防御技术进行了全面的评估，结果表明该漏洞在主流模型中普遍存在[20-21]。Zhan 等人则对当前的防御技术进行了分析，结果表明现有研究普遍缺乏对防御有效性的严格评估，许多声称有效的防御措施在面对自适应攻击时表现不佳[22]。对于越狱攻击，当前也有大量工作构建数据集和测试框架来评估已有的攻防技术[17,23-24]。目前的共识是，单一的防御措施难以应对所有类型的越狱攻击，构建多层次、深度的防御体系是未来发展的必然方向。数据投毒方面，模型层面主要关注训练过程中的后门植入问题[25-26]。

2) 生态安全。随着 LLM 大范围应用，其安全边

界已不再局限于模型本身,而是扩展到了其所在的整个应用生态。研究的焦点也随之转移到LLM作为智能体与外部世界交互时所面临的风险[11,27]。首先,大模型在处理互联网上丰富的信息时,持续面临着恶意输入注入的问题。研究者在真实场景中,对现有的大模型应用进行实验验证,结果表明,目前的大模型极易无意间从互联网上获取恶意内容[25,27-28]。

近期,研究者开始系统性地分析智能体独有的安全脆弱性。Li等人[29]提出了一个智能体安全弱点的分类框架,并演示了一种通用的攻击链路。Kong等人[12]对LLM驱动的智能体通信进行了全面调研。为了解决智能体安全评估缺乏统一标准的问题,RAS-Eval等基准测试平台被提出,提供了一个支持在动态环境中执行真实世界工具的测试框架,覆盖了11类常见漏洞,以量化评估智能体的安全性[11]。

虽然已有大量工作从模型和应用生态2个层面,对大模型的安全问题进行了广泛地研究与实证分析。然而,这些研究仍主要关注于攻防本身,如针对攻击方案、模型安全性和防护技术进行实证研究[30-31]。它们并未深入分析决定攻击潜在破坏程度的根本性因素,即授予智能体所使用工具的权限;而本文则更多关注于权限分析,深入分析MCP工具所能行使的权限和权限的控制机制,进而探讨在无法保证模型自身绝对可靠的前提下,当前的MCP工具能否保证安全,并启发新型防护和授权机制的开发。

2 预备知识

2.1 模型上下文协议

随着LLM能力的飞速发展,其应用场景已不再局限于简单的文本预测和问答。为了在现实世界中执行复杂、自主的任务,LLM迫切需要与外部的工具、数据库和应用服务接口(application programming interface, API)进行交互,以获取实时信息并执行具体操作[5]。然而,在标准化协议出现之前,将模型与每一个外部工具进行连接都需要进行定制化的开发,效率低下、成本高昂,形成一个难以维护和扩展的碎片化生态系统。这一集成困境严重制约了LLM智能体的发展。

为了应对这一挑战,人工智能公司Anthropic开发并开源了MCP[32]。MCP引入了一个通用的、标准化的中间层,其作用类似于硬件领域的USB接口或Web领域的HTTP协议[4]。该协议发布后,迅速获得了业界的广泛采纳,包括OpenAI和Google在内的主要行业参与者均已支持和兼容该标准,标志着MCP正成为事实上的行业标准。

MCP采用一种客户端-服务器模型,其架构由3个不同的组件构成,包括主机(host)、客户端(client)与服务器(server)[33]。主机是用户与AI进行交互的应用程序,负责管理整个对话的生命周期和执行安全策略。客户端存在于主机内部,与单个MCP服务器维持一个专用的一对一连接,充当会话管理器。服务器则是独立运行于远端或本地的一个程序,向AI模型暴露其能力。

服务器通过3种核心原语来暴露其功能,提供只读访问的资源(resources)、允许执行操作的工具(tools)以及用于指导交互的提示(prompts)[32]。工具的使用需要明确的用户同意,这提供了一种“人在环路(human-in-the-loop)”的安全机制。整个通信骨干基于JSON-RPC 2.0协议,以促进结构化的双向通信。这种主机—客户端—服务器的架构是一种注重安全的架构设计,通过隔离机制防止单个受损的工具影响其他工具。

大模型与MCP服务器的交互过程是一个由主机和客户端协调的结构化流程,如图1所示。首先,需要用户将MCP目标服务器的地址注册到其使用的Host应用客户端上。随后,用户的Host会根据MCP规定,开始进行初始化阶段①。在此阶段,服务器会

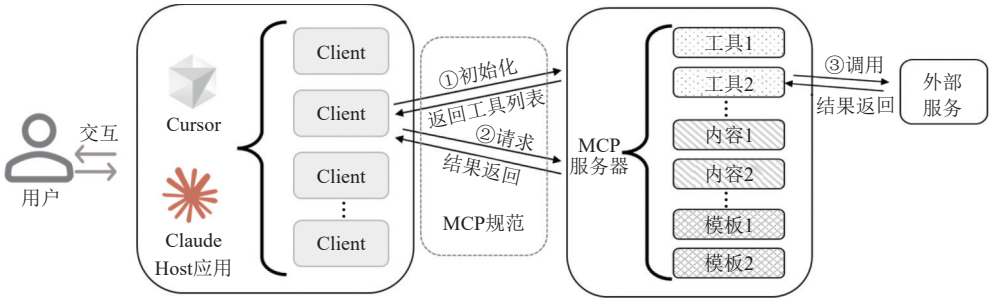


Fig. 1 Illustration of interaction with the MCP

图1 模型上下文协议交互示意图

向 Host 提供其所有的功能，包括它所暴露的资源和工具列表，并提供功能描述或元数据。对于 MCP 工具，服务器会提供工具的详细使用方法，包括调用工具需要传入的参数。完成初始化后，进入阶段②，Host 接收到用户请求后，大模型会根据用户需求和每个工具的描述，决策是否需要调用对应的工具，并生成调用参数。随后 Host 会将其转为标准的工具调用请求，并发给 MCP 服务器。此时，进入阶段③，服务器执行工具，并调用外部服务或与外部环境进行交互，最后会将结果返回给 Host，以供其中的大模型进行回复或对后续行为进行决策。

基于这个交互流程，工具开发者只需要根据协议提供工具和相关功能参数描述，就可以接入任何大模型应用，大幅促进了智能体生态的繁荣。因此，开发者和厂商快速推出了多种多样的 MCP 服务器，提供丰富工具与现实世界进行交互。但在大模型自身安全性不足、极易被攻击者劫持的情况下，种类繁多的 MCP 服务器也造成模型可调用过多的功能，一旦被攻破，将造成更大的危害。

2.2 权限控制

权限控制一直是保障软件系统安全的基础防护手段之一。其中，最基础的原则是最小权限原则（principle of least privilege, PoLP）。PoLP 要求限制用户或进程的访问权限，使其仅拥有完成其指定功能所必需的最低权限[34]。通过对某一软件或实体限制权限，可以保证即使目标对象被攻破和利用，攻击者也仅能进行有限的恶意行为。因此，对于 MCP 服务器和工具来说，每个工具都应当遵循最小权限原则，确保即使 1 个工具被恶意利用，其造成的影响也无法扩散，降低受害损失。

在已有的各类大型软件系统中，鉴于多用户与多实体交互的复杂性，此类系统通常配备了完善的权限管理机制，以保障软件系统安全。如在 Linux 系统上，研究者们提出了基于角色的访问控制和强制访问控制等权限控制模型进行高效灵活的访问[35-36]；在云服务系统中，AWS 提出了 Cedar 语言，旨在支持用户对其服务进行权限控制[37]。与现有成熟软件系统中完善细致的权限控制框架不同，目前的智能体系统缺乏统一有效的权限控制框架。在当前用户可能灵活注册不同的 MCP 服务器的情况下，大量 MCP 服务器和工具都没有进行有效的授权和权限控制，存在严重的安全隐患。

3 实证研究流程

本研究的整体技术流程如图 2 所示，其包含 3 个核心步骤。首先，在 GitHub 等代码托管平台系统性地收集了开源的 MCP 服务器实现，并整理了其功能描述与代码规模等关键信息。其次，为基于 Python 和 TypeScript 语言开发的 MCP 服务器设计了定制化的静态程序分析流程，旨在精确提取每个 MCP 工具的调用入口，并构建其对应的函数调用图。最后，对生成的调用图进行深度分析，以识别所有外部 API 调用，并借助 LLM 的语义分析能力，筛选出其中与权限相关的敏感操作。

3.1 MCP 服务器收集

为收集多样的 MCP 服务器，本研究采用了 2 种互补的收集策略。一方面，本研究系统性地梳理了 GitHub 等平台上的相关项目汇总仓库，分析其整理的开源 MCP 服务器类别，追溯其来源以定位高质量的 MCP 服务器。另一方面，利用“MCP server”等关键词进行自动化检索，搜索 Star 数量超过 20 的项目，并结合 LLM 对搜索结果进行语义过滤与甄别，筛选掉大量虽与 MCP 服务器相关但实际并非 MCP 服务器的项目，如 MCP 服务器搭建教程、MCP 服务器汇总仓库。为了分析开发成熟度高且可能得到真实应用的 MCP 服务器，本研究在上述 2 步中都只关注获得 Star 超过 20 的 MCP 服务器项目。

通过上述方法共收集到 1 609 个开源 MCP 服务器。其中，通过项目汇总仓库获取了 549 个 Star 数超

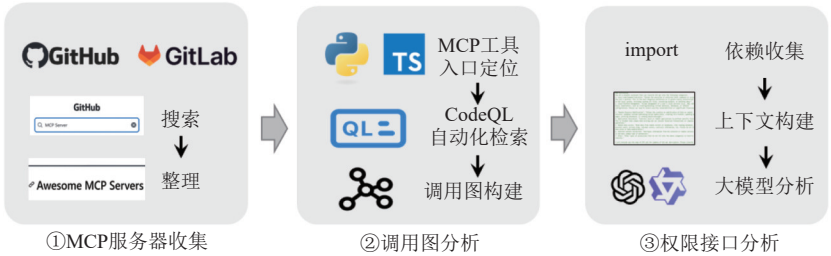


Fig. 2 Flow chart of the empirical study

图 2 实证研究流程图

过 20 的项目；GitHub 搜索阶段共检索到 1 545 个由 Python 或 TypeScript 编写以及名称中包含关键词“MCP Server”的项目。经过筛选无关仓库并对结果进行合并去重后，最终得到 1 609 个有效的待分析 MCP 服务器。

在获得 MCP 服务器的基本信息后，进一步对它们进行初步的功能分析。首先，通过确定其开发语言以制定后续静态分析流程。另一方面，收集其说明文档，对它们的预期用途、功能复杂度进行了初步的评估与分类。

3.2 调用图分析

此步骤的核心任务是对每个 MCP 服务器进行深入的静态程序分析，旨在遍历其执行路径并识别所有对外部 API 的调用，这是后续权限分析的基础。鉴于 Python 和 TypeScript 在 MCP 生态系统中的主导地位，两者开发的 MCP 服务器占比超过 70%，本研究聚焦于为这 2 种动态类型语言设计专门的分析方案。动态语言固有的灵活性（如高阶函数、动态派发等）给传统的控制流分析带来了巨大挑战。为此，本研究开发了一套定制化的分析框架，以有效构建每个 MCP 工具的控制流图并精确识别其外部 API 调用。

具体而言，分析框架的首要任务是定位每个 MCP 服务器项目中的工具入口函数，以此作为调用图构建的起点。现代 MCP 框架通常为开发者提供标准化的库以简化工具创建。本研究的静态分析工具正是利用了这些标准化的编程范式来定位入口。

对于 Python 编写的 MCP 工具，其使用的库通常要求开发者使用*@MCP.tool*() 装饰器（decorator）来标注 MCP 工具的入口函数，该装饰器可作为工具入口的定位标志，高效地识别所有工具的入口函数。

对于 TypeScript 项目，其 MCP 工具的开发则更为灵活和多变，给入口定位带来一定困难。首先，开发者可以直接通过提供的 *registerTool*() 方法，将函数对象传入作为注册的工具入口，此时可以直接通过该函数的调用进行定位。然而，由于 TypeScript 语言的灵活性，部分开发者会使用 *setRequestHandler*() 直接重载 MCP 服务器的工具调用响应方法，将其重写为定制化的路由函数，并在路由函数中直接进行工具的调用。经过对多种实现范式的细致归纳，静态分析框架成功实现了对这些复杂模式的准确定位。

在精确定位所有工具入口之后，可以以其为起点递归地构建函数调用图。为了应对动态语言中因类型不确定性而导致的调用目标模糊问题，本研究集成了 CodeQL 分析引擎，利用其强大的依赖分析与数据流分析能力，最大程度地解析变量类型与函数引用，从而构建出尽可能完整和精确的调用图，以识别所有的外部 API 调用。

3.3 权限接口分析

在构建了函数调用图之后，可以识别出每个 MCP 工具所依赖的大量外部 API。然而，这些 API 调用中仅有一部分与敏感权限相关。因此，该步骤的核心任务之一是从海量的 API 调用中精确筛选出涉及权限操作的接口，并对其进行深入分析。然而，当前 MCP 服务器与工具的功能场景多样，导致其调用的权限接口也十分繁杂，例如，文件系统访问、网络通信、远程服务调用等。这种多样性为传统的、基于预定义规则或手动建模的权限分析方法带来了巨大挑战。

为应对此挑战，本研究提出了一种基于 LLM 的自动化权限调用识别方法。该方法旨在利用大模型强大的代码理解与推理能力，结合 API 调用的具体上下文，判断其功能及可能涉及的系统权限。具体实现上，针对调用图中的每一个外部 API 调用，该分析框架执行以下步骤。首先，自动提取该 API 调用的局部代码上下文以及该源文件中所有导入（import）的外部模块列表。随后，将提取出的代码上下文、外部模块信息以及目标 API 调用本身，整合成一个结构化的提示词，并将其输入给 LLM。同时，该步骤利用了上下文学习（in-context learning）策略。具体而言，通过手动分析，向模型提供若干精心设计的示例，以提升模型判断的准确性和一致性。

通过上述自动化流程，能够高效地从外部 API 调用中筛选出一个与权限高度相关的 API 集合，并且对它们进行初步的分类与统计。然而，LLM 的训练数据可能无法覆盖部分新兴或罕见的 API。因此，为确保最终分析结果的准确性与可靠性，本研究仍进行了人工审计，对自动化分析结果进行逐一复核。对于模型可能误判或无法识别的 API 调用，我们会辅以在线官方文档和开发者社区资源的交叉验证，以纠正错误并补全缺失信息。

4 实证研究

4.1 研究问题

基于上述研究流程，本文围绕 MCP 工具权限使用中的安全问题开展系统性的实证研究。研究从 4 个层面展开，由宏观到细节、由权限使用行为到授权机制逐步深入，对 MCP 服务器中的权限使用模式进

行全面分析。首先，从MCP服务器整体出发，在研究问题1和研究问题2中梳理其安全攸关场景与内部功能复杂度，从宏观层面识别潜在安全风险。其次，在研究问题3和研究问题4对各工具的具体实现进行细粒度分析，考察其权限调用方式与实现细节。在此基础上，研究问题3从权限使用的场景与类型入手，随后研究问题4进一步系统分析权限调用的授权机制，逐步揭示其中可能引发的安全问题。

1）当前MCP服务器和工具应用到了哪些场景，其中包含哪些安全敏感的场景？

该研究问题旨在系统梳理当前智能体生态中MCP服务器及其工具的具体使用场景，特别关注其中具有潜在高风险的安全敏感场景。本研究将安全敏感场景界定为攻击者通过恶意攻击，可直接操纵或破坏用户关键资产的应用情境，其典型例子包括任意命令执行、核心云服务资源管理以及涉及隐私数据访问的场景。通过对MCP服务器应用范围的宏观分析，本研究得以分析MCP在关键领域的使用广泛程度，以及其中攻击者可能利用的攻击面，从而为后续关于权限使用与管理问题的深入研究奠定基础。

2）当前开源MCP服务器的功能复杂度如何，使用了多少权限？

考虑到最小权限原则，一个安全的MCP服务器应具备适度的功能规模与合理的复杂度，其运行所需权限也应尽可能得到控制。本研究通过该问题分析当前开发者在构建MCP服务器时的功能组织方式，考察不同功能是否被合理拆分。该研究的结果能有效反映MCP服务器在权限使用方面可能存在的冗余问题，也有助于揭示开发者在权限边界设定方面的设计实践，为社区提供有关MCP服务器功能设计与权限拆分的启示。

3）MCP工具使用了哪些类别权限，是否存在对敏感权限的使用？

本研究问题针对MCP工具的权限调用行为展开细粒度分析，识别每类权限的具体功能并据此进行分类，进而定位其中具有潜在风险的敏感权限调用。本研究将敏感权限定义为对外部系统的调用以及对本地环境具有破坏能力的操作。其包括外部网络服务API的调用，如数据库访问、云服务资产管理等，以及本地主机中的关键操作，如命令执行、本地文件系统读写等。通过对这些敏感权限调用的系统化分析，本研究能够揭示MCP工具在当前已经持有的权限范围，并进一步揭示其权限使用造成的潜在安全隐患。

4）MCP工具在调用权限时是否进行了恰当的授权，其授权机制如何？

该研究问题直接面向权限授权机制进行分析，通过分析当前常用的MCP授权方式，识别其在设计与实现层面可能存在的风险点，从机制层面揭示当前MCP服务器在权限管理中所面临的挑战。该研究关注权限调用中的授权方式，并分析相关授权方式能否进行严格和细粒度的控制约束，并评估其是否适应MCP服务器的应用场景。通过对授权机制的系统性检视，本研究期望启发社区设计更加完善、契合MCP生态需求的授权机制，从根本上提升MCP权限使用的安全性。

4.2 应用场景分析

针对该问题，本研究收集了1 609个MCP服务器，并针对其进行分类。如表1所示，可将这些MCP服务器根据功能分为5类，包括数据访问、开发者工具、应用控制、任务自动化。这些服务器覆盖了人们和外部世界交互的各个领域，功能多样。通过初步分类后，仍有部分MCP服务器难以归入合适类别，只能将它们归类为“其他”类别。

Table 1 Categorization of MCP Servers

表1 MCP服务器分类

类型	描述	数量
数据访问	查询数据库、文件、API中的数据	788
开发者工具	编码、测试、部署及管理云服务	430
应用控制	自动化浏览器、控制应用软件	172
任务自动化	日程管理、文档协作、邮件处理	132
其他	中间件、游戏、翻译等专业工具	87
总计		1 609

在这些类型中，超过48.97%的MCP服务器与数据获取相关，包括从数据库查询数据、从文件读取数据，或者调用远程API获取特定数据，这是为了给大模型推理提供充分的上下文，提供领域特定的知识。在这些类型中，一方面，这些读取都涉及数据获取的权限，若未妥善管理这些权限，则存在隐私泄露的风险。此外这些MCP服务器还可能成为攻击者进行攻击的入口，如通过MCP工具从搜索引擎或社交媒体获取数据，这些数据可能被攻击者注入恶意内容，导致命令劫持并影响后续MCP应用的使用。

MCP服务器的第二大类型是开发者工具，其主要是协助软件开发者进行开发和运维，包括软件编码、测试、部署等；此外，在云服务场景中，其还被广泛用于云服务资源的管理，包括节点的管理和监控、

服务启停等。这些功能都涉及命令执行、云服务管理等极为敏感的权限，若被滥用，可以直接造成严重的经济财产损失。

此外，MCP 服务器还广泛用于应用控制和任务自动化，在这些场景中，用户可能依赖大模型处理其文档、邮件等，并借助其进行日程管理。这些场景下虽然没有明确的权限行为，但接触了大量用户的隐私数据，若配合其他带有网络发送权限的 MCP 工具，极有可能导致隐私泄露。

总的来说，通过上述研究，可以发现 MCP 服务器生态发展十分迅速，开发者们在社区贡献了各种各样的 MCP 服务器，将与任何服务、资源、数据的交互过程进行自动化。这给用户带来了极大的便利，但是，在权限未能得到合理管控的情况下，轻易将部分权限委托给大模型，使其凭借自身能力进行使用，极有可能被攻击者利用，造成严重危害。

研究发现 1。当前 MCP 服务器在不同场景快速扩张，覆盖数据访问、开发者工具、应用控制和任务自动化等关键场景，一旦 MCP 服务器中相关功能被滥用，将导致关键资产损坏和隐私泄露等风险。

4.3 复杂度分析

在研究问题 2 中，对 MCP 服务器进行了深入分析，对其具体实现复杂度和权限使用进行进一步分析。由于大模型存在分析精度问题，部分权限接口分析需要一定的人工参与，故此处仅能对有限的 MCP 服务器进行了深入分析。具体来说，通过随机采样，从第 1 步收集的 1 609 个 MCP 服务器中抽取了 200 个进行进一步的分析。此处进行随机采样主要目的是针对不同开发完善程度的 MCP 服务器进行分析。在所有 MCP 服务器中，获得 Star 较多的往往是互联网企业官方的 MCP 服务器或有大量开源贡献者进行代码编写和问题汇报的 MCP 服务器，大多都进行了完善的开发和维护；与之相对，获得 Star 较少的则往往是个人维护、开发不够完善的 MCP 服务器。因此，本研究使用随机抽样的方法，覆盖不同类型的服务器。此外，本研究选取代码规模与权限使用数量作为评估 MCP 服务器复杂度的关键指标，二者分别表征了系统的实现复杂度与功能复杂度。

1)代码量分析。首先，图 3 展示了不同项目代码行数的分布。总的来说，200 个开源 MCP 服务器的平均代码行数为 8 490，最复杂的服务器是亚马逊 AWS 官方提供给用户的 MCP 服务器，用于辅助用户监控和管理各类云服务资源，内部集成超过 400 个 MCP 工具，总代码行数超过 40 万。这一方面表明各个主流厂商都在积极促进 MCP 生态，为用户提供多样 MCP 工具。另一方面也表明对于复杂的任务管理，往往也需要功能繁多复杂的 MCP 服务器，而复杂的服务器也往往带来潜在的安全隐患。

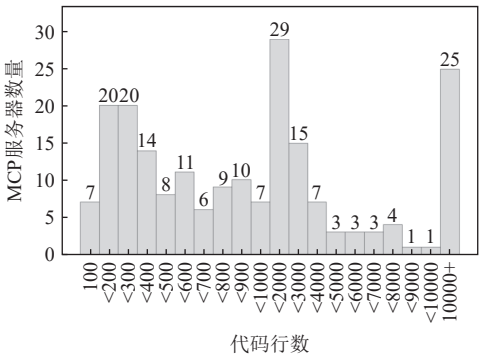


Fig. 3 Line-of-code distribution across MCP servers

图 3 MCP 服务器代码行数分布

此外，观察分布可以发现，34.5% 的 MCP 服务器代码量少于 500 行，这是因为部分 MCP 工具较为简单，往往直接根据传入参数调用对应 API。虽然这些工具功能简单，但其可能开发不完善，未能进行细致授权。最后，图 3 中有 25 个 MCP 服务器代码量超 1 万行，这表明仍有大量服务器功能十分复杂，可能存在多种权限同时使用，面临越权风险。

2)MCP 工具数量分析。通过对 200 个 MCP 服务器进行静态分析，分析框架识别了 2 923 个工具。图 4 中展示了 MCP 服务器内集成的工具的数量分布。由图可知，绝大多数 MCP 服务器都集成了不超过 10 个工具，39.5% 的服务器集成了不超过 5 个工具。这是一种更为安全的实现方式，因为单个 MCP 服务器不会具有太复杂的功能，也会使用更少的权限，降低不同工具间交互导致的间接越权问题。然而，仍然有 35.5% 的 MCP 服务器内部集成了超过 15 个工具，更有 9 个集成了超过 50 个工具。这些复杂的 MCP

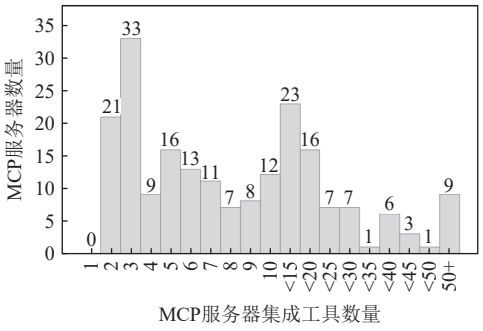


Fig. 4 Number distribution of integrated tools across MCP servers

图 4 MCP 服务器内集成工具数量分布

服务器需要更精细的权限管理。

3)权限使用分析。静态分析框架为 2 923 个工具进行了调用图构建和分析，并分析了其中的权限 API 调用。通过该分析，本节统计了每个 MCP 服务器调用的权限相关 API 接口的数量，并绘制了分布图，如图 5 所示。平均来说，每个 MCP 服务器中所有工具调用的权限敏感 API 个数为 40.74，平均每个 MCP 工具调用的权限敏感 API 数量为 2.79。平均每个 MCP 工具调用的权限敏感 API 数量大于 1，这代表着很多工具有可能未能遵循最小权限原则。理想情况下，每个 MCP 工具仅为 1 个功能设计，应当仅调用 1 个权限敏感的 API，但目前绝大多数的 MCP 服务器未能达成该目标。其中，仍是 AWS 的 MCP 服务器最为复杂，调用了超过 2000 个权限敏感 API，给权限管理带来较大困难。

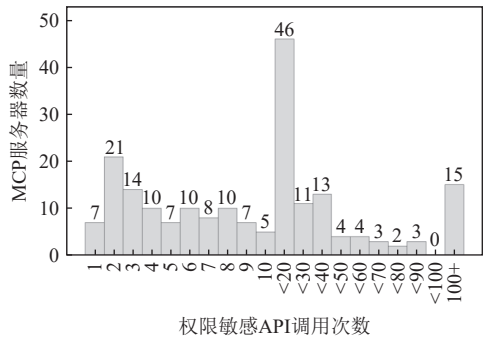


Fig. 5 Number distribution of privileged API calls across MCP servers

图 5 MCP 服务器内权限敏感调用数量分布

此外，观察该分布可以发现，大量服务器调用了 10~20 个权限敏感 API。同时，有 15 个服务器调用超过 100 个权限敏感 API。大量权限集中在某一个服务器内带来了极大的攻击面。一方面，同一个服务器内，各个工具之间的隔离是严重不足的，往往共享内存空间和环境变量。因此，在一个隔离不足的服务器中，一旦某个工具存在漏洞，就可能影响到同一服务器内的其他工具，而过多的权限则会放大这种攻击造成的影响。另一方面，攻击者在实施攻击时，可能需要多个工具相互配合，使用不同类型的权限，构建完整的攻击链。正常情况下，这需要用户恰好同时注册了一组特定的 MCP 服务器，攻击者才能组合攻击链以完成攻击。如果一个服务器包含较多的工具和权限，则攻击者很容易找到可利用的攻击链，在一个 MCP 服务器内部完成攻击。

研究发现 2。总的来说，目前社区开发了功能复杂、使用多种权限的 MCP 服务器。这些服务器存在 3 个风险：1)代码量大、实现复杂、可能存在漏洞。2)功能繁多、缺乏隔离、增大了攻击面。3)权限使用过多，未能保证最小权限原则。

4.4 权限敏感 API 分析

在研究问题 3 中，我们对每个权限敏感 API 进行分类和分析，旨在揭示这些服务器使用了哪些敏感权限，什么场景下可能被滥用，以及可能造成的危害。如表 2 所示，根据这些权限敏感 API 的功能，本研究大致划分了 7 个类别，包括任意命令执行、云服务管理、远程资源修改、应用控制、远程资源读取、互联网内容获取和其他。

1)任意命令执行。这些权限类型中，任意命令执行、云服务管理、远程资源修改被认为是风险等级极高的权限，因为它们能直接对用户的主机、数据和资产造成损失。任意命令执行是指 MCP 服务器使用接口在用户主机上直接进行命令执行，使用 API 诸如 Python 的 *subprocess.run*()。这类的权限敏感接口占比达 11.07%，如果未能对它们的权限进行有效控制，攻击者将能够直接执行恶意命令，造成病毒植入、Token 泄露等严重问题。

Table 2 Classification and Description of Privilege-Sensitive API Calls

表 2 权限敏感 API 调用分类和解释

类别	数量	描述	示例	占比/%
任意命令执行	902	任意执行本地代码和命令，读写本地文件	*subprocess.run*() *os.remove*()	11.07
云服务管理	565	管理云服务系统上的服务和资源	*session.get_credentials*() *create_cluster_v2*()	6.93
远程资源修改	882	在远程服务或数据库上创建或修改数据	*catalog.create_table*() *postgres.insert*()	10.82
应用控制	306	控制本地或远程的私人应用，处理私人事务	*place_market_order*() *move_media_func*()	3.76
远程资源读取	1 948	在远程服务或数据库中读取私有数据	*database.fetch*() *get_current_user_info*()	23.91
互联网内容获取	3 272	从互联网平台搜索和获取公开数据	*search_tweet*() *service.cse*().*list*()	40.16
其他	273	内容处理、内容创作等	*MarkItDown*() *Translate*()	3.35

2)云服务管理。除了在本地的环境中,云服务的管理也是当前 MCP 服务器开发者重点关注的部分。大量的云服务厂商都提供官方的 MCP 工具,这些工具往往调用厂商提供的 API 进行资源管理。例如,AWS 提供 *create_cluster_v2*() 具有创建集群的权限,而 *session.get_credentials*() 则允许用户获取防火墙的相关凭据。统计显示,在所有接口中,6.93% 的 API 接口都与云服务的关键权限和资源相关,将这些权限轻易交由 MCP 服务器,并由大模型进行决策和调用,则必须构建完善的权限控制体系以阻断异常或非法的调用。

3)远程资源修改。用户可能有大量数据放在远程的数据库或服务器上,社区也有大量辅助进行数据管理和分析的工具,约有 10.82% 的 API 涉及到数据的增删改。对这些关键数据进行修改可能导致严重的数据不一致,进而造成真实世界的财产损失。或者至少将这些数据源作为跳板,注入恶意命令,等待其他工具查询时执行危害更大的攻击。

4)应用控制。除了这些直接且风险极大的权限之外,MCP 服务器还往往持有一些与用户个人数据和事务相关的权限,这些被认为是具有高风险的权限敏感 API。如在应用控制中,有 MCP 服务器提供金融服务,可以帮助用户管理个人账户,或者邮箱助手可以进行邮件的撰写和发送。这些权限被滥用也可能间接导致严重财产损失。

5)远程资源读取。远程资源读取可以获取聊天记录、个人信息等敏感信息。针对这些 API 的权限进行攻击虽然难以直接破坏系统设施,但个人数据泄露仍是严重的风险。这些应用和场景较为碎片化,如何构建统一有效的权限控制体系仍需探索。

6)互联网内容获取。首先 MCP 服务器使用了大量 API 调用在互联网进行内容收集。如 *search_tweet*() 可以从推特社区获取公开的帖子,*service.cse*() 则是调用谷歌的 Custom Search Engine 进行联网内容搜索。这些 API 占比 40.16%,这是因为一方面大模型需要引入真实且实时的数据以抑制幻觉并更新知识,故社区需要将不同的数据源接入大模型。其次,虽然部分 MCP 服务器整体是用于其他功能,但其内部可能存在工具从公开的平台获取数据,使用了相关 API。这些调用虽然没有直接与用户权限相关,但一方面,这些搜索调用需要用户进行付费,可能被攻击者滥用。另一方面,攻击者可将这些 MCP 服务器作为攻击入口,将恶意命令从互联网注入大模型。

研究发现 3。MCP 服务器中权限敏感 API 分布广泛且高度集中于任意命令执行、云服务管理、远程资源修改等高危能力,同时大量涉及远程数据读取与互联网内容获取的接口被交由大模型自动调用,放大了本地与云端资产被直接破坏的风险,并为通过数据注入实现命令劫持和隐蔽攻击提供了入口。

4.5 授权机制分析

在研究问题 4 中,探究了这些 MCP 服务器中的授权机制,调研目前开发者对于权限管理和安全方面的考量和实践,结果如图 6 所示。在这些 MCP 服务器中,除去没有进行授权及鉴权的服务器外,本研究总共发现了 4 类授权方案,分别是 Token 授权、OAuth 授权、密码授权和其他授权方法。

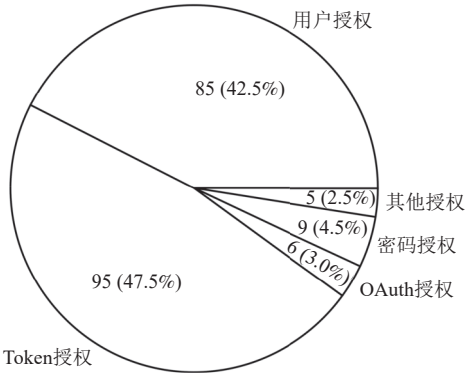


Fig. 6 Authorization method distribution across MCP servers

图 6 MCP 服务器授权方法分布

1)用户授权。首先,有 42.5% 的 MCP 服务器没有进行显式授权和鉴权行为,仅仅依赖人机交互层面的确认。具体来说,MCP 服务器的基础设计准则是由用户为其自己的行为负责,即每次大模型调用一个 MCP 服务器的工具时,其必须显式地提醒用户,征询用户同意。此时,用户的准许行为就构成了基础的授权,得到用户同意的 MCP 服务器即可调用相关 API 执行权限行为。而由于大量 MCP 服务器设计是用于用户本地主机,进行本地文件读写和应用操作等,MCP 服务器等同于取得和用户相同的权限,只要用户同意执行工具,就不需要额外的授权步骤。但是,值得注意的是,此处用户进行授权的判断时,其只获得了 MCP 工具的粗略功能信息,难以判断 MCP 工具的实际行为,无法判断其权限等级。

2)Token 授权。从图 6 可以看出,目前社区中最广泛使用的授权方式是通过 API Token 进行授权。这主要是由于基于 Token 的授权是进行 MCP 服务器和工具开发最便捷的方案。当前主流的服务都会为其 API 配置基于 Token 的授权方式,故只需要提供 MCP 服务器一个 Token,不需要修改 API 的服务端,

就可以完成MCP服务器的开发和部署。

但是，由于Token的申请与管理通常依赖手动配置，这些Token往往没有设置细粒度的权限管理。如大量服务允许用户使用1个Token调用所有的API接口，这意味着1个Token的授权其实授予了MCP服务器大量冗余权限，违背了最小权限原则。此外，尽管部分厂商会对Token的权限进行一定程度的划分，如GitHub会将公开和私有的仓库分开授权。但由于Token申请繁琐，大量用户都会选择直接申请最大权限Token，如在GitHub上申请可访问所有仓库的Token。同时，为了方便，用户往往选择配置长期有效的Token。这些问题叠加带来严重的权限泄露风险。最近，研究者已经发现由于用户使用过高权限的GitHub Token，导致攻击者将用户私有仓库的代码上传至公开仓库，导致核心代码的泄露。

根据上面的分析，本研究发现，基于Token的授权机制短期内仍是主流的授权方式，且其潜在风险仍然没有引起广泛的关注。因为替代方案需要对太多的服务进行改造，目前进展缓慢，社区应当鼓励设置更为精细的Token，限制Token有效时间。同时，MCP服务器也需要添加额外的防护机制，进行额外的权限管控，确保其在被攻击者利用时可以及时阻断攻击。

3）密码授权。密码授权是一部分数据库和应用软件仍在使用的授权方式，如登录社交媒体或连接数据库。在MCP服务器中，仍有部分服务器采用基于密码的授权，直接使用用户的账号密码，登录对应的应用或服务，获取对其的所有操作权限。该方法也是一个实现简便、易于部署的方案。但是通过账户密码进行授权，则意味着MCP服务器掌握了该账号的所有权限，严重违背了最小权限原则，带来极大安全隐患。

经过分析，只有9个MCP服务器使用了该方案，其主要有2个方面的原因。一方面是由于用户安全意识不足、密码强度较低，导致基于密码的授权方式在实践中被普遍视为不可靠，故当前大量应用和服务都在密码授权的基础上要求进行二次校验，这些二次校验导致无法使用MCP服务器进行自动化登录。另一方面，将用户密码提供给MCP服务器后，MCP服务器往往以明文的方式将其存储在本地，存在巨大的安全隐患，若因其他攻击或MCP服务器自身漏洞导致信息泄露，攻击者可以直接凭借密码获得对应软件的全部权限，故该方案并非业界广泛接受的授权方案。

4）OAuth授权。当前MCP社区认为最佳的授权技术方案是使用OAuth框架进行授权和鉴权。其核心思想是，客户端不能直接访问资源所有者的受保护资源，而是需要首先从授权服务器获取一个授权令牌。然后，客户端凭借这个有时效性的令牌去请求资源服务器上的数据，资源服务器验证令牌有效后才会返回资源。因为该令牌是每次调用独立请求，授权服务器可以根据其请求，授予临时令牌最小必要权限。且该令牌时效性有限，最大程度地降低了权限泄露的风险。

尽管OAuth被认为是业界领先的授权方案，但是本研究中只有6个MCP服务器借助它进行授权。这是因为完整的OAuth 2.0框架要求服务方提供对应的授权服务器，这需要对现有服务进行大量改造，带来巨大的工作量。因此，对于很多服务来说，开发者只能被迫转向，采用更简单的令牌认证方式（例如静态的API Token）。同时，尽管MCP社区在积极宣传使用OAuth，但因为主要的改造工作量在服务提供方而非社区，故如何进一步推进该方案仍是一个挑战。

5）其他授权。除了以上的授权方式之外，也有部分MCP服务器采用其独特的授权机制。如AWS在开发了复杂的MCP服务器后，也基于其云服务系统的访问控制框架，为MCP服务器专门设计了授权机制，将MCP服务器视为参与云服务的实体并授予特定的权限。同时，在部分安全攸关场景，AWS虽然允许MCP服务器调用API进行数据的增删改，但其限制MCP服务器只能修改其自身写入的数据，不能改变用户的数据。这些定制化的权限控制框架和规则有效地限制了MCP服务器的权限。类似的，谷歌云也使用其云服务自带的权限控制机制，对用户使用的MCP服务器进行权限控制。这类方案由云服务厂商针对其自身开发，具有较高的安全性，但在开源MCP社区中，开发者难以在其他场景下推广。

总的来说，在所有MCP服务器中，超过一半以上都进行了授权行为，这一方面代表它们代替用户持有了关键的权限，另一方面也代表社区具有较好的安全意识，一定程度正在降低权限泄露的风险。但由于客观原因，完全实现最小权限原则仍有较大挑战，需要厂商和社区的协同推进。

研究发现4。当前MCP服务器的授权机制以用户授权和基于Token的认证为主，密码授权虽已边缘化但仍存在高危使用场景，而被视为最佳实践的OAuth仅在少数服务器中落地，整体上授权实践仍难以有效落实最小权限原则，使得关键权限在集中委

托给 MCP 服务器的同时,持续暴露出较高的滥用与泄露风险。

5 讨论与展望

本研究发现,尽管 MCP 服务器社区正处于快速扩张期,功能覆盖日益广泛,但其安全性,尤其是权限管理与访问控制,并未得到应有的重视。针对这一现状,本文提出以下改进建议,以促进 MCP 生态的安全演进。

首先,应遵循职责分离与最小权限原则,控制单个 MCP 服务器的功能复杂度。开发者应避免构建集成过多功能的单体服务器,而应专注于特定功能的服务器实现。集成度过高的工具集意味着用户必须一次性授予大量权限,这种过度的权限暴露可能被攻击者利用,直接构建完整的攻击链。特别是,应当在设计层面隔离信息获取类工具与高危执行类工具,防止攻击者利用前者作为攻击入口劫持 LLM,进而利用后者实施破坏,降低 MCP 服务器被攻击的风险。

其次,面向智能体的授权与鉴权机制亟待重构和改进。当前生态普遍沿用基于 Token 的鉴权方式。该机制本质上是面向人类用户设计的,假设用户具备足够的安全意识,会设置适当权限,合理地使用 Token; 同时考虑到人类用户无法管理过于细粒度的权限,现有的权限划分较为宽泛且不够细致。然而,将其直接迁移至由大模型驱动的 MCP 场景存在显著的安全失配。一方面,目前的 MCP 服务器主要由大模型驱动,而大模型本身尚缺乏内生的安全判断能力,极易受提示注入攻击; 另一方面,相比人类,智能体具备更强的自动化处理能力,能够进行更为精细化、动态化的权限管理。因此,传统的粗粒度授权已成为安全瓶颈,构建面向智能体的细粒度、动态授权机制将是未来的重要研究方向。

6 研究局限与效度威胁

在一定程度上,本研究的样本规模受限于半自动化的分析流程。目前仅对 200 个获 Star 数量大于 20 的 MCP 服务器及其 8 000 余个 API 接口调用进行了深度分析,可能存在数据偏差,即未能充分覆盖社区中大量开发尚不完善、关注度较低的小型项目。下一步,本研究将进一步提升分析自动化程度,以对更多 MCP 服务器进行分析。

此外,虽然 Python 和 TypeScript 占据了社区主流,但官方已提供 Java、Go 等语言的 MCP 服务器开发工具套件,且已有大量基于这些语言的服务器出现。由于跨语言静态分析工具的适配成本较高,本研究暂未包含其他语言的项目。在实际调研中,我们观察到 MCP 服务器的语言选择存在一定异构性,这主要取决于开发者的技术偏好及服务自身的架构特性。具体而言,Java 常被用于构建与现有 Java 后端系统强耦合的 MCP 服务器,以实现更优的集成效率; 而 Go 语言则更多出现在云服务提供商发布的官方 MCP 服务器中,或被特定 Go 技术栈的开发团队所采用。然而,鉴于跨语言静态分析技术面临的挑战,不同语言特性的差异要求分析工具进行针对性的适配与开发,故本研究现阶段主要覆盖占据主导地位的 MCP 服务器。未来工作计划进一步扩展静态分析工具的适配范围,以涵盖更多编程语言,从而全面评估多语言混合生态下的安全风险。

最后,在服务器分析和权限分析流程中,仍有部分步骤分析精度受限,可能导致结果偏差。由于 Python 和 TypeScript 均为动态类型语言,广泛存在的动态特性(如反射、动态函数调用)对调用图的构建带来了挑战,可能导致部分函数指针无法精确解析。为缓解这一威胁,本研究采用了轻量级指针分析技术,在保证分析效率的同时,尽可能构建完备的调用图,以减少漏报风险。在部分分析环节,本研究依赖 LLM 进行语义分类,模型潜在的幻觉问题可能影响结果的精确度。针对该威胁,本研究采取了多重消解策略。一方面,通过集成搜索工具构建检索增强生成(RAG)流程,为模型提供实时、精确的上下文信息; 另一方面,采用多模型交叉验证与多次采样投票机制,最大限度地缓解随机性干扰,并配合人工分析,确保分析结果的可靠性。

7 结 论

随着 LLM 为执行复杂的现实世界任务,越来越多地通过 MCP 与外部工具进行交互,其权限使用的安全性问题日益凸显。本研究通过收集 1 609 个开源 MCP 服务器,并对其中 200 个进行深入分析,系统性地揭示了当前 MCP 服务器和工具在权限管理方面存在的安全风险。研究发现,MCP 服务器广泛应用于数据访问、开发者工具和云服务管理等高风险场景,这些工具平均代码量大、功能集成度高,且平均每个服务器调用超过 40 个权限敏感 API。进一步分析表明,这些服务器普遍使用了包括任意命令执行、

远程资源修改和云服务管理在内的高危敏感权限。然而，在授权机制上，高达52%的权限验证依赖于粗粒度的API Token和账号密码，严重违背了最小权限原则，为权限滥用埋下了严重的安全隐患。研究表明，当前MCP生态中功能复杂的工具、对敏感权限的普遍使用以及粗放的授权机制，共同构成了一个广阔而脆弱的攻击面。鉴于LLM自身易受提示注入等方式攻击影响的特点，这些权限冗余的工具极易被攻击者劫持，对用户的数字资产和关键基础设施构成直接且严重的威胁。因此，本研究呼吁开发者应严格遵循最小权限原则，并积极探索如OAuth等更安全的授权机制，以共同构建安全可靠的AI智能体系统。

作者贡献声明：张泉负责方案整体设计和实验并撰写论文；冯诀宵负责部分工程实践和实验实施；周炽金负责指导方法设计，并负责论文写作；姜宇对方案设计提供了重要意见，并修改论文。

参 考 文 献

[1] Kasneci E, Sessler K, Küchemann S, et al. ChatGPT for good? On opportunities and challenges of large language models for education[J]. Learning and Individual Differences, 2023, 103: 102274

[2] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]// Proc of the 31st Annual Conf on Neural Information Processing Systems. Red Hook, NY: Curran Associates, Inc. , 2017: 5998−6008

[3] Maharana A, Lee D H, Tulyakov S, et al. Evaluating very long-term conversational memory of LLM agents[C]//Proc of the Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2024: 13851−13870

[4] Wikipedia. Model Context Protocol[EB/OL]. 2025[2025-09-15]. https://en.wikipedia.org/wiki/Model_Context_Protocol

[5] Khoei T T, Ehtesham A, Kumar S, et al. A survey of the model context protocol (MCP): Standardizing context to enhance large language models (LLMs)[J]. arXiv preprint, arXiv: 2025040245, 2025

[6] GitHub. GitHub official MCP server[EB/OL]. 2025[2025-09-18]. https://github.com/github/github-mcp-server

[7] AWSLabs. AWS official MCP servers[EB/OL]. 2025[2025-09-18]. https://github.com/awslabs/mcp

[8] Zhao W X, Zhou Kun, Li Junyi, et al. A survey of large language models[J]. arXiv preprint, arXiv: 2303.18223, 2023

[9] Hou Xinyi, Zhao Yanjie, Wang Shenao, et al. Model context protocol (MCP): Landscape, security threats, and future research directions[J]. ACM Transactions on Software Engineering and Methodology, 2026 [2025-11-20]. https://doi.org/10.1145/3796519

[10] Xu Honghui, Li Kaiyang, Chen Wei, et al. A survey: Towards privacy and security in mobile large language models[J]. arXiv preprint, arXiv: 2509.02411, 2025

[11] Fu Yuchuan, Yuan Xiaohan, Wang Dongxia. RAS-Eval: A comprehensive benchmark for security evaluation of LLM agents in real-world environments[J]. arXiv preprint, arXiv: 2506.15253, 2025

[12] Kong Dezhang, Lin Shi, Xu Zhenhua, et al. A survey of LLM-driven AI agent communication: Protocols, security risks, and defense countermeasures[J]. arXiv preprint, arXiv: 2506.19676, 2025

[13] Das B C, Amini M H, Wu Yanzhao. Security and privacy challenges of large language models: A survey[J]. ACM Computing Surveys, 2025, 57(6): 1−39

[14] Ye Wentao, Hu Jiaqi, Wang Haobo, et al. A trusted evaluation system for safe deployment of large language models[J]. Journal of Computer Research and Development, 2025, 62(7): 1668−1684 (in Chinese)
(叶文涛, 胡家齐, 王皓波, 等. 面向大语言模型安全部署的可信评估体系[J]. 计算机研究与发展, 2025, 62(7): 1668−1684)

[15] Zhang Chi, Zhu Changjia, Xiong Junjie, et al. Guardians and offenders: A survey on harmful content generation and safety mitigation of LLM[J]. arXiv preprint, arXiv: 2508.05775, 2025

[16] Goldstein O, La Malfa E, Drinkall F, et al. Jailbreaking large language models in infinitely many ways[J]. arXiv preprint, arXiv: 2501.10800, 2025

[17] Li Nan, Ding Yidong, Jiang Haoyu, et al. Jailbreak attack for large language models: A survey[J]. Journal of Computer Research and Development, 2024, 61(5): 1156−1181 (in Chinese)
(李南, 丁益东, 江浩宇, 等. 面向大语言模型的越狱攻击综述[J]. 计算机研究与发展, 2024, 61(5): 1156−1181)

[18] Chen Kang, Zhou Xiuze, Lin Yuanguo, et al. A survey on data security in large language models[J]. arXiv preprint, arXiv: 2508.02312, 2025

[19] Wang Yixu, Teng Yan, Huang Kexin, et al. Fake alignment: Are LLMs really aligned well?[C]//Proc of the 2024 Conf of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Stroudsburg, PA: ACL, 2024: 4696–4712

[20] Liu Yupei, Jia Yuqi, Geng Runpeng, et al. Formalizing and benchmarking prompt injection attacks and defenses[C]//Proc of the 33rd USENIX Security Symp. Berkeley, CA: USENIX Association, 2024: 1831−1847

[21] Yeo A, Choi D. Multimodal prompt injection attacks: Risks and defenses for modern LLMs[J]. arXiv preprint, arXiv: 2509.05883, 2025

[22] Zhan Qiusi, Fang R, Panchal H S, et al. Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents [C]//Proc of Conf of the North American Chapter of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2025: 7116–7132

[23] Yi Sibo, Liu Yule, Sun Zhen, et al. Jailbreak attacks and defenses against large language models: A survey[J]. arXiv preprint, arXiv:

2407.04295, 2024

[24] Tai Jianwei, Yang Shuangning, Wang Jiajia, et al. Survey of adversarial attacks and defenses for large language models[J]. Journal of Computer Research and Development, 2025, 62(3): 563−588 (in Chinese)
(台建玮, 杨双宁, 王佳佳, 等. 大语言模型对抗性攻击与防御综述[J]. 计算机研究与发展, 2025, 62(3): 563−588)

[25] He Pengfei, Xing Yue, Xu Han, et al. Multi-faceted studies on data poisoning can advance LLM development[J]. arXiv preprint, arXiv: 2502.14182, 2025

[26] Zhao Shuai, Jia Meihuizi, Guo Zhongliang, et al. A survey of recent backdoor attacks and defenses in large language models[J]. arXiv preprint, arXiv: 2406.06852, 2024

[27] Wang Qiaochen, Wu Zhengang, Liu Hu. A survey on security and privacy problems in applications of large language models[J]. Industry Information Security, 2024(5): 40−45 (in Chinese)
(王乔晨, 吴振刚, 刘虎. 大语言模型应用的安全与隐私问题综述[J]. 工业信息安全, 2024(5): 40−45)

[28] Chen Yulin, Li Haoran, Sui Yuan, et al. Can indirect prompt injection attacks be detected and removed?[C]//Proc of the 63rd Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2025: 18189−18206.

[29] Li Ang, Zhou Yin, Raghuram V C, et al. Commercial LLM agents are already vulnerable to simple yet dangerous attacks[J]. arXiv preprint, arXiv: 2502.08586, 2025

[30] Liu Mingxing, Wang Junfeng, Lin Tao, et al. An empirical study of the code generation of safety-critical software using LLMs[J]. Applied Sciences, 2024, 14(3): 1046

[31] Xu Zihao, Liu Yi, Deng Gelei, et al. A comprehensive study of jailbreak attack versus defense for large language models[C]//Proc of the 62nd Annual Meeting of the Association for Computational Linguistics: Findings. Stroudsburg, PA: ACL, 2024: 7432−7449

[32] Model Context Protocol. What is the model context protocol (MCP)?[EB/OL]. 2025[2025-09-15]. https://modelcontextprotocol.io/docs/getting-started/intro

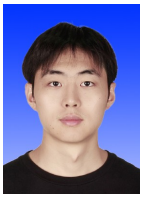
[33] Krishnan N. Advancing multi-agent systems through model context protocol: Architecture, implementation, and applications[J]. arXiv preprint, arXiv: 2504.21030, 2025

[34] NIST. Least privilege[EB/OL]. 2025[2025-09-15]. https://csrc.nist.gov/glossary/term/least_privilege

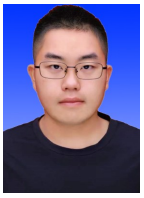
[35] Xu Chen, Zhu Runsu, Wu Zhenzhou, et al. Research on access control security model[J]. Cyberspace Security, 2024, 15(1): 97−102 (in Chinese)
(徐晨, 朱润酥, 吴振洲, 等. 访问控制安全模型研究[J]. 网络空间安全, 2024, 15(1): 97−102)

[36] Sandhu R S. Role-based access control[J]. Advances in Computers, 1998, 46: 237−286

[37] Cutler J W, Disselkoen C, Eline A, et al. Cedar: A new language for expressive, fast, safe, and analyzable authorization[J]. Proceedings of the ACM on Programming Languages, 2024, 8(OOPSLA1): 670−697

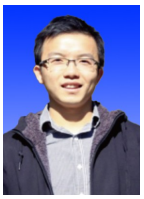


Zhang Quan, born in 1998. PhD, lecturer. His main research interests include intelligent agent security and system security.
张　泉, 1998 年生。博士, 讲师。主要研究方向为智能体安全、系统安全。



Feng Juexiao, born in 1998. PhD. His main research interests include artificial intelligence and intelligent software.
冯诀宵, 1998 年生。博士。主要研究方向为人工智能、智能软件。

Zhou Chijin, born in 1995. PhD, lecturer. His main research interests include program analysis and system security.
周炽金, 1995 年生。博士, 讲师。主要研究方向为程序分析、系统安全。



Jiang Yu, born in 1989. PhD, associate professor, His main research interests include cyber-physical systems and software systems security.
姜　宇, 1989 年生。博士, 副教授。主要研究方向为信息物理融合系统、软件系统安全。