arXiv:2603.20933v1 [cs.CR] 21 Mar 2026

AC4A：智能体访问控制

RESHABH K SHARMA，美国华盛顿大学DAN GROSSMAN，美国华盛顿大学

大语言模型（LLM）智能体将大语言模型的聊天交互能力与调用外部工具和 API 的能力相结合，使其能够执行复杂任务并自主采取行动以实现用户目标。然而，当前的智能体系统采用全有或全无模式：代理要么完全访问某个 API 的功能和某个网页的内容，要么完全无法访问。这种粗粒度方法迫使用户授予代理超出特定任务实际需求的能力。例如，当要求代理在日历中创建事件时，用户必须授予它对整个日历 API 的访问权限，这可能使代理能够随时创建事件、修改现有事件或访问现有事件。

本文介绍了*AC4A*，一种面向智能体的访问控制框架。随着智能体能力日益增强、自主性不断提高，用户需要一种方法来限制这些智能体可以访问哪些 API 或网页的哪些部分，从而无需将 API 或网页允许的一切都托付给它们。*AC4A* 旨在提供一种权限定义框架，让智能体只能访问获准访问的资源。*AC4A*适用于基于 API 和基于浏览器的智能体。它不规定权限应如何设置，而是提供一种灵活的定义和实施权限的方式，使其适用于现实世界中的系统。

AC4A 通过创建授予资源访问权限的许可来运作，其设计借鉴了成熟的访问控制框架，例如 Unix 文件系统的访问控制框架。应用程序将资源定义为层级结构，并提供一种方法，在运行时计算成功访问资源所需的权限。该框架会跟踪当前已授予的权限，并提供一个权限管理仪表板，用于创建和撤销权限。

我们通过案例研究展示了*AC4A* 在真实世界的 API 和网页上实施权限控制的实用性。AC4A 的源代码可在 *https://github*.com/reSHARMA/AC4A。。

ACM 参考格式：
Reshabh KSharma 和 Dan Grossman。2026。AC4A：智能体访问控制。第 1 卷，第 1 期（2026年3月），25页。https://doi.org/10.1145/nnnnnnn。nnnnnnn

1 引言

大语言模型（LLM）已从被动的问答系统发展为能够通过 API 和网页浏览器与外部世界交互的主动智能体 [9, 29, 30]。如今，这些智能体可以通过调用外部服务和浏览网页界面，执行预订旅行、管理日历或编写代码等复杂任务 [1, 2]。尽管这种自主性带来了强大的应用能力，但也引发了资源滥用问题，而现有系统并不具备妥善应对此类问题的能力。

与传统软件系统不同，当前的大语言模型智能体缺乏访问控制框架。尽管访问控制系统本身无法抵御所有安全威胁，但它们是使其他安全机制得以发挥作用的关键组件。当前的智能体系统采用全有或全无模式：代理要么拥有对 API 全部功能以及网页的完全访问权限，要么

作者地址：Reshabh K Sharma，华盛顿大学，美国华盛顿州西雅图，reshabh@cs.washington.edu；Dan Grossman，华盛顿大学，美国华盛顿州西雅图，djg@cs.washington.edu。

允许为个人或课堂使用而免费制作本作品全部或部分内容的数字副本或纸质副本，前提是这些副本不得用于营利或商业用途，且每份副本均须附有本声明和首页上的完整引文。对于本作品中归 ACM 以外其他方所有的组件，必须遵守其版权规定。注明出处后允许摘录。若要以其他方式复制、再版、发布到服务器或分发至列表，须事先获得明确许可和/或支付费用。请向 permissions@acm.org 申请许可。© 2026 计算机协会。
XXXX-XXXX/2026/3-ART $15.00 https://doi.org/10.1145/nnnnnnn.nnnnnnn

内容，或者完全无法访问。这种粗粒度方法迫使用户信任智能体拥有超出特定任务实际需求的能力。例如，当用户要求代理在日历中创建事件时，必须授予它访问整个日历 API 的权限，这可能使代理能够随时创建事件、修改现有事件或访问现有事件。同样，使用基于浏览器的智能体查看日历时，该智能体会获得对整个网页的访问权限，其中包括其他日历条目和个人详细信息等敏感信息。

访问控制的缺失与智能体面临的其他安全挑战有所不同。与智能体及其底层使用的大语言模型相关的许多重要安全问题，包括幻觉 [12],提示注入攻击 [13, 25], 和恶意指令隐藏 [31, 32]。不同于针对这些问题的防御措施，访问控制框架无法直接缓解它们，这类似于传统系统中的缓冲区溢出、侧信道攻击和网络钓鱼等漏洞无法仅通过访问控制系统加以解决。然而，很难想象一个没有访问控制系统的安全系统。我们首先专注于控制智能体能够访问哪些资源，我们认为这是应对其他安全问题的前提。

我们提出了*AC4A*，这是一个面向大语言模型智能体的访问控制框架。*AC4A* 提供了用于定义细粒度权限的基础设施。我们的框架同时适用于基于 API 和基于浏览器的智能体，并采用统一的权限基础设施，未来可扩展以支持新类型的智能体实现。

AC4A 通过创建授予资源访问权限的许可来实现访问控制。应用程序以层级结构定义其资源。*AC4A* 中的许可由资源和应用程序定义的操作组成，例如读取、写入或创建。该框架包含一个权限管理仪表板，可用于创建、查看和移除许可。在运行时，当智能体调用 API 或与网页交互时，系统会拦截该调用，并且只有在存在明确许可的情况下，才会授予对相应资源的访问权限。

我们的贡献是提出了一种面向智能体的访问控制框架，API 型智能体和基于浏览器的智能体使用相同的权限表示。据我们所知，*AC4A* 是首个为基于浏览器的智能体提供资源级访问控制的框架，也是首个统一这两类智能体权限执行的框架。本文其余部分安排如下。第2 节介绍*AC4A*所针对的威胁模型。第3 节介绍*AC4A*的设计，涵盖权限表示与执行。第4 节介绍用于管理权限的用户界面。第5 节通过两个案例研究展示*AC4A*： 一个用于说明核心组件的井字棋游戏，以及一项涉及 Outlook 日历、Expedia 和支付钱包的航班预订任务。第6 节讨论局限性和未来工作。第7节讨论相关工作。第8节作总结。

2 威胁模型

我们的访问控制框架的主要目标是为智能体强制实施给定的一组权限。我们的威胁模型有意限定在访问控制层。我们假设代理不可信，而访问控制框架本身及其获得的权限可信。该框架的保证严格限于正确实施其所获授的权限。

智能体与底层大语言模型。 代理是核心的不可信实体。无论其来源或意图如何，代理都可能遭到攻陷，或本身就是出于恶意而设计的。同样，底层大语言模型也不可信，可能存在偏见、实现错误或后门。即使代理出于善意，大语言模型也可能因实际用户意图与其对意图的理解不一致而出错。例如，用户要求代理寻找合适的航班，代理却可能在未经明确授权的情况下实际购买航班。如果用户要求寻找

日历上安排会议的空档，代理可能会自作聪明地删除现有事件来腾出空档。访问控制框架可确保，除非用户明确授予必要权限，否则不会发生此类非预期操作。*AC4A* 的价值在于确保不完善的指令或不完善的大语言模型不会执行用户未明确许可的操作。作为一种访问控制系统，*AC4A* 无法确保执行的是正确的操作，只能确保执行的任何操作都在允许的权限范围内。

API、网页和应用程序提供方。我们假设用户已经信任自己使用的应用程序，例如日历或旅行预订网站。我们的框架控制智能体如何使用这些应用程序，而不是控制应用程序本身。我们信任应用程序能够准确地将其 API 端点和网页 DOM 元素映射到它们所提供的资源。例如，我们信任日历应用程序能够正确识别：删除特定日期的日历条目需要对该日期拥有写入权限。我们也信任应用程序不会泄露或损坏数据，这与传统系统中对 Unix 文件系统实现的信任一样。

最终用户。本文假设最终用户会向智能体授予权限。我们的系统也适用于由不同主体授予权限的情形。例如，在企业环境中，权限可能仅由主管或法务部门授予。无论如何，我们信任授予的权限是正确的。这些权限由 *AC4A* 直接管理，未经最终用户允许，智能体无法修改这些权限。

访问控制是实现更高层次安全属性的必要但不充分的工具。我们的框架并不旨在保证所有对资源的非预期访问都能被阻止，而是执行所提供权限集中规定的限制。如果受信任的作者遗漏或错误配置了必要的限制，框架就无法防止资源滥用。例如，假设某个日历应用程序在事件中包含重复信息，例如每周。如果代理可以访问本周的信息，却不能访问下周的信息，它仍可能通过重复元数据推断出下周的事件信息。这是否构成应用程序权限函数中的缺陷尚有争议，但这超出了访问控制执行的范围。

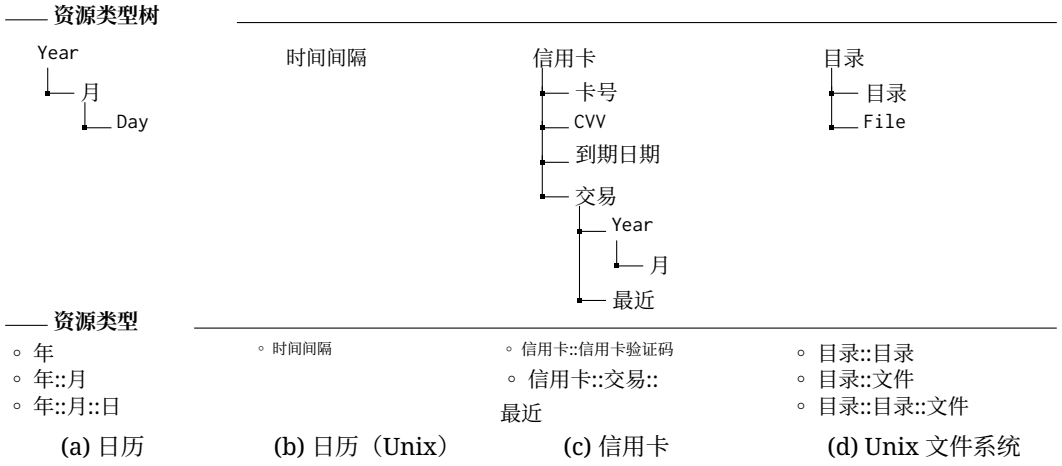
3 设计

权限是*AC4A*的核心。*AC4A* 中的一项权限由资源值规范和操作组成。资源值规范表示一组资源值，操作则表示一项操作。*AC4A* 将资源值视为不透明实体，并不了解它们代表什么，但每个资源值都有一种资源类型。应用程序使用*AC4A*提供的接口定义资源类型。

3.1 定义资源类型

资源类型定义为一条起始于资源类型树根节点的有向路径。它表示一组资源值，并由资源类型节点通过::Operator连接而成。资源类型树中的边表示父子关系，即父节点所表示的资源值集合是子节点所表示资源值的超集。在图1中，我们展示了日历、文件系统和信用卡信息的资源类型树及资源类型示例。应用程序可以针对同一组资源，以不同方式进行分解，从而建模多个资源类型树，如图1（a）和（b）所示，其中日历事件在（a）中按日期表示，在（b）中则按 Unix 时间戳区间表示。

如图所示1(a)，这三种资源类型分别是年、年::月和年::月::日。我们期望（但*AC4A*不会强制要求）同一特定类型（例如年::月）的不同资源值互不相交，因为 2026 年 3 月的资源与 2026 年 2 月的资源互不相交。树结构确实意味着并假定 2026 年的资源是 2026 年 3 月和 2026 年 2 月资源的超集。资源值不一定要互不相交，也并非总是



(a) 日历 (b) 日历（Unix） (c) 信用卡 (d) Unix 文件系统

图 1. 资源类型树及其派生出的资源类型。(a) 和 (b) 展示了日历资源的两种替代表示方式。(c) 展示了一个具有重叠访问路径的信用卡模型。(d) 展示了一个递归文件系统模型。

所需。例如，图1(b) 中的时间间隔资源类型，其值可以表示任意时间间隔（因此，资源值集合可以表示任意间隔集合）。资源类型也不要求互不相交，它们也可以具有共同的资源值。同一资源类型树可以结合互不相交和重叠两种方式。例如，在图1(c) 中，我们期望信用卡的信用卡验证码、到期日期和交易组成部分是互不相交的资源；但对于交易，可以针对某一年或某一月，或者最近的 k 笔交易授予权限，这两种方式都只允许访问部分交易历史。

资源类型中的多个资源类型节点可以具有相同的名称，从而使资源类型树具有递归性，并导致树无限延伸。例如，在图 1(c) 中，目录节点可能会多次出现。即使在资源类型树无限延伸的情况下，资源类型仍然是有限的，因为它们是从根节点出发的定长路径。

资源类型的主要用途是表示父子关系，而非强制语义有效性。*AC4A* 既不定义也不验证任何资源类型的合法值。例如，类型 年::月 可以表示值 *3*、三月，甚至是 三月月。同样，*AC4A* 将表示 Unix 时间戳的时间间隔视为不透明字符串。应用程序可根据需要对这些值施加自己的约束。我们不需要语义有效性系统的原因是，我们的目标仅仅是检查（如第 3.4 节所述）授予代理的权限是否足以满足某项操作所需的权限。授予 Year(-3)::Month(Marchuary) 权限并无害处，除非某个 API 调用需要此类权限才能继续。

3.2 资源值与资源值规范

一个资源值规范表示一组资源值。它写作 TypeNode1(*Val1*)::TypeNode2(*Val2*)::...::TypeNodeN(*ValN*)，其中每个*Val*要么是字符串，要么是通配符?（表示该资源类型节点所有可能值的并集），每个 TypeNode 则是沿资源类型树中的一条路径、通过:: 操作符链接起来的资源类型节点。在图2中，我们展示了由相应资源值规范表示的资源值集合示例。

资源值规范	资源值集合
年(2026)::月(十月)::日(15)	2026年10月15日的日历事件
年(?)::月(一月)	所有年份一月的日历事件
年(2026)::月(?)::日(15)	2026年每个月15日的日历事件
时间间隔(1696809600–1696896000)	2023年10月9日至10日的日历事件
目录(/)::目录(主目录)::目录(user)::文件(report.txt)	路径 /主目录/user/report.txt 下的文件
目录(/)::目录(home)::目录(?)::文件(report.txt)	所有名为 report.txt 的文件，位于任意直接主目录的子目录中
信用卡(?)::交易(?)::最近(5)	所有信用卡最近的5笔交易
信用卡(“GoldPlus”)::交易(?)::年(2026)::月(一月)	GoldPlus信用卡在2026年一月的所有交易 card

图2. 资源值规范及其对应的资源值集合。

3.3 定义操作

AC4A 采用一种简单的访问控制模型，其中每个应用程序都定义自己的一组操作。操作表示一种可对资源执行的操作类型。与资源类型节点不同，操作没有结构，并且假定彼此不重叠。大多数应用程序可能会定义读取、写入和创建操作，但应用程序也可以根据需要自由定义其他操作。将读取／写入／创建用作操作仅仅是一种惯例，*AC4A*不会对操作施加任何含义。尽管读取／写入／创建这一惯例确实简单，也不够精确，但对大多数应用程序而言已足够适用。读取涵盖所有使代理能够查看资源、但不修改底层资源存储库的操作，包括资源检索和查询操作、搜索、资源显示以及只读 API 调用。写入权限涵盖修改资源系统内部状态的操作，包括资源修改和删除。创建允许创建新的资源实例，而无需授予该资源一概而论的写入权限。例如，日历应用程序可以定义读取操作以允许查看日历事件，定义写入操作以允许修改或删除事件，并定义创建操作以仅允许创建新事件。文件系统应用程序可以使用 {read, write}，其中读取涵盖文件检索和目录列表，写入则涵盖文件创建、修改和删除。应用程序可以自由定义删除和修改等其他操作。

AC4A 中的权限*AC4A*由资源值规范（第 3.2 节）和操作（第 3.3 节）组成。当资源值规范包含 ? 时，该权限代表所有权限的并集，其中 ? 分别实例化为该资源类型节点的每个可能值。图 3 展示了 *AC4A* 中权限的示例。

3.4 权限检查

当代理尝试访问资源时，*AC4A* 必须确定代理当前已获授的权限是否足够。这需要了解两点：（1）访问所需的权限；（2）已授予代理的权限。第3.5节说明了*AC4A*如何确定代理继续操作所需的权限。第4节介绍了用户通过其授予权限的用户界面。在本节中，我们假定这两者均已知，并说明如何检查已授予的权限是否足以允许访问。

资源值规范	操作	描述
年(2026)	写入	对值为 2026 的 Year 类型资源进行写入访问。
Year(?)::Month(一月)	read	读取所有值为 一月的 Month 类型资源，涵盖所有年份。
目录(主目录)::目录(?) ::文件(report.txt)	read	访问所有名为 report.txt 的文件以读取任意 信用卡模型中主目录的直接子目录里的名为 report.txt 的所有文件。
信用卡(?)::交易(?) ::最近(5)	read	读取所有信用卡中的最近 5 笔交易 在信用卡模型中。

图3. 权限示例。每项权限都是由资源值规范和操作构成的一对。

我们关注的是：已授予的资源值规范所表示的资源值集合是否涵盖代理执行特定操作所需的资源值集合。我们会针对每个操作分别重复此检查，以确保一个操作的权限不能满足另一个操作的需求。要确定已授予的集合是否涵盖所需的集合，*AC4A* 使用由每个应用程序定义的资源_差异函数。*AC4A* 在遍历已授予权限时使用资源_差异，同时跟踪必要权限中哪些部分尚未确定已被授予。

3.4.1 资源差异函数。 每个应用程序都实现了一个资源_差异函数，其签名如下：

资源_差异(需求，已有) →剩余

其中，需求 和已有 是资源值规范（其中可以包含 ?），而剩余 是一个有限的资源值规范集合，其并集表示在计入已有后需求 中仍未满足的部分。当任一参数包含 ? 时，该函数必须针对该规范所表示的所有可能实例的并集进行推理。

AC4A 使用资源_差异函数，对需求集中的每项操作执行权限检查。它会维护一个尚未满足的资源值规范需求工作集。从所需的资源值规范开始，逐步减去每项已授予的资源值规范所涵盖的部分，直至剩余为空（需求已满足），或所有已授予的资源值规范均已用尽（需求未满足）。只有当每项所需操作对应的需求最终都为空时，才允许访问。算法 1 将此方法形式化，其中 *getResources* 返回与特定操作关联的资源值规范列表， *Need* 是某项操作所需的资源值规范集合 *Action* ，该操作来自所需操作集合 *NeedActions*，而 *HaveResources* 是有效权限中操作为 *Action*的已授予资源值规范集合。

算法1 假设应用程序提供的资源_差异函数是可靠的。这意味着它不会允许访问未获授权的资源。为实现这一点，该函数无需在扣除已授予的资源后，精确返回剩余资源。只要它返回的资源始终不超出最初所需的范围，该算法就仍然可靠。例如，如果某个代理需要访问 {*tag1*、*tag2*、*tag3*}，并且针对同一操作拥有资源 {*tag1*}，那么资源_差异函数可以返回 {*tag2* 、 *tag3*} （精确结果），也可以返回 {*tag1* 、 *tag2* 、 *tag3*} （保守结果）。这两种实现都是可靠的，因为它们不会授予超出所需范围的访问权限。

顺序无关性 是资源_差异函数的一项性质，确保无论按何种顺序处理已授予的资源，最终的权限检查判定都保持不变。

算法 1*AC4A*的权限检查算法

对于 $Action \in NeedActions$ **执行**
  $HaveResources \leftarrow \text{getResources}(ActivePermissions, Action)$
  $Remaining \leftarrow \{Need\}$
  对于 每个 *have* 在 $HaveResources$ **执行**
    $Next \leftarrow \emptyset$
    对于每个 n，在 $Remaining$ **执行**
      $Next \leftarrow Next \cup$资源_差集$(n, \ have)$
    结束循环
    $Remaining \leftarrow Next$
    如果 $Remaining = \emptyset$**则**
      跳出
    结束条件判断
  结束循环
  如果 $Remaining \neq \emptyset$**则**
    返回假
  结束条件判断
结束循环
返回真

如果资源_差异函数从不返回超出最初所需资源范围的资源，*AC4A* 可确保权限检查始终可靠，但这并不能保证顺序无关性。资源_差异函数无法返回精确的剩余资源集合的应用程序，必须确保该函数具有顺序无关性。例如，如果资源_差异函数能为日历区间资源类型节点精确计算区间减法，那么它就具有顺序无关性。如果该函数有时会过度估算覆盖范围（例如向上取整区间），那么已授予资源的不同处理顺序可能会产生不同的最终剩余 集合，从而破坏顺序无关性，尽管该算法仍然是可靠的。

应用程序开发者应确保其资源_差异实现无论算法 1中处理已授予资源的顺序如何，都能保持一致。基于集合的资源天然具备这一特性。层级资源和基于区间的资源则需要进行一致的细化处理，或采取保守的处理方式。应用程序可以自行选择精度与性能的权衡，但应确保其选择不会引入顺序依赖，并且保持可靠。*AC4A* 支持常见模式，例如保守地查找枚举、区间和树中剩余的资源，从而简化资源_差异函数的实现。这些辅助函数不会解读资源类型所表示的实际数据，只处理简单情况，例如需求是 已有资源 的子树，或者 需求与已有资源 的枚举值相同。对于重叠资源类型或非层级关系等更复杂的情况，应用程序可以自行实现资源_差异函数。例如，在图 1(c) 中，同一笔信用卡交易可以通过基于最近时间和基于时间的路径访问。资源_差异函数可以通过不同方式处理这种重叠：

- **保守的方法**：该函数可能会将新近性维度与基于时间的维度视为不可比较。通过基于新近性的资源获得访问，不会被视为满足对基于时间的资源的需求，而 resource_difference 可能会保守地返回对基于时间的资源的全部需求。
- **精确方法**：该函数可能会识别出，基于时间的访问涵盖了该月的所有交易。当代理同时拥有基于近期性的访问和基于时间的访问时

两者存在重叠，该函数可以计算出重叠数据已满足需求，并返回空集。

两种方法都有效，选择完全由应用程序决定。框架只要求资源_差异函数是可靠的，且不违反顺序无关性。

3.5 访问所需的充分权限

当代理执行访问操作（例如调用 API 端点或与网页元素交互）时，我们需要了解哪些权限足以支持该访问。掌握这些信息后，我们便使用资源_差异函数（第 3.4.1 节）来计算当前授予的权限是否足够。API 和网页所需的充分权限各有不同的确定方式。我们先介绍 API 的处理流程，再介绍网页的处理流程。

3.5.1 API。对于基于 API 的智能体，每个应用程序都会为其 API 端点提供一个 权限函数。权限函数接收端点名称和调用时的实际参数值，并返回一个有限权限集（资源值规范与操作对），这些权限足以支持该调用。图 4 展示了权限函数针对不同 API 调用返回的权限示例。

API 调用	**足够的权限**
获取_日历_事件（开始_日期：2024-01-01，结束_日期：2024-01-31）	年（2024年）::月（一月），读取
获取_事件（开始_时间：1696809600，结束_时间：1696896000）	时间间隔(1696809600-1696896000)，读取
读取_文件(路径: "/home/user/docs/report.txt")	目录(home)::目录(user):: 目录(docs)::文件(report.txt)，读取

图4. 权限函数为 API 调用返回的充分权限示例。

*AC4A*提供强制执行基础设施，但不了解应用程序的 API语义，也不知道给定端点实际访问哪些资源。权限函数可以像应用程序开发者所选择的那样精确或近似，而*AC4A* 完全不受这种选择的影响。例如，如果某个 API 调用访问 2026 年 1 月1日至2月15日的日历事件，权限函数可以返回恰好涵盖这 46 天的日级资源值规范，也可以将粒度向上取整到月级粒度，并返回 {Year(2026)::Month(January), Year(2026)::Month(February)}。

权限函数的正确性完全由应用程序负责。*AC4A*信任权限函数的输出，且无法独立验证其是否准确反映端点实际访问的资源。如果日历权限函数报告某个端点只需要一月的数据，而该端点实际上会读取一月至二月中旬的数据，*AC4A* 就会执行这一范围较窄（但不正确）的要求，端点可能会访问超出用户预期授权范围的资源。权限函数还必须处理未映射端点。一种保守的做法是要求宽泛权限，这样除非代理已获准广泛访问，否则未映射的调用都会被阻止。图5 展示了一个适用于日历应用程序的权限函数示例。

随着应用程序不断演进，可以扩展资源类型树，而不会破坏现有权限。例如，如果日历应用程序新增了管理工作时间的功能，就可以在*WorkingHours* 节点下添加一个*Day* 子节点，并更新权限函数，使其针对相关端点返回该节点。授予日历事件访问权限的现有权限仍会

```
定义 权限_函数(参数, 端点_名称):
    资源 = []
    开始_时间 = 参数['开始_时间']
    持续时间 = 参数['持续时间']
    end_时间 = 开始 _时间 + 持续时间
    月_名称 = ['一月', '二月', '三月', '四月', '五月', '六月',
               '七月', '八月', '九月', '十月', '十一月', '十二月']
    定义 获取_层级(时间_值):
        返回 [
            (365, '年', 时间_值。年),
            (日历。月份范围(时间_值。年, 时间_值。月)[1], '月',
                月份_名称[时间_值。月份 - 1]),
            (1, '日', 时间_值。日),
            (0, '小时', 时间_值。小时)
        ]
    用于 (_, 标签, 开始_值), (_, _, 结束_值)在 压缩包(
        获取_层级(开始_时间), 获取_层级(结束_时间)):
        资源.追加(f'日历: {标签}({开始_值})')
        如果 开始_值 ! = 结束 _值:
            中断
    操作 = (
        "创建"
        如果 任意(k位于 端点_名称 位于 k 位于 ["reserve", "创建", “添加”])
        else “写入”
        如果任意(k位于端点_名称对应的k 位于 [“更新”时, “编辑”, “修改”])
        else “读取”
    )
    返回 '::'.join(资源), 操作
```

图 5. 日历应用程序的权限函数示例。该函数接收端点名称和参数值，并返回资源值规范和一项操作。

继续有效。涵盖*Day*节点的权限会隐式涵盖新的*WorkingHours*子节点，因此现有的宽泛权限无需修改即可继续有效。

3.5.2 网页。对于基于浏览器的智能体，我们确保智能体可以查看并与某些 DOM 元素交互，但不能与其他元素交互。原则上，我们可以将每个 DOM 元素都视为一个 API 入口点，并设置一个权限函数来返回访问该元素所需的权限。然而，网页通常包含数千个 DOM 元素，因此分别为每个元素指定权限并不实际。

相反，我们分别将类似 CSS 的选择器映射到资源值规范和操作。每个网页的配置文件都包含一些条目，用于将选择器与资源值规范和操作关联起来。对于给定的 DOM 元素，会根据所有与该元素匹配的选择器计算所需权限。不匹配任何选择器的元素不需要权限，默认保持可见。这种做法切实可行，因为大多数网页元素（标题、标识、导航）都不包含受保护资源。

每当 DOM 发生变化（由于导航、JavaScript 执行或智能体操作），或代理获授的权限发生变化时，*AC4A* 都会根据当前 DOM 和有效权限重新评估配置。若代理的有效权限未涵盖某个元素所需的权限，该元素会在代理看到页面之前被隐藏。

最简单的形式是，配置文件将普通 CSS 选择器映射到资源值规范或操作。例如，日历页面可以将显示 2026

年六月事件的日历网格 #calendarGrid 映射到 Year(2026)::Month(六月)，也映射到读取操作。同样，它也可以将用于保存事件的按钮 #saveEventBtn 映射到 Year(?)，并映射到创建操作。

真实网页带来了两项复杂情况，因此需要超越普通 CSS 选择器。首先，许多网站会使用 JavaScript 动态生成 HTML，并随机化 ID 和类名以防止数据抓取，因此像 #calendarGrid 这样的简单选择器在不同会话中并不可靠。为了解决这个问题，配置文件使用扩展 CSS 选择器，支持基于稳定的元素属性和内容进行查询，例如 button[aria-label*='选择以更改月份'] 或 span:contains('新建活动')。

其次，单个 HTML 元素可能会根据动态页面内容表示不同的资源值。日历可能使用相同的元素结构和不同的数据来显示不同的月份，文件系统浏览器也可能使用相同的模板列出不同的目录。对于这些情况，必须在运行时根据页面内容计算资源值规范，使用 资源值字符串规范。图 6 展示了 Outlook 日历的完整配置，说明了扩展选择器和动态资源值字符串规范的用法。

```
{
    "https://outlook.live.com/calendar/0/view/workweek": {
        "已验证": true  ,
        "读取": [
          "div.ZlutZ"
        ],
        "写入": [
          "span:contains('保存')"
        ],
        "创建": [
          "span:contains('新建活动')"
        ],
        "数据": {
          "年($data{button[aria-label*='选择以更改月份'] > span}[1])::
          月($data{button[aria-label*='选择以更改月份'] > span}[0])::
          日(?)": [
                "div.ZlutZ"
          ],
          "年($data{input[aria-label='开始日期']}{split_slash}[2]@value)::
          月($data{input[aria-label='开始日期']}{split_slash}[0](卡号_至_月)@value)::
          日($data{input[aria-label='开始日期']}{split_slash}[1]@value)": [
            "span:contains('保存')"
          ],
          "年(?)": [
            "span:contains('新建活动')"
          ]
        }
    }
}
```

图 6. Outlook 日历的网页映射配置示例。静态映射将 CSS 选择器与操作相关联，而动态映射则使用 $data 语法，在运行时从页面内容中提取资源值。

资源值字符串规范。 资源值字符串规范会在运行时从页面内容中提取资源值。其语法如下：

$*data*{css selector}{list transformation}[索引]{value transformation}@attr

此语法会从指定的 CSS 选择器和属性（例如 value、href、src、alt、title 或 text）中提取数据。数据经列表转换处理后会变成一个字符串列表，然后由索引选取其中一个字符串；之后还可以通过可选的值转换进一步处理该字符串（写在索引后的括号中，例如 (卡号_转为_月)）。多个转换可以串联使用。

例如：

年($*data*{input[aria-label='开始日期']}{split_slash}[2]@value)

此选择器指向一个输入元素，其值类似于 MM/DD/YYYY。该值会按 / 字符拆分，索引为 2 的元素（年份）将成为“年”资源类型节点的值。

或者，网页开发者可以直接使用 HTML 属性为页面添加注释：使用 data-ac4a-resource 指定资源值规范，使用 data-ac4a-action 指定操作，使用 data-ac4a-static 指明资源值是否为动态值。当 data-ac4a-static 为 false 时，data-ac4a-resource 中的值应为资源值字符串规范，并在运行时进行求值。

4 已授予权限的用户界面

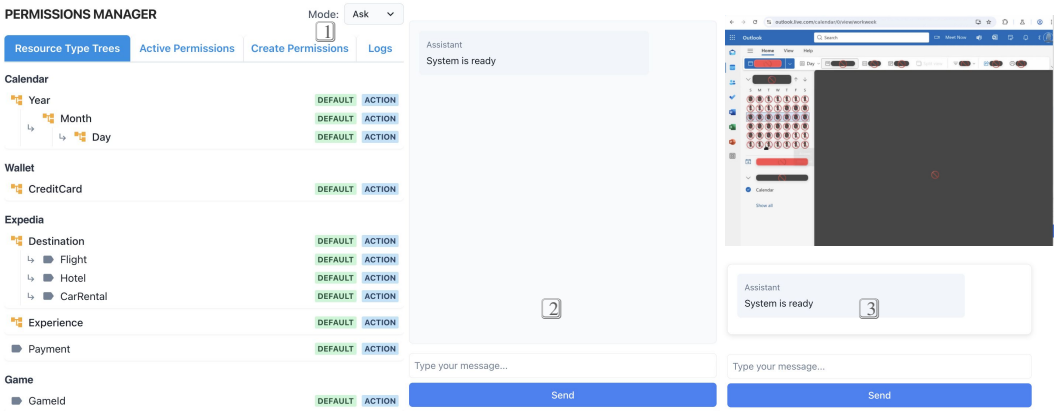


图 7. *AC4A* 界面。仪表板（左侧）提供多个选项卡，可用于浏览资源类型树、查看有效权限、创建权限和查看日志。(1) 权限处理模式选择器；(2) 基于 API 的智能体的聊天界面；(3) 基于浏览器的智能体的浏览器预览。

我们实现了一个 *AC4A* 原型，如图 7所示。该系统以一个由基于 AutoGen 的多智能体系统支持的聊天机器人为核心 [27]，可访问日历和旅行服务等应用程序端点。这些端点由基于大语言模型的模拟响应或真实 API 端点提供支持。每个应用程序都会提供其资源类型、操作、权限函数和资源_差异函数。基于 API 的智能体和基于浏览器的智能体同时执行同一组有效权限。

原型界面包括一个权限管理器、一个用于与基于 API 的智能体交互的聊天界面、一个实时浏览器视图，用于展示基于浏览器的智能体所看到的内容，以及可选的

完整网页视图（最终用户可见，但代理不可见）。对于这两种代理类型，我们都使用 GPT-5.4-mini 作为底层模型。对于浏览器代理，GPT-5.4-mini 会使用 PyAutoGUI 以 Python 代码的形式生成浏览器交互操作 [4]。

AC4A 提供了一个仪表板，用于创建、查看和移除有效权限。用户可以通过仪表板（图 8 ）手动创建权限：从应用程序的资源类型树中选择资源类型节点，提供相应的资源值规范，并选择一项操作。创建的权限会显示在有效权限视图中（图 10），该视图按资源值规范和操作组织成结构化层级。用户可以随时从此视图中移除权限。日志视图会记录权限创建、移除和使用的完整审计跟踪，包括哪些权限允许或阻止了每次访问。当访问被拒绝时，仪表板会显示资源差异函数（第 剩余 需求_产生的剩余需求（第 3.4.1节），指导用户添加哪些权限，以便重试时成功访问。

除了手动创建权限之外，*AC4A* 还支持两种额外的方式。大语言模型辅助生成（图9）可将自然语言描述转换为形式化权限。例如，用户可以输入 授予访问日历的权限，以便查看我七月的日程安排，系统便会根据应用程序的资源类型树生成相应的权限。用户可以在添加生成的权限之前进行审核，并酌情修改。系统还可以从用户请求或执行计划中推断权限：系统会分析任务描述中的隐式访问需求，并通过同一大语言模型辅助生成流程处理推断出的需求。此外，还可以将有效权限呈现为自然语言陈述（图 11），以便更易于人们理解。

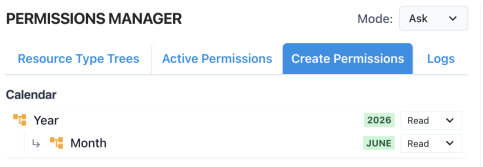


图 8. 手动创建权限。用户在仪表板中选择资源类型节点、值和操作。

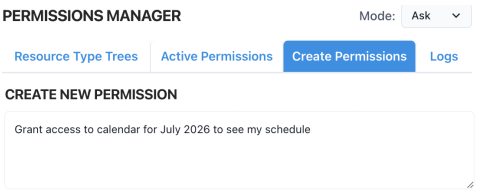


图 9. 借助大语言模型生成权限。自然语言描述被转换为正式权限。

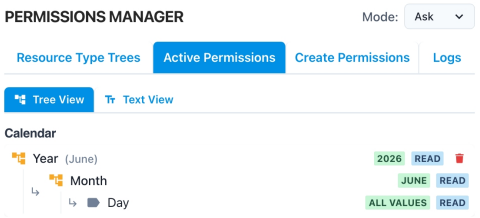


图 10. 在结构化的层级界面中显示有效权限，便于查看。

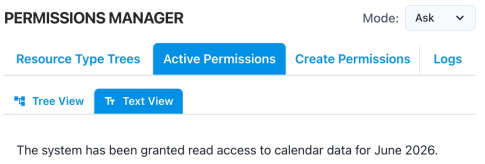


图 11. 将有效权限呈现为自然语言陈述。

系统还会使用资源_差异函数，在添加权限时检测冗余。例如，在日历应用程序中，如果已经拥有 Year(2026) 的读取权限，那么添加 Year(2026)::Month(October) 会被标记为冗余，因为授予某一年的权限就隐式授予了

对该年份所有月份的访问权限。然而，如果先创建了 October 权限，则这两项权限会同时存在。移除其中一项并不保证其涵盖的访问权限会被撤销，因为另一项权限可能仍涵盖该资源。用户有责任管理此类重叠。

4.1 权限处理模式

在正常运行期间，访问被拒是意料之中的情况。智能体可能会尝试执行超出其当前授权范围的操作，原因可能是用户尚未授予所需权限，也可能是智能体正在探索可用选项。*AC4A*提供了可配置的处理模式，用于设定智能体应如何响应，而不是将访问拒绝视为错误。

当权限检查算法确定智能体的有效权限不足并拒绝访问时，必须让智能体知晓这一情况，以便其做出恰当响应。对于基于浏览器的智能体，带有✗标记的受阻区域表示智能体缺少必要权限。对于基于 API 的智能体，系统会返回一条描述被拒绝访问的错误消息。*AC4A* 支持四种处理访问被拒情况的模式，可通过智能体的系统提示词进行配置。

(1) **询问：** 智能体请求用户授予必要权限；如果可以，还会告知用户具体是哪项访问被拒绝，以便用户自行配置权限。(2) **跳过：** 智能体提出一种避开未经授权资源的替代方案，或询问用户接下来该怎么做。(3) **推断：** 系统使用大语言模型自动推断所需权限，但未经用户批准不会启用这些权限。智能体会展示推断出的权限并等待确认。(4) **YOLO：** 系统使用大语言模型自动推断并启用权限。智能体会重试被拒绝的操作，直到系统自动设置好正确的权限。

无论采用哪种模式，所有生成并部署的权限及其使用情况都会被记录，以便允许或拒绝访问。用户可以随时监控正在创建和使用的权限，即使大语言模型自动生成权限，也能进行监督。

这些模式体现了安全性与便利性之间的权衡。手动创建权限并结合询问或跳过模式，是最安全的配置，因为用户可以完全掌控授予哪些权限。大语言模型辅助模式（推断和 YOLO）更加便利，但也会引入潜在的攻击途径，因为生成权限的大语言模型可能会受到提示词注入 [15, 16] 或其他技术的操纵。事实上，除了记录智能体添加了权限之外，YOLO 完全不安全。只有在审计足以满足需求时，或为了试验智能体在无需用户干预的情况下能够自行完成多少操作时，才适合使用此模式。

5 案例研究

我们通过两个案例研究展示 *AC4A*在基于 API 和基于浏览器的智能体中的访问控制执行。第一个案例研究使用井字棋游戏，展示 *AC4A*核心组件的端到端运行过程。第二个案例研究展示了在真实航班预订任务中的执行情况，该任务涉及 Outlook 日历、Expedia 和支付钱包等实际应用程序。图 12 展示了这些案例研究中所用应用程序的资源类型树。

5.1 设计演示：井字棋游戏示例

在本节中，我们将通过一个简单的井字棋游戏，演示*AC4A*的端到端运行过程。我们假设代理可以随意下井字棋。我们要保护的资源是先前对局历史，未经授权，代理不应能够查看或删除该资源。我们

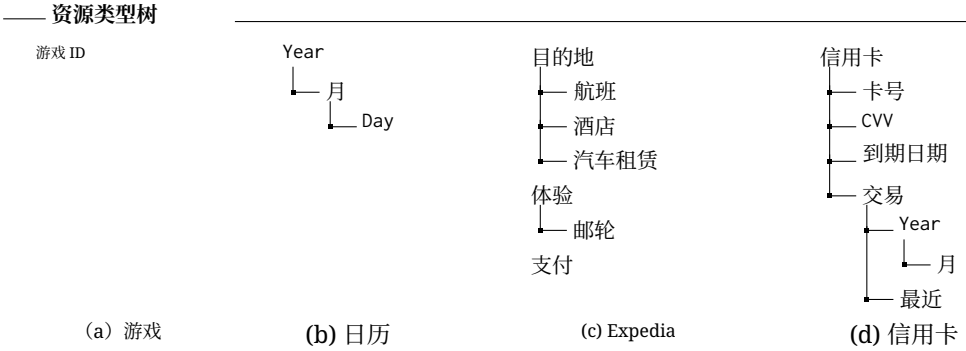


图12. 案例研究应用的资源类型树。(a) 井字棋游戏使用单个 GameId 节点。(b) 日历包含 Year、Month 和 Day 资源类型节点。(c) Expedia 使用三棵资源类型树对旅行进行建模，分别用于目的地、体验和支付。(d) 钱包应用中的信用卡资源类型树。

首先展示*AC4A* 如何为基于 API 的智能体实施访问控制，然后展示同样的保护如何适用于与游戏网页交互的基于浏览器的代理。

5.1.1 应用程序定义的组件。 要使任何网页或应用程序能够与 *AC4A* 配合使用，应用程序必须提供四个组件：一棵或多棵资源类型树、一组操作、一个资源_差异函数，以及一个权限函数。下面以井字棋游戏为例逐一介绍。

资源类型树。 我们希望对游戏历史数据实施访问控制，以防止代理在未获适当授权的情况下查看或删除这些数据。我们将资源类型树建模为单个根节点 *GameId*，如图 12(a) 所示。虽然我们可以创建更细粒度的子节点，例如 *GameStatus* 或 *GameResult*，以实现更精细的访问控制（例如，禁止访问玩家输掉的游戏），但我们采用简单模型，仅使用 *GameId*。

操作。我们为此应用程序定义了两个操作：读取，用于查看游戏数据；写入，用于删除游戏。

资源差异函数。该应用程序为_游戏 ID资源类型节点提供了差异函数。给定资源_差异(游戏 $ID(x)$，游戏 $ID(y)$)，如果 $y = ?$ 或 $y = x$，该函数将返回空集；否则返回单元素集合{游戏 $ID(x)$}。此实现是保守的：如果存在一个游戏，且用户已授予对它的访问权限，那么精确的实现会识别出游戏 $ID(?)$ 已满足，但我们的函数会返回完整的需求，因为它不会跟踪现有游戏的集合。这种保守性是可靠的（第3.4.1节），并且简化了实现。

权限函数（*API*）。权限函数将 API 端点映射到资源值规范和操作。考虑 get_games() 和 delete_game(*game_id*)。权限函数将 get_games() 映射到 *GameId*(?)，并赋予读取操作，因为它会返回所有游戏。对于 delete_game(*game_id*)，它会映射到 *GameId*(*game_id*)，其中 *game_id* 在运行时解析。例如，如果调用 delete_game 时使用的 ID 为 *45*，映射后的资源值规范将变为 *GameId*(“45”)，并赋予写入操作，因为删除会修改游戏数据。API 权限函数如图 13所示。

```
定义 生成_属性(自身,              kwargs, 端点_名称, 通配符）:
    游戏_标识 = kwargs。获取('游戏_标识',   '?')
    如果 通配符:
        游戏_标识 ='?'

    如果端点_名称 =='获取_游戏':
        资源 = f"Game:GameId(?)"
    否则如果端点_名称在 ['获取_游戏',    '删除_游戏']:
        资源 = f"Game:GameId({游戏_ID})"
    否则:
        资源 = f'游戏:游戏 ID({游戏_ID})'

    操作 =   '写入' 如果 '删除' 在 端点_名称否则          '读取'

    返回 {
        '资源      ': 资源     ,
        '操作': 操作
    }
```

图13. 井字棋游戏的 API权限函数，将端点及其参数映射到资源值规范和操作。

权限函数（网页）。对于游戏的 Web 界面，权限函数使用配置文件将 HTML 元素映射到资源值规范和操作。为简便起见，我们将游戏历史元素映射到 *GameId*(?)，并赋予读取操作；将删除按钮映射到*GameId*(?)，并赋予写入操作，而不是为单个游戏 ID 分别创建映射。网页元素映射配置如图 14所示。

```
{
    "读取":  [
        "#historyList"
    ],
    "写入":  [
        "button:contains('删除')"
    ],
    "游戏:GameId(?)":  [
        "#historyList",
        "button:contains('删除')"
    ]
}
```

图14. 井字棋游戏的网页元素映射配置，将 CSS 选择器与资源值规范和操作相关联。

5.1.2 基于 *API* 的代理强制执行。当代理调用 get_games() 时，*AC4A* 会拦截该调用，权限函数会返回足够的权限：游戏 *ID*(?)，并附带读取操作。权限检查算法使用资源_differencefunction 来计算剩余。没有有效权限时，剩余不为空，因此拒绝访问（图15）。

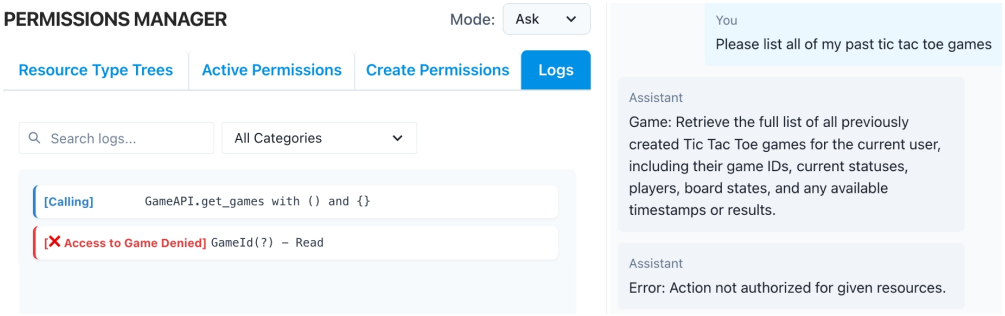


图15. 未启用任何权限。（左）API 访问日志显示，get_games 权限充足且访问遭拒。（右）聊天摘录显示返回给代理的权限错误。

授予读取权限后，get_games() 调用成功。但是，当代理尝试删除游戏时，*AC4A* 会拦截 delete_game()，权限函数会将写入权限作为足够的权限返回。没有有效的写入权限时，请求会被拒绝（图16）。当读取和写入权限均有效时，代理可以成功删除游戏（图17）。

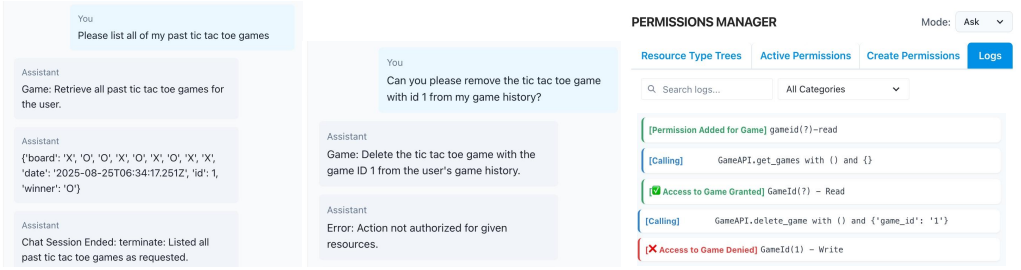


图16. 已启用只读权限。（左）聊天摘录：代理成功读取游戏数据。（中）聊天摘录：代理因缺少写入权限，无法删除游戏历史记录。（右）API 访问日志：get_games 调用成功，delete_game 遭拒。

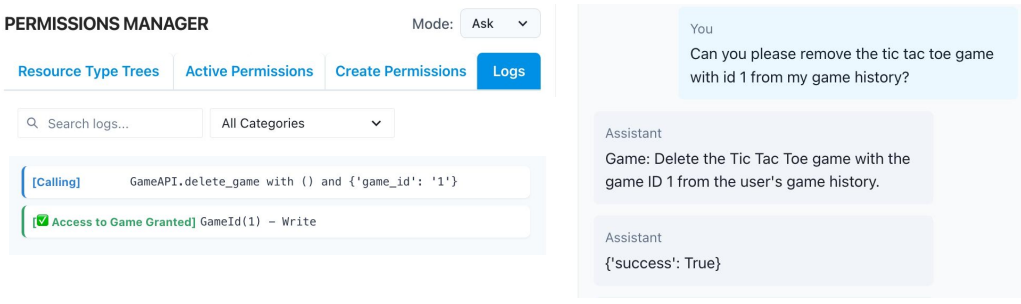


图17. 已启用读取和写入权限。（左）聊天摘录：代理成功删除游戏数据。（右）API 访问日志：delete_game 获准。

5.1.3 基于浏览器的智能体强制执行。 对于游戏的 Web界面，强制执行在 HTML 元素级别进行。当指示基于浏览器的智能体查找游戏历史记录时，它会打开游戏网页应用。 *AC4A*会在智能体访问页面之前拦截页面，并使用配置文件确定每个映射元素所需的充分权限。权限检查算法使用资源_差异函数计算*Remaining* 与有效权限之间的差异。对于 ✗不为空的元素，会使用显示

在没有有效权限的情况下，所有游戏历史元素都会被阻止（图18，右）。授予 *GameId*（?）的读取权限后，游戏历史记录变得可见，但由于不存在写入权限，删除按钮仍会被阻止（图19，左）。在读取和写入权限均有效时，删除按钮变为可访问（图19，右）。

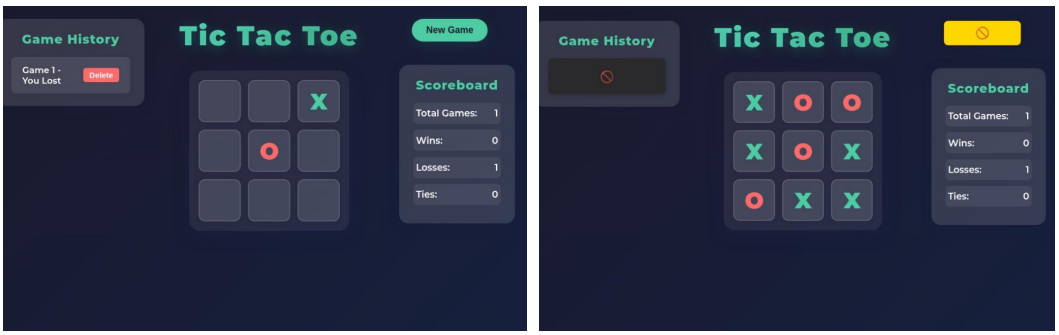


图 18。无有效权限。（左）显示所有元素的原始浏览器视图。（右）呈现给智能体的视图，游戏历史元素被实体覆盖层阻挡。

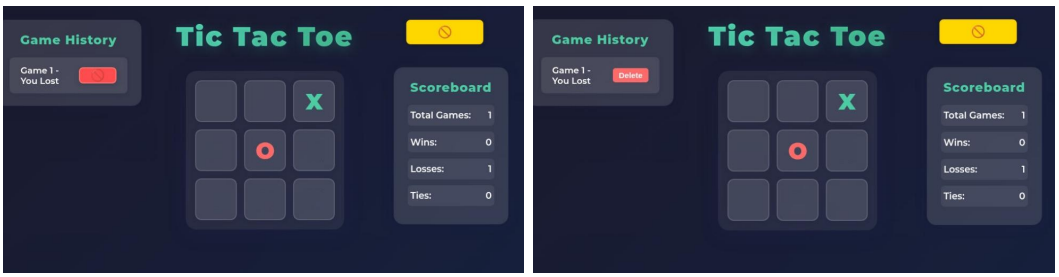


图 19.（左）只读权限：可以查看游戏历史记录，但删除按钮不可用。（右）读取和写入权限：删除按钮变为可用。

此示例展示了同时包含 API 和网页的 *AC4A* 。现实世界中的应用程序更为复杂，其中多个元素与各种资源类型相连接。我们已将此技术扩展到 Outlook 日历和 Expedia 等实际用例。

5.2 FlightBooking 演示

我们通过一个真实的多步骤航班预订任务来演示*AC4A*，该任务涉及三个应用程序：日历（Outlook）、旅行服务（Expedia）和支付钱包。我们将任务拆解为多个子任务，以测试访问控制边界。从*AC4A* 的角度来看，意外越权和恶意越权无法区分，因此访问控制层会以相同方式处理。

演示从未授予任何有效权限开始，我们展示了如何使用相同的底层权限，对基于 API 和基于浏览器的智能体实施管控。

每个应用程序都提供 *AC4A* （第 5.1 节）所需的四个组件。日历使用图 1（a）中的资源类型树，其中包含年、月和日资源类型节点，并定义读取、写入和创建操作。其权限函数将读取和空闲时间检查端点映射到读取权限，并将预订端点映射到创建权限，根据 开始_时间 和 持续时间 参数推导资源值规范。Expedia 的资源类型树如图 12（b）所示。其权限函数将搜索和信息端点映射为对航班数据的读取权限，并将预订端点映射为对正在预订的特定航班的创建权限（例如，Flight（“DL 1847”））。电子钱包定义了多个资源类型树，其中之一是图 1（c）中的信用卡树。电子钱包的权限函数将查询端点映射到读取权限。

5.2.1 基于 *API* 的智能体执行。

场景 *A*（未经授权的日历读取）。 用户要求代理查找 2026年6月的日历可用性。*AC4A* 拦截生成的 API 调用，权限函数返回所需权限（Year(2026)::Month(六月)，读取）。由于没有任何有效权限涵盖此需求，权限检查算法生成了非空的 *Remaining* 集合，因此调用被阻止。用户通过仪表板授予适当权限后，代理便成功重试了相同的调用。该场景的聊天摘录和 *AC4A*访问日志见图 20。

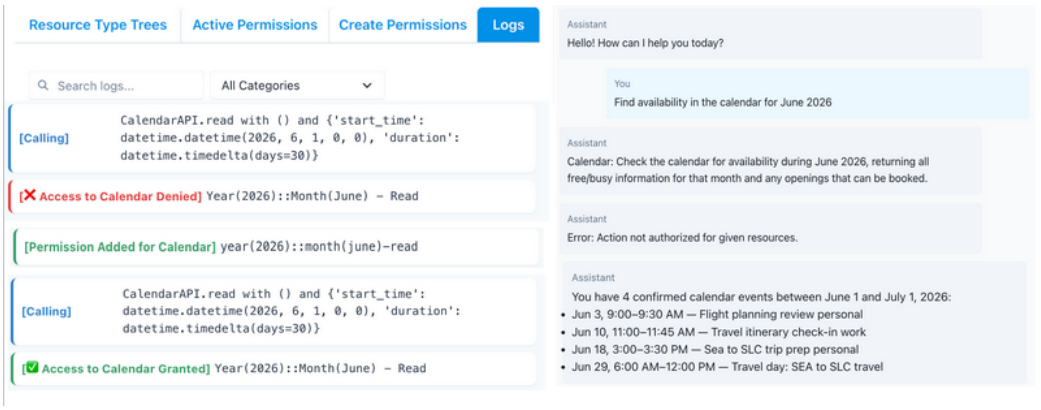


图20。A 场景 A 的 PI 型代理强制执行：2026年6月的日历读取先被阻止，随后获准 .

场景 *B*（范围与操作边界）。在2026年6月的读取权限现已生效后，本场景将检验两个彼此独立的边界：资源范围和操作类型。首先，用户请求查询2026年7月的空闲时间。Thepermissionfunction 返回 Year(2026)::Month(July) 作为所需资源，但权限检查算法判定，针对 Month(June) 的有效权限不涵盖 Month(July)，因此该调用被阻止。其次，创建操作（添加2026年6月29日的事件）也被阻止，因为有效权限仅授予读取权限，而不包括创建权限。这两种情况如图21所示。

场景 *C*（航班预订与跨应用访问）。用户要求代理根据检索到的日历可用性，预订 2026年6月29日的航班。用户已授予代理读取所有航班数据的权限，因此代理可以搜索并查看航班选项。当用户要求代理预订特定航班（例如 DL 1847）时，代理尝试创建预订。权限

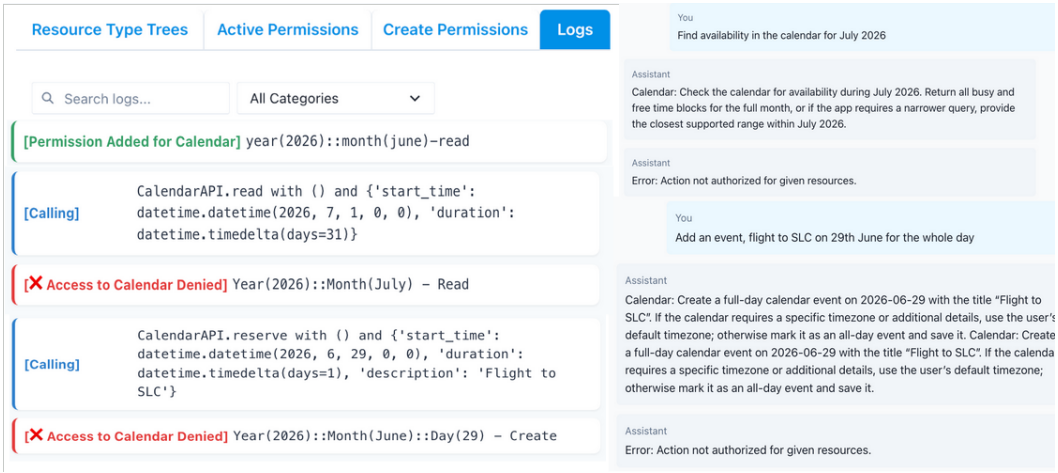


图21。场景 B：2026年7月的读取被阻止（权限仅涵盖六月）；2026年6月29日的创建被阻止（只读权限）。

函数返回了足够的权限（航班（“DL 1847”），创建），但没有任何有效权限涵盖该操作，因此请求被拒绝。用户通过仪表板为该航班添加创建权限后，代理重试，预订顺利完成。随后，代理尝试从电子钱包中获取信用卡信息以进行支付。*AC4A* 拦截了该调用；权限函数返回了信用卡资源所需的足够权限，但权限检查算法发现没有有效的读取权限涵盖该操作，因此请求被拒绝。图22 展示了此场景中的聊天摘录和访问日志。

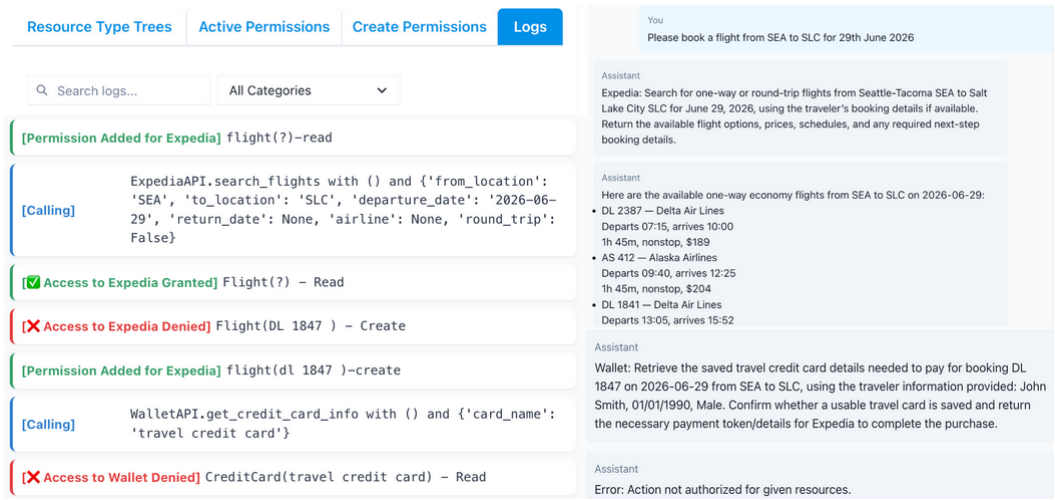


图22。场景 C：允许读取航班；在添加创建权限之前，创建 DL 1847 的预订被阻止；信用卡读取遭拒。

在上述场景中，我们手动触发了本应被阻止的请求（例如，在只允许访问六月的情况下请求访问七月），以演示*AC4A*如何处理未经授权的访问

尝试，无论是由代理错误还是恶意意图导致的。我们还通过仪表板手动创建了权限。实际上，也可以根据自然语言描述生成权限，或使用第4.1节所述的处理模式自动推断权限。

5.2.2 基于浏览器的代理强制执行。 我们使用基于浏览器的智能体复现相同的工作流程，以展示 *AC4A* 在用户界面层面强制执行相同的权限。*AC4A* 不会拦截 API 负载，而是注入 CSS 覆盖层，在智能体访问未经授权的 DOM 元素之前，以视觉方式将其屏蔽。

场景 *A*（日历屏蔽与增量授权）。指示智能体打开以下网址的日历：*outlook.live.com/calendar*。在智能体访问页面之前，*AC4A* 会拦截该页面，配置文件则会确定每个映射的用户界面元素所需的权限。在没有有效权限时，权限检查算法会将所有日历元素判定为未经授权，并注入 CSS 以屏蔽所有映射的资源（图 23a）。智能体报告称无法查看日程安排。用户授予 2026 年 6 月的读取权限后，日历网格便会显示，智能体也会解析空闲时间（图23b）。

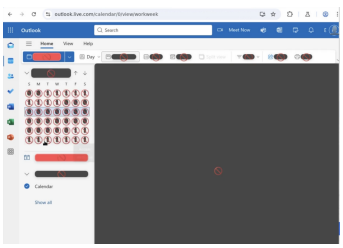
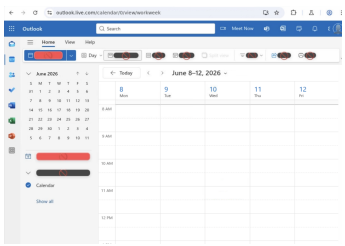
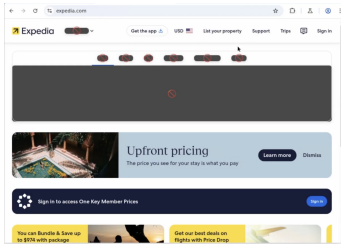
随后，我们测试另外两个边界情况。首先，请求查看 2026 年 7 月的日历时，日历仍会被屏蔽，因为有效权限仅涵盖 6 月。其次，请求在 2026 年 6 月 29 日添加事件时，事件创建界面（包括保存按钮）仍会被屏蔽，因为有效权限仅授予读取权限，而未授予创建权限。

场景 *B*（*Expedia* 航班预订）。代理前往 Expedia 预订航班（SEA 至 SLC，2026年6月29日）。由于没有 Expedia 权限，航班搜索区域在视觉上被屏蔽（图 23c）。用户授予航班资源的读取权限后，航班内容和搜索控件变为可见（图 23e），而授权范围之外的元素（例如租车弹出窗口）仍被屏蔽，因为该权限仅涵盖航班资源（图 23d）。代理选择航班（例如 DL 1847）并尝试创建预订时，该操作会被阻止，直到用户授予针对该航班的创建权限。预订继续进行后，代理尝试获取用于支付的信用卡信息；在用户授予对信用卡资源的读取权限之前，该访问会被阻止（图23f）。

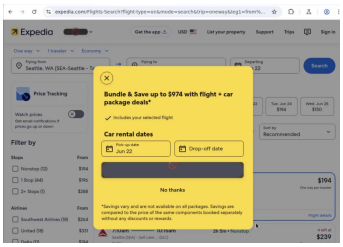
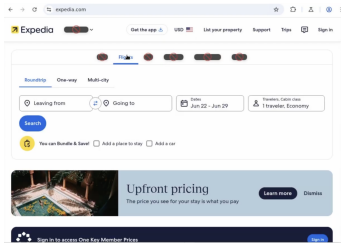
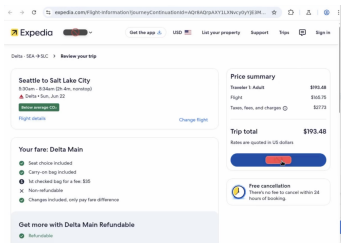
API 和浏览器场景共同表明，*AC4A*能够成功实施访问控制，无论代理是通过 API 调用还是前端 DOM 操作进行交互，且两种情况下都使用相同的权限表示和同一组有效权限。

6 局限性与未来工作

映射文件要求与维护。 必须编写并维护 API 的权限函数以及网页的配置文件。对于 API，最适合提供权限函数的是应用程序开发者，因为他们了解每个端点的语义。对于网页，配置文件也可以由第三方甚至通过社区协作开发，因为无需服务器访问权限即可为在线 Web 应用程序提供配置。例如，我们无需 Microsoft 参与，就成功为 Outlook 日历创建了可用的配置。不过，第三方配置很脆弱。如果应用程序更改了 DOM 的表示方式，配置就可能过时。编写这些映射可能很有挑战性，而且需要持续维护，才能与不断演变的网页和 API 端点保持一致。映射中的错误可能导致权限执行过度限制（阻止合法访问）或限制不足（允许未经授权的访问）。我们认为这是 *AC4A* 在实际应用中最显著的局限。开发用于验证权限函数和配置文件的工具，并为编写正确的映射提供指导，是未来工作的重要方向。

(a) 未授予任何权限的 Outlook 日历。(b) 授予六月和 2026 年的读取权限后。现在可以看到周视图。(c) 未授予任何权限的 Expedia。日历网格和日期单元格均被屏蔽。搜索区域被屏蔽。

(d) 即使授予了航班读取权限，汽车租赁弹出窗口仍被屏蔽。

(e) 之后 授予 航班 读取。搜索表单和航班结果均可见。

(f) 授予航班的创建权限后，即可进行预订；在用户授予读取权限之前，信用卡读取操作会被阻止。

图23. 基于浏览器的代理强制执行。*AC4A* 会注入 CSS 覆盖层，在代理访问未授权元素之前将其阻止。

浏览器映射定义的复杂性。 当前浏览器映射语言的表达能力足以处理我们已测试的应用程序，但可以通过支持更多编程结构来加以改进，例如依赖页面内容的条件语句，以及遍历动态生成的子元素。我们计划在未来工作中增强映射语言的表达能力。

复杂访问控制需求建模。 某些访问控制需求难以用当前框架表达。例如，若要仅在金额低于 200 美元时才允许结账访问，就需要将结账 和 金额低于 *200* 美元的结账 建模为独立的资源类型节点。未来工作中，我们计划为权限添加条件语句，使其能够比较资源值，从而支持诸如访问金额低于 *200* 美元的结账这样的权限。

寻找合适的权限。 *AC4A* 只专注于强制执行权限，而不负责确定哪些权限才是合适的。虽然它提供了生成权限的方法（手动生成、通过大语言模型生成，或从用户输入中推断），但对于给定任务而言，什么才算是合适的权限，仍是一个悬而未决的问题。这种分离使 *AC4A* 成为评估权限生成方法的实用试验平台。例如，用户驱动的访问控制 [21] 会从用户意图中推断隐式权限。*AC4A* 可以通过检查生成的权限是否能正确允许预期的访问并阻止非预期的访问，来评估这类方法的有效性。

权限生命周期管理。 *AC4A*中的权限会一直保留，直到被明确移除。这会带来风险。用户可能授予代理针对某个日历日期的写入权限，以修改一项事件，而

代理之后可能会利用相同权限，删除该日期的另一项事件。使用一次后即过期的单次使用权限可以缓解这一问题，但也会引发一些疑问：由谁决定权限何时过期，以及如何处理多个权限覆盖重叠资源的情况。权限生命周期管理属于访问控制策略的范畴，而*AC4A* 目前尚未对此作出规定。

7 相关工作

智能体给访问控制带来了新的挑战。现有技术影响了*AC4A* 的设计，但并不适用于此场景。我们首先讨论传统访问控制系统，接着讨论智能体遏制和信息流控制，最后介绍专为智能体设计的访问控制。

传统访问控制系统。传统访问控制历经多种模型的发展 [10]。Unix 系统通过权限位（用户/组/其他人的读取/写入/执行权限）确立了基于文件的访问控制 [20, 22]。*AC4A*的资源类型树直接借鉴了 Unix 文件层级结构，将层次化的资源组织与应用程序定义的操作相结合。与 Unix 不同，在 Unix 中，用户可能对父目录有权限，却无权访问子文件；而*AC4A*目前确保对父资源类型节点拥有权限，即意味着对所有子节点也拥有权限。

移动平台和网络浏览器通过声明式权限扩展了 Unix 模型，这类权限会预先指定所需能力 [18]。Android 引入了基于清单的声明，并为危险权限设置保护级别和运行时同意 [5]。网络浏览器则通过明确的用户同意，对地理位置、摄像头和麦克风等能力实施基于源的权限范围限定 [3, 17]。与这些系统一样，*AC4A* 会计算所需的权限。与它们不同的是，AC4A 会在运行时根据智能体的实际交互进行计算，而不是依赖静态声明。

基于角色的访问控制（RBAC）通过角色定义来组织权限，并支持层级继承和职责分离 [8]。基于属性的访问控制（ABAC）则根据主体、对象和环境属性动态评估权限 [11]。*AC4A* 的权限模型采用层级资源（对父资源类型节点的权限涵盖其所有子节点），并进行动态评估（与 ABAC 一样，根据实际参数值在运行时计算所需权限）。

Zanzibar [19] 通过将所有权限表示为全局图中的关系元组，在 Google 的各项服务中引入了统一的授权系统（基于关系的访问控制）。正如 Zanzibar 将访问控制与各个微服务解耦，以提供通用词汇体系，*AC4A*也将访问控制与各个智能体实现解耦。然而，Zanzibar 通过遍历用户与对象之间的关系图来评估权限，而*AC4A* 则根据层级资源类型树评估智能体交互；这种结构适用于 DOM 和基于 API 的智能体。

智能体遏制与信息流控制。一些框架通过隔离不可信的外部输入，来缓解间接提示注入和未授权操作。IsolateGPT [28]在独立的系统级沙箱中运行各个工具，并通过中央规划器协调跨应用交互。F-Secure [26] 将智能体拆分为上下文感知管线，由安全监视器在不可信输入进入规划过程之前对其进行过滤。这些系统提供执行隔离，但不会管理智能体获准访问哪些资源的细粒度权限。

另一条研究路线通过追踪并分离控制流与数据流来保护智能体。CAMEL [7]采用双大语言模型架构和污点跟踪，确保不可信数据无法劫持控制流。ACE [14] 将规划拆分为抽象阶段与具体阶段，并使用静态分析验证执行计划是否符合信息流约束。FIDES [6] 为

所有经过处理的数据附加机密性和完整性标签，并引入原语以选择性地隐藏或检查污染变量。

这些方法将智能体安全视为数据遏制问题，通过追踪不可信数据在智能体中的流动，防止其影响可信操作。*AC4A*无法直接缓解这些威胁，但访问控制是实现其他安全机制不可或缺的组成部分。

人工智能智能体的访问控制。 Progent [23] 和 Conseca [24] 在为智能体专门提供访问控制方面与 *AC4A* 最为接近。它们在策略的定义方式上有所不同。Progent 通过一种基于 JSON 的领域特定语言，在工具调用层级实施最小权限原则，使开发者能够根据工具参数指定允许、禁止或有条件允许使用哪些工具（例如，将电子邮件工具限制为只能联系特定收件人）。确定性运行时会根据该策略评估每次工具调用。Conseca 采用上下文感知方法。独立的策略生成器大语言模型仅根据可信上下文（用户意图、系统状态），为每个请求生成任务专属的安全策略。不可信上下文（例如网页内容）与策略生成过程隔离，而确定性的策略执行器会在执行前，根据这一临时策略评估智能体的工具调用。

AC4A 与二者的不同之处在于，它对资源 而非工具调用或任务上下文进行权限建模。虽然 Progent 可以限制工具参数（例如限制电子邮件收件人），但这些限制针对各个工具分别设定。*AC4A* 在资源本身上定义一次权限，无论通过何种方式访问该资源，都会统一执行该权限。

8 结论

传统软件系统将访问控制视为基础设施。很难想象部署一套没有文件权限的多用户操作系统，或一个没有身份验证的 Web 应用程序。然而，如今的大语言模型智能体能够自主调用 API 并代表用户浏览网页，却没有类似的机制。本文介绍了*AC4A*，这是一个为智能体提供细粒度、以资源为中心的访问控制框架。*AC4A* 背后的关键见解是，权限应针对资源 而非工具调用进行定义；将资源表示为层级类型，能够形成一种可在基于 API 和基于浏览器的智能体中统一执行的权限模型。我们在 Outlook 日历和 Expedia 等真实应用程序上开展的案例研究表明，智能体可以在受限权限范围内运行，同时避免超越其授权范围。

AC4A 有意将范围限定在执行层。它确保给定的一组权限得到遵守，但不会规定这些权限应该是什么。从实践角度看，这种区分既是务实的设计选择，也带来了研究机遇。正如我们的案例研究所展示的，*AC4A* 如今即可部署，只需手动编写权限函数和配置文件。与此同时，这种区分也为围绕执行层构建的生态系统开启了新的研究方向：对于一项任务而言，怎样的一组权限才算良好？权限应如何随会话进程演变？应用程序接口与资源类型之间的映射，能否通过学习或众包获得，而不必手动编写？我们认为，正如*AC4A* 所做的那样，建立一个有原则的执行基础，是有意义地探究这些问题的先决条件；正如访问控制框架为数十年来有关访问控制策略、审计和权限分离的研究奠定了基础。

参考文献

[1] Cursor - AI 代码编辑器 — cursor.com。 https://cursor。com/en。 [访问日期：03-07-2025]。 [2] Operator 震撼登场 — openai.com。 https://openai。com/index/introducing-operator/。 [访问日期：03-07-2025]。

[3] 权限 API - Web API | MDN — developer.mozilla.org。 https://developer.mozilla.org/en-US/docs/Web/API/ Permissions_API。 [访问日期：04-07-2025]。[4] 欢迎阅读 PyAutoGUI 文档！— PyAutoGUI 文档 — pyautogui.readthedocs.io。 https://pyautogui.readthedocs.io/en/latest/。 [访问日期：18-03-2026]。[5] Kathy Wain Yee Au、Yi Fan Zhou、Zhen Huang 和 David Lie。《PScout：分析 Android 权限规范》。载于《2012 年 ACM 计算机与通信安全会议论文集》，第 217–228 页，2012 年。[6] Manuel Costa、Boris Köpf、Aashish Kolluri、Andrew Paverd、Mark Russinovich、Ahmed Salem、Shruti Tople、Lukas Wutschitz 和 Santiago Zanella-Béguelin。《利用信息流控制保障人工智能智能体的安全》。arXiv 预印本 arXiv:2505.23643，2025年。[7] Edoardo Debenedetti、Ilia Shumailov、Tianqi Fan、Jamie Hayes、Nicholas Carlini、Daniel Fabian、Christoph Kern、Chongyang Shi、Andreas Terzis 和 Florian Tramèr。《从设计层面击败提示注入》。arXiv 预印本 arXiv:2503.18813，2025年。[8] David Ferraiolo、Janet Cugini、D Richard Kuhn 等。《基于角色的访问控制（RBAC）：特性与动机》。载于《第 11 届年度计算机安全应用会议论文集》，第 241–48 页，1995 年。[9] Hongliang He、Wenlin Yao、Kaixin Ma、Wenhao Yu、Yong Dai、Hongming Zhang、Zhenzhong Lan 和 Dong Yu。《WebVoyager：利用大型多模态模型构建端到端网页智能体》，2024年。[10] Pavlo Hlushchenko 和 Valerii Dudykevych。《访问控制范式与策略管理引擎探索性综述》。《CMIS 2024 计算机建模与智能系统》，第 263–279 页，2024年。[11] Vincent C Hu、D Richard Kuhn、David F Ferraiolo 和 Jeffrey Voas。《基于属性的访问控制》。《Computer》，48(2):85–88，2015年。[12] Lei Huang、Weijiang Yu、Weitao Ma、Weihong Zhong、Zhangyin Feng、Haotian Wang、Qianglong Chen、Weihua Peng、Xiaocheng Feng、Bing Qin 和 Ting Liu。《大语言模型幻觉综述：原理、分类、挑战与开放问题》。《ACM 信息系统汇刊》，43(2):1–55，2025年1月。[13] Sangyeop Kim、Yohan Lee、Yongwoo Song 和 Kimin Lee。《多样本攻击中真正重要的是什么？大语言模型长上下文漏洞的实证研究》，2025年。[14] Evan Li、Tushin Mallick、Evan Rose、William Robertson、Alina Oprea 和 Cristina Nita-Rotaru。《ACE：大语言模型集成应用系统的安全架构》，2025年。[15] Xiaogeng Liu、Zhiyuan Yu、Yizhe Zhang、Ning Zhang 和 Chaowei Xiao。《针对大语言模型的自动化通用提示注入攻击》，2024年。[16] Yi Liu、Gelei Deng、Yuekang Li、Kailong Wang、Zihao Wang、Xiaofeng Wang、Tianwei Zhang、Yepang Liu、Haoyu Wang、Yan Zheng 和 Yang Liu。《针对大语言模型集成应用的提示注入攻击》，2024年。[17] Tongbo Luo 和 Wenliang Du。《Contego：网络浏览器的基于能力的访问控制（短文）》。载于《国际信任与可信计算会议》，第 231–238 页。施普林格，2011 年。[18] Sardasht Mahmood、Bakhtiar Amen 和 Rebwar M Nabi。《移动应用安全平台综述》。《国际计算机应用期刊》，133(2):40–46，2016年。[19] Ruoming Pang、Ramon Caceres、Mike Burrows、Zhifeng Chen、Pratik Dave、Nathan Germer、Alexander Golynski、Kevin Graney、Nina Kang、Lea Kissner 等。《Zanzibar：Google 的一致性全球授权系统》。载于《2019 年 USENIX 年度技术会议（USENIX ATC 19）》，第 33–46 页，2019 年。[20] Kristal T Pollack 和 Scott A Brandt。《分布式分层文件系统的高效访问控制》。载于《第 22 届 IEEE/第 13 届 NASA 戈达德大容量存储系统与技术会议（MSST'05）》，第 253–260 页。IEEE，2005年。[21] Franziska Roesner、Tadayoshi Kohno、Alexander Moshchuk、Bryan Parno、Helen J. Wang 和 Crispin Cowan。《以用户为导向的访问控制：重新思考现代操作系统中的权限授予》。载于《2012 年 IEEE 安全与隐私研讨会》，第 224–238 页，2012 年。[22] Charles Salemi、Suryakanta Shah 和 Eric Lund。《Unix System V Release 4 操作系统的特权机制》。载于《USENIX 1992 年夏季技术会议（USENIX 1992 年夏季技术会议）》，1992 年。[23] Tianneng Shi、Jingxuan He、Zhun Wang、Linyu Wu、Hongwei Li、Wenbo Guo 和 Dawn Song。《Progent：面向大语言模型智能体的可编程特权控制》。arXiv 预印本 arXiv:2504.11703，2025年。[24] Lillian Tsai 和 Eugene Bagdasarian。《情境化智能体安全：为每种目的制定策略》。载于《操作系统热点主题研讨会论文集》，HOTOS '25，第 8–17 页。ACM，2025年5月。[25] Bibek Upadhayay、Vahid Behzadan 和 Amin Karbasi。《认知过载攻击：针对长上下文的提示注入》，2024年。[26] Fangzhou Wu、Ethan Cecchetti 和 Chaowei Xiao。《针对间接提示注入攻击的系统级防御：信息流控制视角》，2024年。[27] Qingyun Wu、Gagan Bansal、Jieyu Zhang、Yiran Wu、Beibin Li、Erkang Zhu、Li Jiang、Xiaoyun Zhang、Shaokun Zhang、Jiale Liu 等。《AutoGen：通过多智能体对话实现新一代大语言模型应用》。arXiv 预印本 arXiv:2308.08155，2023年。

[28] Yuhao Wu、Franziska Roesner、Tadayoshi Kohno、Ning Zhang 和 Umar Iqbal。Isolategpt：一种适用于基于大语言模型的智能体系统的执行隔离架构。*arXiv* 预印本 *arXiv:2403.04960*，2024年。[29] Zhiheng Xi、Wenxiang Chen、Xin Guo、Wei He、Yiwen Ding、Boyang Hong、Ming Zhang、Junzhe Wang、Senjie Jin、Enyu Zhou、Rui Zheng、Xiaoran Fan、Xiao Wang、Limao Xiong、Yuhao Zhou、Weiran Wang、Changhao Jiang、Yicheng Zou、Xiangyang Liu、Zhangyue Yin、Shihan Dou、Rongxiang Weng、Wensen Cheng、Qi Zhang、Wenjuan Qin、Yongyan Zheng、Xipeng Qiu、Xuanjing Huang 和 Tao Gui。基于大语言模型的智能体的兴起与潜力：一篇综述，2023年。[30] Tianbao Xie、Danyang Zhang、Jixuan Chen、Xiaochuan Li、Siheng Zhao、Ruisheng Cao、Toh J Hua、Zhoujun Cheng、Dongchan Shin、Fangyu Lei 等。Osworld：在真实计算机环境中对开放式任务的多模态智能体进行基准测试。神经信息处理系统进展，第37卷：52040–52094，2024年。[31] Baolei Zhang、Haoran Xin、Minghong Fang、Zhuqing Liu、Biao Yi、Tong Li 和 Zheli Liu。检索增强生成中投毒攻击的溯源，2025年。[32] Wei Zou、Runpeng Geng、Binghui Wang 和 Jinyuan Jia。Poisonedrag：针对大语言模型检索增强生成的知识污染攻击。2024年。