组合崩塌：多智能体编排框架如何丢失发起主体的权限

匿名提交

摘要

多智能体编排框架（CrewAI、LangGraph、AutoGen）允许一个智能体把工作交给另一个智能体。本文表明，这种组合会静默破坏一个单个智能体在隔离状态下满足的安全性质：一次操作不应以高于其发起主体的权限执行。当请求跨过组合边界时，发起者权限没有以受保护资源可使用的形式继续传播；因此资源自身的访问控制（RBAC、文件系统 ACL、OAuth scope）仍然正确执行，却是对代理者而不是对发起者执行。我们称之为**主体替换**。低权限请求 → 交接 → 高权限操作这一底层结构，已经作为默认形态出现在 ServiceNow Now Assist 等企业平台中。

框架已经尝试修复该问题，并提供了身份原语：AutoGen 的 `source`、CrewAI 的 `fingerprint`、LangGraph 的 `config.configurable`、A2A 的传输身份。但没有一个真正成立：跨组合传播发起者权限需要一个同时满足起源性、不可伪造性、可达决策点和权限衰减的通道，而这些原语各自只保护了不同的真子集。因此，有身份字段并不意味着安全。我们称这种现象为**虚假保证**。它不同于 1988 年的经典 confused deputy：在那里还没有人尝试提供一个身份修复字段，也就没有看似可信的字段可供开发者误信。本文把每个性质转化为运行时判定，用去除 LLM 混杂因素的确定性差分测试，在真实授权后端和已执行的正控实现上验证，并在一个预注册、此前未见的第四个框架上确认预测。我们把这些检查打包为框架无关的一致性测试，而不是提出新的委托协议。

1. 引言

1.1 一个动机事件

2025 年 11 月，AppOmni 披露了 ServiceNow Now Assist agents 中的一条二阶提示注入链 [3, 55]。低权限的 Workflow Triage agent 接收攻击者可影响的请求，并通过 Now Assist 默认的 agent-to-agent discovery，把派生任务交给更高权限的 Data Retrieval agent，后者执行任务并导出敏感记录；即使 ServiceNow 内置提示注入保护已启用，链条仍然成立。

这个事件适合作为起点有两点原因。第一，数据检索 agent 的动作是忠实授权的：它用自己的合法权限运行，没有绕过 guardrail；但操作源自一个本不应能提出该请求的主体。第二，厂商并没有把它当成一个要修的 bug；ServiceNow 确认该行为是预期行为并更新文档，而不是发布修复。披露者也把该链称作由某些默认配置选项定义出的 expected behavior，而不是 AI 本身的 bug [3, 55]；agent 发布后默认可被发现并自动组队。漏洞存在于框架如何组合本来合理的 agent，而不在某个单独 agent 的逻辑。

这并非单一产品的问题。同样的低权限请求 → 跨 agent 交接 → 高权限操作结构，已经作为默认能力出现在主流企业平台中，例如 Microsoft Copilot Studio 的 agent 节点和 Salesforce Agentforce 的 topic delegation；随着采用增长，这个缺口也会被默认激活（§2.3 和 §7.3 会回到部署广度和厂商表述的不变量）。问题不是这种结构是否已经部署，而是使它安全的通道是否已经部署；在我们研究的框架中，答案是否定的。

1.2 增加身份字段并不能弥合缺口

上述事件是多智能体编排结构性性质的一个症状 [15, 49, 59, 61, 64]。当 agent A 委托给 agent B 时，请求跨过框架边界（handoff、delegation call、shared-state update 或序列化消息）。如果下游受保护资源要对正确主体执行访问控制，提出该工作的权限必须随请求一起传播（图 1）。

主流框架没有携带这样的通道。发起主体常被降级为自然语言文本或建议性的标识符，而下游授权不消费这些内容；框架也没有提供运行时设置、完整性保护、起源绑定的原生通道。因此，受保护资源的原生访问控制仍然正确执行，却针对**代理者**主体，而不是**发起者**主体。一个在隔离状态下可证明安全的 agent，仅仅因为被组合起来就变得不安全。

自然的反驳是：这不过是在重述经典 confused deputy [17]，所以显而易见且早已解决。事实并非如此。最有力的证据是：**框架已经尝试修复它，但修复并不成立**。现代编排器已经提供了直指这个问题的身

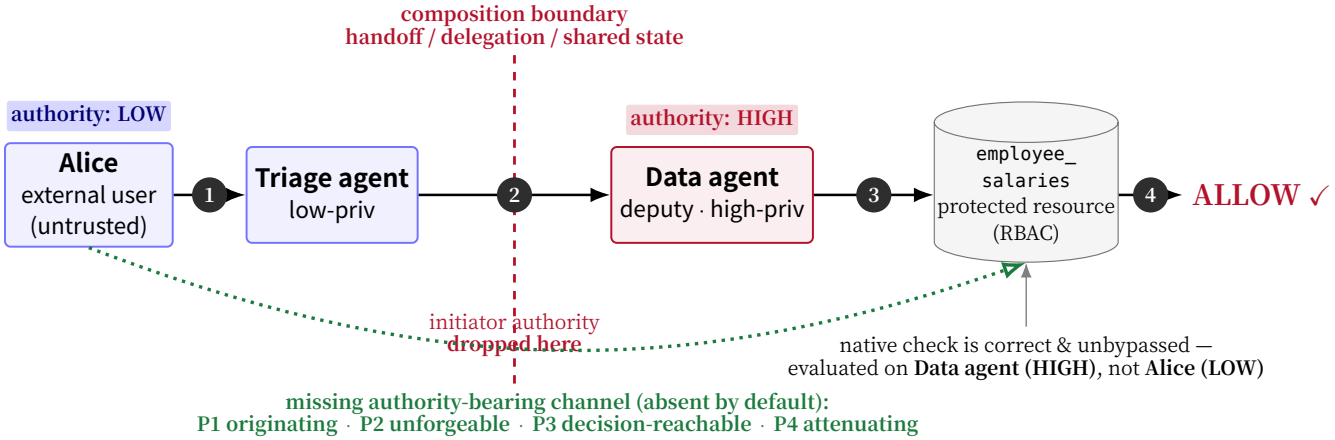


图 1: 组合边界处的主体替换。运行示例中，低权限用户 **Alice** 最终驱动了对受保护资源 `employee_salaries` 表的读取；真实案例对应 §1.1 中的 ServiceNow Now Assist 事件。实际发生的是：❶ Alice 的请求以低权限进入；❷ 请求跨过交接边界进入高权限代理者，发起者权限在边界处丢失；❸ 代理者用自己的凭据调用资源；❹ 资源原生访问控制（RBAC/ACL/OAuth）正确且未被绕过，但检查对象是代理者（**HIGH**），而不是 Alice（**LOW**），于是特权读取成功。缺失的部分（虚线）是：没有原生通道把 Alice 的权限以所需四个性质传到决策点：起源性 P1、不可伪造性 P2、可达决策点 P3、权限衰减 P4。

份原语：AutoGen 给每条消息盖上 `source` 字段，CrewAI 给组件分配不可变的 `fingerprint`，Google A2A 在传输层建立调用者身份。可是，正如 §4 所示，这些原语各自只补上问题的一部分，主体替换仍然存在。

原因是结构性的：跨组合传播发起者权限要求通道**同时**满足起源性、不可伪造性、可达决策点、权限衰减（§2.1 的四性质标尺）。每个已发布原语只提供不同的真子集，没有一个达到合取条件；逐原语 verdict 构成表 4 的 failed-fix matrix，也是本文中心结果。字段存在，但它暗示的安全保证不存在。这种部分通道甚至可能比没有通道更危险，因为开发者看到身份字段后会合理地假设主体已被携带；缺失性质则静默击穿该假设。1988 年的 confused deputy 没有这种虚假保证，因为当时并不存在一个可供信任的身份字段。多智能体框架首次把这种字段作为默认机制交付，这正是新失败模式所在。

官方材料也支持这一点：CrewAI 的 tool-hook 文档用执行 agent 的角色保护敏感工具，而不是用发起者角色 [8]；AutoGen 文档把说话者 `name` 描述为“实际发送消息的人” [31]，这对相邻一跳是合理的，但当发送者本身是代理者时就不再安全。实践者也注意到了同样的问题：身份只是“字符串（角色名）”，没有“cryptographic proof” [7]；运行时也不会“enforce that the source matches the actual sender” [32]。

1.3 为什么难以看见，以及我们如何隔离它

这类失败很容易被误认为提示注入或模型随机性，因为真实 agent 事故常伴随注入指令、不可信文档或概率性工具调用 [10, 14, 40, 42, 65]。我们用确定性差分测试隔离框架层失败（§3）：一个良性的忠实转发器让“只有组合后才失败”的现象无法归因于模型或 payload；可伪造性差分进一步把原因定位到缺失通道（P2）。

1.4 贡献

一句话概括差异： 既有工作主张应该存在一个承载权限的通道；我们在实现层测量到，该通道默认并不存在。我们也明确哪些不是本文的新意：confused-deputy 模式是经典问题 [17]；已有工作观察过多智能体 confused deputy 行为（SEAgent [18]），也命名了 trust--authorization gap（SoK [50]）。本文贡献如下。

1. **一个失败修复的自然实验**（§5）是我们的中心结果：四个框架为解决身份传播而发布的原语（AutoGen `source`、CrewAI `fingerprint`、LangGraph `config.configurable`、A2A transport identity）各自只保护 {P1--P4} 的不同真子集，没有一个达到合取；没有一个做 P4 衰减，最强的也未在决策点被咨询（P3）。因此已发布身份字段不能预测安全性（**虚假保证**）。

2. **四性质标尺让上述判定可决定**（§2.1）：起源性、不可伪造性、可达决策点、权限衰减。这些条件必须共同成立，每一个都独立必要，并且每一个缺失都有一个已发布原语作为 witness。它不是分类法，而是把“修复失败”变成“因为缺失某个 P_k 而失败”的测量仪器；充分性由运行过的正控支撑，而非形式证明。

3. **一个移除 LLM 混杂因素的确定性差分方法**（§3）：用只转发任务、不含模型且无攻击能力的 dutiful relay，在良性输入下把 P2 可伪造性

和 P1 多跳起源性转化为运行时判定，而不是只观察 ALLOW/DENY。由于我们的主张关于一类原语，我们还在版本控制中预注册了此前未见的第四个框架 OpenAI Agents SDK 的判定，并确认所有单元格，避免事后拟合（§4）。

4. **框架无关的一致性测试**（§7.1）：可运行诊断会报告每个原生通道满足 P1--P4 中的哪些性质，并用最小的 conformant out-of-band 通道验证根因。我们**不**提出新的委托协议。

两个支持性检查补全论证：三个生产模型在 90 次良性试验中有 89 次走上框架允许的不安全路径（§6.1）；对 13 个框架/应用的 census 未在任何审计过的跨 handoff site 中发现承载发起者权限的字段（§6.2）。二者都从属于差分测试，后者承担根因归因。

路线图。 论证顺序是：不变量 (†) 与四性质标尺（§2.1）→ 决定各性质的差分方法（§3）→ (†) 违例复现、归因，并在保留框架上样本外确认（§4）→ 中心结果：每个已发布身份修复只保护真子集，因此修复失败（§5）→ 支持性广度检查（§6）→ 可运行一致性测试（§7.1）。

2. 模型与威胁模型

2.1 组合何时保持权限？四性质说明

我们从组合 workflow 必须保持的不变量出发，推出任何解决方案需要满足的条件。记 $\mathrm{init}(r)$ 为发起请求 r 的主体，$\mathrm{exec}(r)$ 为最终在受保护资源处执行该请求的主体。下式用 effauth 表示有效权限，用 auth 表示权限；最小权限不变量为：

$$\forall r: \quad \mathrm{effauth}(\mathrm{exec}(r)) \le \mathrm{auth}(\mathrm{init}(r)). \qquad (\dagger)$$

请求经过中间者路由时，不应放大它所代表的权限。**主体替换**正是 (†) 的违反：受保护资源的原生检查通过了，但检查的是代理者 $\mathrm{exec}(r)$ 而不是发起者 $\mathrm{init}(r)$，因此发起者不能直接执行的动作在组合后成功。

两个范围约定使 (†) 精确。第一，$\mathrm{init}(r)$ 固定在 workflow 的信任边界，即威胁模型中把 r 引入系统的外部方；它不会被中间代理者重新定义，所以在多跳链上有稳定含义。第二，偏序 $\le$ 和算子 min 都在单个资源的授权域内解释（一个 warehouse 的 RBAC、一个文件系统 ACL、一个 API 的 OAuth scopes）。在这个域内，权限对我们要衰减的 $(\mathrm{deputy}, \mathrm{init})$ 对形成偏序并有定义良好的 min，但不必是完整 lattice：POSIX uid 和 RBAC role 都未必全序。我们不假设跨异构资源存在统一权限序；§4 中每个结果都相对于一个受保护资源表述。跨资源统一权限不在本文范围内。

要跨组合边界维持 (†)，必须有某个通道把发起主体以可用形式带到 enforcement point。这样的通道必须**共同**满足四个性质：

- **(P1) 起源性。** 它携带发起主体，而不仅是上一跳的相邻 peer。每跳重新建立的通道只认证立即发送者，无法代表多跳链起点。
- **(P2) 不可伪造性。** 它由可信运行时设置并受完整性保护，而不是来自攻击者可影响的内容。攻击者能在文本中声明的身份只是 ambient claim，不是 capability。
- **(P3) 可达决策点。** 它必须抵达并可被受保护资源的授权决策读取，而不是只停留在日志、trace 或审计路径中。这个性质是必要但不充分的：一个可达决策点的通道仍可能是可伪造的（P2），因此防御者能读到一个值，并不意味着已经得到可信值。
- **(P4) 权限衰减。** 每次操作的授权应计算为 $\min(\mathrm{authority}(\mathrm{deputy}), \mathrm{authority}(\mathrm{init}))$，使携带发起者真正约束操作。

P1/P2 是通道性质；P3/P4 是消费侧规则，使通道真正承载安全意义。它们分别来自 capability security 与 protection systems 中熟悉的要求 [12, 22, 25, 33, 34]；本文贡献在于经验性地证明这些性质必须作为**合取**出现。任意真子集都不足以维护 (†)，而部分通道会造成 §1.2 中的虚假保证。

每个性质都是必要的，并且每种失败都由今天发布的原语实现。 这四个性质不是我们先验构造出的愿望清单：去掉任何一个，仍会留下具体绕过，并且都有已发布通道作为 witness。A2A transport identity 与 AutoGen 重新盖章的 `source` 丢失 P1（多跳 laundering）；in-band origin claim 丢失 P2（可伪造文本）；CrewAI `fingerprint` 丢失 P3（audit-only）；LangGraph `config.configurable` 丢失 P4（从不衰减）。这些正是 §5 的 failed-fix matrix，并在 §4.4 中对 P1/P2 做运行时判定。每个性质都独立承载安全意义。本文不以形式证明主张充分性，而是用下面运行过的正控支撑：P1--P4 是一个经验验证过的目标，不是不可能性定理；在我们的测试中，缺任一性质的通道都会允许替换。

正控：P1--P4 是已部署的权限保持设计的投影，而不是本文发明的性质。 成熟的非 agent 委托机制已经强制这些标准要求：OAuth 2.0 token exchange / on-behalf-of 在多跳中保留原 subject（P1），由运行时签发（P2），并由 resource server 在决策处强制收窄 scope（P3、P4）[16, 19]；macaroons 与 SPKI 把衰减设为一等概念，而不是建议性字符串 [5, 13]；数据库 `SECURITY INVOKER` 例程则按调用主体授权（$\mathrm{exec}(r) = \mathrm{init}(r)$）。这些机制在自己的场景中都满足完整合取。我们不仅引用它们，还把其中两个跑进同一个四元组 harness：一个 RFC 8693 token exchange（真实 RS256 签名）和一个 live PostgreSQL 16 `SECURITY INVOKER` routine 都拒绝 composed low-privilege cell，同时保留 high-privilege cell；`SECURITY DEFINER` 在同一个真实引擎上作为负控重新展示崩塌（`poc/ poc_positive_control_rfc8693.py`、`poc/`

poc_positive_control_pg_invoker.py）。 §7.2 的 out-of-band channel 是把这些成熟机制最小投影到 agent boundary 上的结果，说明合取不仅可实现，而且已经被执行验证，并非我们论证框架制造出的假象。

这一说明给出一个可证伪且反直觉的预测：身份字段的存在并不能预测安全性。§5 会在每个已发布原语上验证这个预测。

2.2 威胁模型

场景。一个建立在主流编排框架上的多智能体 workflow。Agent 会委托、交接或共享状态。我们把执行敏感操作并实施部署原生授权策略的下游接口称为受保护资源：例如带 RBAC/RLS 的数据库关系、带 ACL 的文件，或带 OAuth scope 的 API 操作。至少有一个资源会对执行调用的主体做检查。这是生产部署依赖的平台原生授权，扎根于 RBAC 和访问控制逻辑等标准访问控制模型 [2, 48]；我们不添加自己的 guard。在附录 C 的形式框架中，我们用 sink/sink_allows 作为同一 enforcement point 的简写；在相关工作中，sink 则保留 taint analysis 中的标准含义。

攻击者。攻击者只控制流入 workflow 的**不可信外部输入**：最终用户 prompt 以及外部工具返回的数据，这正是 RAG 和工具集成式 LLM 应用中的常见输入面 [26]。因此攻击者能以文本形式声明任意内容，包括伪造谁在请求某个动作。

信任假设（强版本）。每个 agent 都是**良性的**：没有投毒、没有恶意、没有运行注入 prompt。我们故意排除恶意 agent 的弱模型，因为那会退化为普通提示注入。这个设定使结果更强：缺口在框架最有利的条件下仍然存在，没有注入可过滤、没有不对齐 agent 可检测。因此，即便一个部署完全解决了提示注入和 alignment，仍会表现出该缺口；若存在恶意 agent 或 payload，只会更糟。注入是触发器，框架组合才是我们隔离出的放大器。本文结果是框架行为的下界，而不是我们刻意构造的最坏情况。

安全目标。§2.1 的不变量 (†)：请求执行时不应拥有超过发起者的权限。我们研究的违反即主体替换：资源原生检查正确通过，但对象是代理者而非发起者。

不在范围内。模型 jailbreak、受保护资源自身访问控制实现漏洞、传输/网络安全、框架供应链妥协。我们假设资源访问控制正确；失败位于其上游的组合边界。

2.3 为什么 (†) 是安全性质，而不是配置选择

厂商可能会像 ServiceNow 那样反驳：如果 agent 被配置为彼此发现和委托，那么高权限 agent 响应低权限请求只是按设计工作 [55]。这混淆了功能默认值与安全论证。最小权限原则 [47]要求每个操作以不超过其代表主体的权限执行；一个操作因为被路由经过代理者而获得权限，就是权限放大，恰好是最小权限要防止的失败。“by design”描述机制如何运作，并不能证明行为安全。Microsoft Copilot Studio 的指导也以朴素语言陈述了同一目标：一个 agent 不应通过另一个 agent 间接获得自己无法直接访问的信息或触发自己无法直接触发的动作。这正是 (†)。因此分歧不在于 (†) 是否应该成立，而在于它实际是否成立；我们的结果（§4）显示，在这些平台所依赖的框架中，没有已发布通道默认强制它。

放弃 (†) 不是局部决定：如果代理者权限占主导，任何能到达代理者的发起者都继承其权限，实际访问控制 lattice 会坍缩为所有可达 agent 权限的并集。(†) 是防止这种坍缩的最小条件：它正是每个 agent 内部的 RBAC/ACL/OAuth 已经强制的东西，而组合边界静默丢失了它。ServiceNow 的提示注入保护已启用但没有阻止外泄，也说明该失败与注入防御正交：by-design 不等于 by-security-design。

3. 方法：确定性差分测试

我们的方法是判定 §2.1 四个性质的仪器，把每个性质从观察到的 ALLOW/DENY 转化为运行时 verdict。直觉如下：dutiful relay 移除模型混杂因素（§3.1）；四元组复现 (†) 违反（§3.2）；forgeability differential 判定不可伪造性 P2，多跳变体判定起源性 P1（§4.4）；消融测试在已经恢复 P1∧P2 的前提下，分别测试恢复 decision-reachability（P3）或 attenuation（P4）是否足以关闭缺口（§3.4）。附录 C 给出两因素控制实验、truth table 和四个命题；我们对每个单元格都采用它能承受的最强证据层级：能执行差分的地方使用 executed/runtime evidence，不能执行的 non-read 使用 structural evidence（表 9）。

3.1 Dutiful relay：移除模型混杂因素

我们把每个 agent 的 LLM 替换为**确定性、尽责的转发器**：一个规则化 stand-in，它转发任务并发出固定下游动作，无论请求是谁发起的 [38, 39, 57]。关键是，它会忠实抄录请求中陈述的 origin，不管真假；良性 relay 无法区分注入的 origin claim 与真实 origin，因为两者都以文本到达。在没有攻击能力的情况下，任何“只有组合后发生”的失败都来自框架，而非模型。Relay 是受控 witness：它说明框架允许一条良性、无 payload 的替换路径；随后 §6.1 说明真实模型也会走这条路。

3.2 四元组：复现现象

对每个框架，我们针对同一个敏感操作运行四种配置（图 6）：(i) `isolated-low`：低权限主体直接访问受保护资源，期望 `DENY`；(ii) `isolated-high-benign`：高权限用户经高权限 agent 访问资源，期望 `ALLOW`；(iii) **`composed`**：低权限请求经过 delegation/handoff/broadcast 到高权限 agent，再访问资源，出现**`ALLOW`**，即主体替换；(iv) `patched-composed`：组合路径加上发起者权限保留与衰减，恢复 `DENY`。

结果在**受保护资源**处读取：特权操作是否实际执行，而不是 agent 是否抛出异常。一个 relay 可以吞掉错误并仍然回答。`isolated-low` 与 `composed` 的对比是经典 confused deputy（附录 C 命题 1 的蕴含）；本文的新重量在 §3.3 的可伪造性差分，它把崩塌归因到缺失通道。

我们在三个结构不同的组合机制上实例化四元组，并使用三个真实授权后端：CrewAI 的 delegation-constructor boundary 与 live PostgreSQL 16 RLS，LangGraph 的 `Command(goto)` 与 kernel-enforced POSIX ACL（mode-`0600`，`sudo -u nobody`），AutoGen/AG2 的 per-recipient role rebinding 与 live GitHub REST API fine-grained PAT scopes（`200`/`404`）。安全结果只依赖一个性质：资源按执行主体授权，而没有通道知道是谁提出请求。这对 RBAC、ACL、OAuth scope 都是定义性事实，而不是我们写的 guard；真实后端承担每一次决策，modeled Snowflake/OAuth 语义只作为 hermetic fallback。因为崩塌严格发生在受保护资源上游，结果不依赖资源是 live 还是 modeled：四场景不变量在三个真实后端上 verdict-for-verdict 成立（§4.1）。Relay 在 isolated-high 与 composed 运行中发出的操作相同，确认它是 relay，而不是 attacker。

3.3 可伪造性差分：把原因归到缺失通道

四元组说明组合会失败，但本身还不能证明框架缺失的通道就是原因。为归因，我们让代理者变成**防御性且各单元格完全相同**：受保护资源侧的 least-privilege-aware 组件试图把有效权限衰减为 min(`self`,`initiator`)。这个 defensive deputy 是**作者编写且站在防御者侧**的组件：框架没有给受保护资源接入 provenance，所以我们给防御者一个最有利的防御机会，即一个主动想衰减到 initiator 的组件，并展示即便如此也会被框架唯一携带的通道绕过。这里的责任边界很明确：该差分只证明 P2；没有原生 authority-bearing channel 这一事实来自结构阅读和 census（§6.2），不由这个对比单独承担。

- **In-band（框架唯一携带的通道）：** 代理者从消息/任务文本读 initiator。该文本确实会跨过边界：例如 LangGraph 的 `add_messages` 保留 message object，AutoGen 的 role rebinding 改变 `role` 而不是 content。因此我们不是声称信息被销毁；问题是它没有认证。诚实请求中，该 claim 是真实低权限发起者，代理者正确衰减 → `DENY`，防御看起来有效。伪造请求中，攻击者按威胁模型声明更高 origin，良性 relay 忠实转录，代理者衰减到伪造值 → `ALLOW`。因此 in-band 防御被伪造绕过。
- **Out-of-band（框架缺失的通道）：** 代理者从可信运行时基于真实 session 设置的完整性保护通道读取 initiator，永不来自请求文本。同样的伪造请求不能移动它 → `DENY`。

代理者代码、请求和非原生 plumbing 在 in-band forged 与 out-of-band 单元格之间逐字相同；唯一独立变量是通道可伪造性。因此我们编写的 plumbing 不可能解释一个单元格 ALLOW、另一个 DENY，能解释翻转的只能是 forgeability。这说明 in-band provenance 是 ambient claim，不是 capability（P2 失败），也证明关闭缺口需要框架没有提供的通道。这个对比承担附录 C 命题 2 的因果归因，而不是 §3.2 中相同资源操作本身。

3.4 单变量消融与最小修复

组合失败建立在一个缺失前提上：框架没有携带 originating 且 unforgeable 的通道（P1∧P2）。在该前提之上，两个消费侧修复分别关闭缺口，各自对应剩余两个性质中的一个。它们不是两个独立根因：一旦假设存在 out-of-band channel，它们处理的是 recovered initiator 在受保护资源处仍可能被忽略的两种方式：要么根本没有到达决策，要么到达了但没有参与衰减。我们把它们作为单变量干预来测试，每个干预只预设 provenance 已经跨过边界被携带（即 §3.3 的 out-of-band channel）。

- **E3：origin-aware authorization（policy layer）。** 受保护资源在代理者凭据不变的情况下咨询保留下来的 initiator；因为决策成为 initiator 的函数，动作被拒绝。
- **E4：authority attenuation（capability layer）。** 代理者的有效凭据被下调到 min(`deputy`,`initiator`)，资源原生检查保持不变；被下调的凭据无法通过原有检查，动作被拒绝。

二者在三个框架中都独立恢复 `DENY`；它们都是 §2.1 的消费侧修复，并且都**预设**已有 P1∧P2 通道，而框架并不提供这个通道。安装该通道是**非默认且侵入式**的：这里的 seam 指正确衰减因为缺少原生 API 而必须添加代码的位置。在 CrewAI、LangGraph 和 AutoGen 中，计数都相同地为三处：(1) 用 out-of-band channel 承载 initiator，因为 payload 没有 authority 字段；(2) 在代理者处加入衰减点，因为组件绑定的是静态凭据，没有逐调用 `min(self, initiator)` hook；(3) 在信任边界设置该通道的 runtime，因为没有原生 initiator propagation。这个数只是 qualitative repair-distance indicator，不是度量；稳健结论是该计数非零，并且在结构不同的框架中一致。我们把这个修复严格作为根因验证，而不是新协议提案。

局限。 每份证据的边界统一收束在 §6.4 的 claim-strength 摘要（表 5）中，附录 C 给出完整陈述，包括 in-band modeling、sink fidelity、controlled-witness vs. real-model check、corpus 和 falsifiable boundary。

形式框架。 这些组件实例化了一个双因素控制实验（composition × initiator channel）：relay 保

持不变，deputy rule 固定，因此每个单元格只隔离一个变量。单一 sink 的 truth table 和命题 1--4 位于附录 C（§C）：命题 1 是蕴含关系，所以四元组是 controlled witness；命题 2/P2 和命题 3/P1 承担经验重量；命题 4 综合 necessity-with-realized-witnesses。

4. 评估

评估围绕四性质说明组织。我们先固定实验设置（§4.1），再做最强检验：在此前未见的框架上预注册预测（§4.2）。随后转向三个深入研究框架：(†) 违反在结构不同的组合原语上复现（§4.3），并通过可伪造性差分与单变量消融完成归因（§4.4）。高潮是§5：所有已发布身份原语都只满足 {P1--P4} 的部分子集，因此不能维护不变量。所有确定性 artifact 首次运行即可复现（exit 0）。PoC 源码位于 `poc/`，包括输出 per-property matrix 的框架无关一致性测试 `poc/conformance_aggregate.py`。

4.1 设置

我们研究三个主流框架及其结构不同的组合原语，并接入三个真实授权后端：CrewAI / live PostgreSQL 16 RLS、LangGraph / kernel-enforced POSIX ACL、AutoGen / live GitHub PAT scopes（表 1；机制和资源 fidelity 见 §3.2）[23, 24, 59]。除 in-vivo 检查外，每个 agent 的 LLM 都被 §3.1 的确定性 dutiful relay 替换。OpenAI Agents SDK 的 `handoff()` 边界被**保留**用于 §4.2 的预注册样本外预测；在建立四性质说明时刻意不检查它。

我们把两代 AutoGen 作为一个框架家族处理：AG2 的 `name`-stamped messages 用于 §4.3 的 role-rebinding 四元组；autogen-core 的 per-message `source` 字段用于 §5 的 failed-fix 实验。

失败修复实验还使用 autogen-core/ autogen-agentchat `0.7.5` 的 per-message `source` 字段，多跳 P1 差分也使用 autogen-agentchat `0.7.5`。

真实后端确认。 为排除 modeled backend 的质疑，我们把 in-process resource 替换为真实后端，并重新运行四元组，让真实系统完全承担决策。对 live PostgreSQL 16 RLS engine（CrewAI）和 live GitHub REST API（AutoGen），verdict 与模型化后端完全一致：DENY/ALLOW/ALLOW/DENY。composed 的 `ALLOW` 和 patched 的 `DENY` 都由后端自身产生（`source_of_verdict` ∈ {`postgres-engine`, `github-api`}），没有应用层 guard：composed 情况下，防御性代理者找不到可信 initiator channel，于是以自己的主体执行，后端正确授权它，这就是主体替换；patched 情况下，同一段代码衰减到 `min(deputy, initiator)`，后端拒绝。对 kernel POSIX ACL 而言，后端本身就是真实系统；模型化实现只保留为可复现 fallback。Artifact 位于 `experiments/real-sink-postgres/` 和 `experiments/real-sink-github/`，后者包含不需要 token 的 `--replay` fixture。

4.2 预注册的样本外测试

我们先把方法用于最强检验：一个四性质说明从未见过的原语。因为该说明声称覆盖一类组合原语，最强形式是预测：先注册 verdict，再检查保留框架。这是 out-of-sample，而不是 post-hoc fit。该说明预测：任何组合原语，只要其边界没有携带原生、运行时设置、完整性保护、起源绑定的通道，就会允许主体替换，不管边界表面机制如何。

我们在编写或运行测试之前，在版本控制中预注册 OpenAI Agents SDK `handoffs` 原语（`openai-agents` 0.17.4，HEAD `8eaa4b9`）的 verdict。该边界是第四种机械上不同的成员（图 2）：transfer 是模型发出的 `transfer_to_<agent>` 工具调用，跨越边界的是 conversation transcript 的 reprojection。特别地，它会跨 handoff 保留完整 transcript，包括原始请求中的可伪造 origin claim；这已经是 provenance survival 的最强情形。预测仍然成立：**保留不等于认证**，保留下来的 originator 仍是可伪造、未认证文本，没有 principal--authority binding。

原生 handoff 不满足 P1--P4 中任何一个性质（§7.1 的 conformance test）。观察修正了一个机制细节：P1 失败不是像 AutoGen `source` 那样重盖章到相邻 peer，而是 transcript 被保留但可伪造；我们在 `results.md` 中报告该差异，而不是为了贴合预测事后改写。

> **R4（样本外）** 我们为保留的 OpenAI Agents SDK 预注册的每个 verdict 都得到确认；其原生 handoff 不满足 P1--P4。该说明预测了未见过的原语，而不仅是构造它的原语。

4.3 现象：(†) 违反复现

三个框架的结果一致（图 6）：`isolated-low` 被拒绝，`isolated-high-benign` 被允许，说明代理者在隔离状态下是合法的；但 `composed` 被允许，特权操作虽然由低权限方发起却在受保护资源处执行。这里的 (†) 违反是在受保护资源处读取的：ground truth 是操作是否执行，而不是 agent 是否抛出异常。`patched-composed` 在三个框架中均恢复 `DENY`。这就是经典 confused deputy 的受控 witness，而不是本文发现本身（附录 C 命题 1）；真正的因果重量来自 §4.4 的归因。

> **R1（† 被违反）** 在框架默认设置下，一个低权限发起者无法直接到达的操作，在组合后于受保护资源处执行：资源检查通过了，但检查对象是代理者。

表 1: 实验设置。三个深入研究框架在结构不同的组合原语上接入三个真实授权后端，并保留第四个框架仅用于预注册预测测试。贯穿全文测试的是同一性质：资源按执行主体授权，且没有原生通道知道请求由谁发起。

框架	组合原语（边界）	授权后端
CrewAI	task-object reconstruction：`base_agent_tools._execute` 重建 `Task(...)`	**真实 PostgreSQL 16 RLS**，两个真实 DB role
LangGraph	`Command(goto=...)` handoff + `add_messages` reducer	**kernel POSIX file ACL**，mode-`0600`，`sudo -u nobody` → `EACCES`
AutoGen / AG2	per-recipient role-rebinding：`conversable_agent.py`	**真实 GitHub REST API**，fine-grained PAT scope，`200/404`
OpenAI Agents SDK (held-out)	transcript-reprojection：`handoff()` 触发 model-emitted `transfer_to_<agent>` call	OAuth scope，HTTP `403` `insufficient_scope`

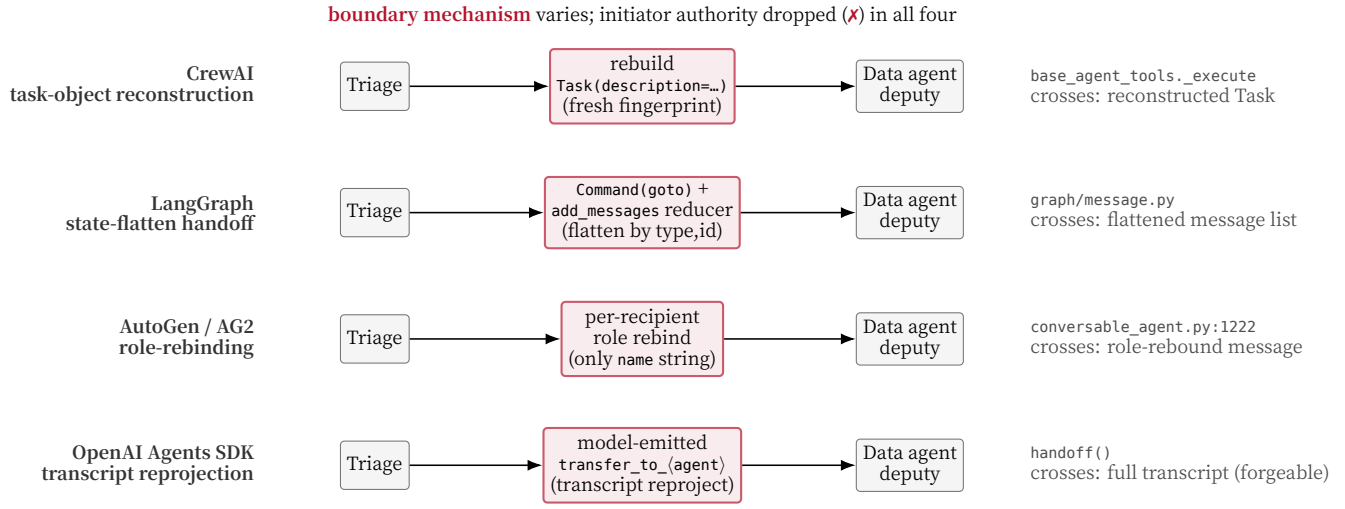


图 2: 本文测试的四个组合原语边界。每个框架以机械上不同的方式丢失发起者权限：CrewAI 重建 task object，LangGraph 按 id flatten state，AutoGen 对每个 recipient 重绑 role，保留的 OpenAI Agents SDK 做 model-emitted transcript reprojection。尽管机制不同，丢失以相同方式复现，符合四性质说明的预测。

表 2: 保留框架预测（OpenAI Agents SDK）：预注册 verdict 与观察结果。每个单元格都被确认；两次运行字节相同，exit 0。

单元格 / 对比	预测	观察
isolated-low	DENY	DENY
isolated-high-benign	ALLOW	ALLOW
composed	**ALLOW (collapse)**	**ALLOW**
patched-composed (out-of-band)	DENY	DENY
forgeability P2 — in-band honest	DENY	DENY
forgeability P2 — in-band **forged**	ALLOW (bypassed)	**ALLOW**
forgeability P2 — out-of-band forged	DENY	DENY

表 3: 四元组结果：三个框架 verdict 完全一致。每个单元格确定性复现，verdict 在受保护资源处读取。

单元格	verdict	含义
`isolated-low`	DENY	本地策略正确
`isolated-high-benign`	ALLOW	代理者在隔离状态下合法
`composed`	**ALLOW**	**主体替换，(†) 违反**
`patched-composed`	DENY	根因修复验证

4.4 归因：可伪造性 P2 与单变量消融 P3/P4

可伪造性差分（P2）。 保持防御性代理者代码逐字相同，只改变通道：in-band 情况下，防御对诚实请求会拒绝，但会被伪造请求绕过；out-of-band 情况下，同样的伪造请求无法移动 initiator，结果为 `DENY`。唯一自由变量是可伪造性，因此 in-band provenance 是 ambient claim，而非 capability。

R2（可伪造通道，P2） 在逐字相同的 defensive-deputy 代码下，in-band 恢复的 initiator 会在伪造请求中从 `low` 移到 `high`。唯一自由变量是通道；in-band provenance 不是 capability。

多跳起源性差分（P1）。 起源性只在多跳中显现，因此我们单独测试且不使用伪造。在真实 AutoGen `RoundRobinGroupChat` 链 $A \to B \to C$ 中，A 是低权限发起者，B 是良性 admin-tier relay，C 是防御性代理者。运行时把每条消息的 `source` 重新盖章为产生该消息的 agent，所以 C 恢复到的是 B（相邻 peer），不是 A，并按 B 的权限衰减，得到 `ALLOW`；若按 min(deputy,A) 应为 `DENY`。out-of-band 通道端到端携带 A 时会 `DENY`；单跳控制 $A \to C$ 中，source 与 A 重合，也会 `DENY`。结构不同的 LangGraph `StateGraph` 链（`poc/poc_*_multihop_p1.py`）也复现同样失败。这使 A2A corollary 具体化：单跳传输层 peer identity 能认证 $B \to C$，但不能认证 A，因此第二跳崩塌；`source` 回答的是“谁最后说话”，不是“谁最初发起”。

单变量消融（P3/P4）。 一旦假设存在 P1∧P2 通道，缺口有两个消费侧修复：E3 让资源在代理者凭据不变时咨询 initiator，恢复 P3；E4 把代理者有效凭据下调为 min(deputy,initiator)，资源原生检查保持不变，恢复 P4。二者都足以恢复 `DENY`，但都预设 §5 证明没有已发布原语提供的 P1∧P2 substrate。

5. 已发布身份原语：失败修复的自然实验

框架并非忽视身份；它们已经发布了身份原语。我们问的问题是：如果防御者尝试把某个原语当作权限通道，它满足 P1--P4 中哪些性质？P2 由 §4.4 的可伪造性差分运行时判定。AutoGen `source` 的 P1 由多跳起源性差分判定：重新盖章、hop-local 的 `source` 恢复的是相邻 peer，而不是 originator。CrewAI `fingerprint` 的 P1 由结构 witness 判定：delegated `Task` 会用新的 fingerprint 重建；A2A 的 P1 由其规范判定，因为它认证的是传输层一跳身份。P3 由结构证据判定（native API 是否存在、authorization path 是否读取，并带代码位置）；LangGraph 的 P4 由结构证据判定；CrewAI `fingerprint` 和 AutoGen `source` 的 P4 则由运行时 attenuation differential 判定。

我们检查四个已发布通道：三个 framework-native 通道（AutoGen `source`、CrewAI `fingerprint`、LangGraph `config.configurable`）和一个 protocol-level 通道（A2A transport identity，作为文档 witness）。选择它们不是为了挑容易的案例，而是为了覆盖设计空间：我们故意包含 `config.configurable` 这个最强已发布原语，因为它同时满足两个通道侧性质 P1/P2，因此最接近真正修复。表 4 给出结果。

这些 verdict 不是手工拼接的：它们来自框架无关 conformance test（`poc/conformance_aggregate.py`）。该测试驱动每个框架 probe，并以共享 schema 输出 per-property matrix。运行它会在运行时复现三行 framework-native 结果：AutoGen `source`（P1✗ P2✗ P3✓ P4✗）、CrewAI `fingerprint`（P1✗ P2✓ P3✗ P4✗）、LangGraph `config.configurable`（P1✓ P2✓ P3✗ P4✗），并在所有框架中同时给出 conformant out-of-band channel（四项全 ✓）。A2A 作为文档 witness 进入，因为它没有发布可运行的框架 adapter。这个测试就是四性质说明的 artifact 形态：维护者运行它，是为了知道一个原语缺哪一个性质，而不是仅因存在身份字段就假设安全。

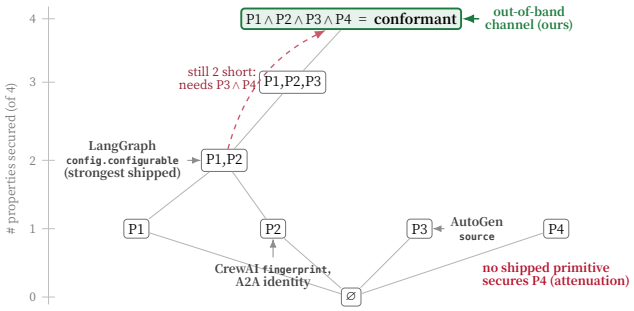


图 3：已发布身份原语位于 {P1,P2,P3,P4} 子集 lattice 中的位置。每个只保护真子集；没有一个到达顶端 conformant 合取。没有已发布原语做 P4 衰减；最强的 LangGraph `config.configurable` 保护了通道侧 P1∧P2，却仍缺 P3∧P4。图的几何含义就是主张：身份字段必要但不充分，会造成虚假保证。

上标表示证据来源：^{r} 为运行时差分，^{s} 为结构分析，^{d} 为文档 witness。子集 lattice 见图 3。

发现并不是“每个原语忘掉一个不同性质”这么弱，而是更强，并直接反驳“加一个身份字段就安全”的直觉：这些原语散落在 {P1--P4} 的子集 lattice 中，各自保护不同真子集，而且有些原语保护的性质更多、逐步接近 apex，但没有一个达到合取（图 3）。这个对已检查已发布原语的 universal 是可证伪的：若存在一个已发布原语满足合取，就会推翻它；我们没有发现。与此同时，conformant out-of-band channel 说明目标可达，因此失败是偶然设计缺失，而非不可能定理。

按代码读取 verdict。AutoGen `source`：由 producer 设置（`messages.py:86`，无 validator），并可被下游当作可信文本读取（`_selector_group_chat.py:224`），所以 decision-reachable（P3✓）；但它命名的是相邻发送者，每跳重新盖章（P1✗），而 forgeability differential 能把它从 `low_user` 移到 `admin_user`，代理者代码不变（P2✗）。CrewAI `fingerprint`：不可变 runtime UUID（P2✓），但命名的是执行组件；delegated `Task` 会得到新 fingerprint（P1✗）；它只到达 tracing（`tool_usage.py:258,489`），authorization path 零读取（P3✗）。A2A 共享这种 audit-only profile，只认证立即 peer（P1✗）。LangGraph `config.configurable` 是最强已发布原语：它在 `RunnableConfig` 上 out-of-band 设置，伪造 in-message claim 无法移动它（P2✓），并以 run-scoped 方式合并进每个 node（`_internal/_config.py:195`，P1✓）；但它只被用于 thread routing（`pregel/ remote.py: 386`），从不用于授权。当 `configurable.initiator=nobody` 时，特权读取仍然 ALLOW（P3✗），也没有原生 min(deputy,init) at-

表 4: 失败修复的自然实验：每个已发布身份原语满足 P1–P4 中哪些性质，以及为什么依赖它的防御者仍不安全。没有一个达到合取；只有 §7 的 out-of-band channel 是 conformant。该规则区分了两个设计目标就是身份/provenance的机制（`source`、`fingerprint`，是真正的 failed fixes）与两个只是能够携带 initiator 的 opportunistic metadata（A2A、`config.configurable`）。每个单元格的 evidence basis（runtime vs structural）见表 9。

身份原语	P1 起源	P2 不可伪造	P3 可达决策	P4 衰减	conformant	缺失性质及风险
AutoGen `source` (autogen-core 0.7.5)	$\times^r$	$\times^r$	$\checkmark^s$	$\times^r$	否	**P2**。sender 字段是攻击者可设置的普通文本，伪造 origin 会被相信。
CrewAI `fingerprint` (commit `051fa0c`)	$\times^r$	$\checkmark^r$	$\times^s$	$\times^r$	否	**P3**。不可伪造 UUID 只到达日志/trace，不到达授权决策。
A2A transport identity	$\times^d$	$\checkmark^d$	$\times^d$	$\times^d$	否	**P1**。认证一跳的立即 peer，而不是最初发起请求的主体。
LangGraph `config.configurable` (0.6.x)	$\checkmark^r$	$\checkmark^r$	$\times^r$	$\times^s$	否	**P3/P4**。起源且不可伪造，但框架从不在受保护资源决策处咨询它，也不会据此衰减。
out-of-band channel（本文）	$\checkmark^r$	$\checkmark^r$	$\checkmark^s$	$\checkmark^r$	**是**	四性质同时满足，是唯一恢复正确拒绝的配置。

tenuation（P4✗）。这里 P4 之所以是 structural，正是因为 runtime-confirmed P3 failure 已经说明没有 consultation point 可运行 attenuation differential；它的剩余缺口完全是消费侧问题，是关于 channel 的事实，而不是声称 LangGraph 应用没有认证用户。

没有任何已发布原语满足 P4（attenuation），只有一个满足 P3（且它是可伪造的 `source`）。通道侧性质本身是可达的，`config.configurable` 就满足 P1∧P2；因此它的失败更有决定性：即便通道安全，如果框架从不在决策处咨询并做衰减，仍然不能维护 (†)。

R3（失败修复，中心结果） 没有已发布身份原语满足 P1∧P2∧P3∧P4；即使最强的 `config.configurable` 也只保护通道侧，并未在决策处被咨询。增加身份字段不能让组合 workflow 安全。

6. 广度与生态检查

根因已由 §4--§5 建立并归因。本节加入三个支持性检查：真实模型可达性、默认缺失的广度、以及与提示注入防御的正交性。它们确认 reach、breadth 和 orthogonality，但都不承担因果主张；因果主张来自差分测试（§4.4）与性质矩阵（§5）。表 5 明确每份证据的支持边界。

6.1 真实模型检查：真实模型会走框架允许的不安全路径

确定性结果建立并归因了失败，但 relay 可能被认为过于人工。因此我们把 relay 替换为三个独立开发的生产模型：OpenAI gpt-5.5 `gpt-5.5-2026-04-23`、DeepSeek `deepseek-chat`、Alibaba `qwen-max`，运行在真实 AutoGen `RoundRobinGroupChat` 中。低权限 `triage` agent 接收一个**良性**请求，没有注入、jailbreak 或伪造 claim；高权限 `data_agent` 持有按执行主体强制 scope 的 `export_records` 受保护资源。三个场景各运行十次。

三个模型作为普通 worker 都会把任务路由给 data agent，后者触发低权限发起者无法直接执行的敏感导出（低权限方只持有 `export:public`）：gpt-5.5 和 DeepSeek 各 30/30，qwen-max 29/30，总计 89/90（逐模型 breakdown 见附录 B 表 7）。唯一 non-fire 不是授权拒绝，而是模型提出澄清问题并在工具调用前终止；该 trial 中框架仍然没有携带 initiator channel，只是模型当轮没有继续执行。

这说明什么。 框架允许的不安全路径会被普通模型在良性输入下行为性触达，从而回应“relay 是 strawman”的质疑。把良性请求路由给 data agent 正是 agent 的预期工作，因此这个比例反映正常 task-following，而不是攻击。该检查只在 AutoGen 上运行（CrewAI/LangGraph 的崩塌由确定性实验建立），所以 89/90 是 ecological signal，不是安全指标，也不是根因结果（表 5）。三个模型都通过提供商官方 first-party API 访问；附录 B 记录端点 resolved fingerprint，并在 Qwen date-pinned snapshot 上复跑（28/30，确认版本独立性），同时提供逐 trial transcript。

6.2 缺失通道是默认状态

在 13 个主流框架与真实应用的 corpus 中（一个 **mainstream-skewed convenience sample，不是生态总体抽样**；明确纳入规则和 pinned commits 见附录 A.1），我们扫描 22,818 个文件和 6,076 个 delegation/handoff/shared-state sites（图 4）。两个互补分母都得到零：**粗分母** 6,076 个 keyword sites 中没有一个 carry（0 carry；7 个 proximity candidates 全为 false positives；Wilson 95% UB $\leq 0.061\%$）；**严格分母** 384 个 non-test cross-handoff sites 中，247 个 identifier-flagged windows 经人工判定全部为 negative（0 carry；UB $\leq 0.99\%$，这是 nominal bound，因为 site 聚簇在 13 个 repo 内，不是独立抽样，只作指示性解释）。完整协议、词表、codebook、两个分母和 codebook reproducibility（第二个确定性 codebook 复现所有 verdict；故意宽松的 coding variant 降到 $\kappa = 0.58$，这是信息量更大

的敏感性数字）见附录 A。因此 census 是一个测量，而不是 grep heuristic。最接近的构造也没有为下游 attenuation 携带 initiator authority：在该样本中，authority-bearing channel 默认缺失。

R5（默认缺失） 在 13 个主流系统中，384 个审计过的 cross-handoff sites 中有 0 个携带 authority-bearing initiator field（Wilson 95% upper bound $\leq 0.99\%$，按主流部署现实解释，而非均匀生态抽样）。

6.3 与部署级提示注入防御正交

一个自然质疑是：这只是换名的 indirect prompt injection。不是。我们把 §4 中的精确攻击输入，也就是三个 in-vivo export requests（gpt-5.5 上 30/30）和确定性 CrewAI/ AutoGen composed-collapse tasks，交给生产提示注入检测器（`protectai/deberta-v3-base-prompt-injection-v2`，LLM-Guard 默认，rev `e6535ca`）。所有五个 composed-benign 输入都被判为 **SAFE**（$p \geq 0.986$），但每个都会触发主体替换；同一检测器会以 $p = 1.000$ 阻断四个经典 IPI payload。检测器通过，是因为没有注入特征可检测；这对 detector class 无关：composed-benign request 是 injection-free natural language，pattern matcher、classifier 或 LLM judge 都没有 adversarial feature 可依赖。该失败因此与部署级 injection defense 正交，这一点在 ServiceNow 启用却无效的保护中已经可见（§1.1；`experiments/defense-orthogonality/`）。ServiceNow 的 second-order injection 与本文并不矛盾：injection 是触发器，composition 是把任何输入（injected 或 benign）升级到 deputy authority 的放大器，而我们的 injection-free 测试正是在隔离这个放大器。

R6（与注入正交） 生产注入检测器放行所有 composed-benign 输入，却每个都触发主体替换。问题是授权主体错误，而不是 adversarial text。

6.4 每份证据支持什么，不支持什么

表 5: Claim-strength 摘要：每份证据支持什么，以及不支持什么。根因归因来自 forgeability/multi-hop differential 和 failed-fix matrix；其余证据是支持性。

证据	支持	不支持
四元组可伪造性差分	组合可违反 (†) in-band provenance 失败 P2	单独证明根因 prevalence
多跳差分	hop-local identity 失败 P1	所有协议
失败修复矩阵	已发布字段是部分子集	安全设计不可能
Census (0/384)	审计 corpus 默认无通道	生态总体比例
真实模型 (89/90)	普通模型会走允许路径	框架根因
保留预测	失败属于原语类别	穷尽设计空间证明

7. 讨论

7.1 面向组合崩塌的一致性测试

四性质刻画是可操作的：它不是抽象口号，而是框架作者可以运行的诊断。我们把 §4 中的性质检查打包为框架无关的一致性测试 `poc/conformance_aggregate.py`。对每个原生身份通道，它报告 P1--P4 哪些成立，以及该通道是否 conformant；其中 P1/P2 由运行时差分决定，P3 由结构检查决定，P4 对设计为身份/溯源机制的原语用运行时差分决定，对 LangGraph 则用结构检查决定。接入一个新框架只需要一个 adapter 来驱动其 delegation primitive；保留的 OpenAI Agents SDK（§4.2）就是工作示例：一个 adapter，所有原生通道不 conformant，out-of-band 缓解通道 conformant。

唯一 conformant 的一行不是对我们自定义通道的过拟合。§2.1 的正控机制（OAuth on-behalf-of、macaroons、`SECURITY INVOKER`）在各自原生语境中满足同一个 P1--P4 合取。一个 conformant 判定只是必要筛选，不是完整可靠性证明：它只对 adapter 驱动的那个原生通道判定 P1--P4；它可能放过条件安全通道（P3/P4 只通过外部决策点或某个配置满足）；并且有些结构单元格来自代码阅读，而不是一次执行出的拒绝。它的收益在于定位缺失性质。由于每个性质都映射到 §3.4 的三个 seam 之一，输出告诉维护者在哪里修，而不只是说通道失败。

7.2 最小缓解是结构性的，不是微不足道的

同一 harness 验证了最小的 conformant 通道：一个 out-of-band initiator reference，由运行时设置，指向真正起源，在决策处读取，并按 $\min(\text{deputy}, \text{init})$ 衰减（P1--P4）。接入它会在三个框架的 `patched-composed` 单元格中恢复正确拒绝（§4.3），也是一致性测试发现的唯一满足四性质的通道（§5）。它抵抗与击破 in-band provenance 相同的 forged origin：因为该引用由运行时设置且完整性受保护，攻击者陈述的文本不能移动它（P2 差分），所以它的安全性覆盖本文威胁模型中的对抗性输入，而不仅是良性一致性。

审稿人可能会把这个修复读成微不足道的“传一个字段”。它不是。正确衰减要求在同样三个 seam 上改变系统：在信任边界建立通道，跨组合边界保持它，在代理者或资源侧按 initiator 做决策或下调凭据。没有任何一个 seam 有原生 API 或配置开关可直接打开；这也解释了为什么没有框架默认支付这笔成本，而 census（§6.2）找不到已经拥有该通道的默认实现。

我们有意**不**提出新的委托协议。已有工作已经设计了完整性受保护的 substrate（capability token、

signed hop chain、trust layer，见 §8）；这些机制都预设本文证明为缺失的通道。本文贡献是给出这个缺失的实证前提，以及一个最小验证器。

7.3 意义：一个默认激活的缺口

审稿人可能接受机制，但追问：在良性模型下，低权限到高权限的路由难道不就是预期委托吗？是的，但它静默携带的权限放大才是安全事件：只控制不可信输入的一方，会驱动一个自己无法直接调用的受保护资源操作，把资源所有者的策略扩张为任何能到达特权代理者的人（这正是最小权限原则要阻止的放大，§2.3）。这不同于被授权的升级：后者是显式策略把代理者权限授予发起者；本文缺陷是没有通道可用于做这个条件决策，所以放大是无条件且静默的。

我们从三个层面量化代价。**幅度（在本文 fixture 上）：** 在单个资源的授权域内，组合把发起者的有效权限提升到代理者权限，使三个受保护资源上的可达特权操作集合放大 2--3×（图 7）；新增可达操作恰好是 $(\text{init}, \text{deputy}]$ lattice tier band，且没有超过这个 band（§2.3），所以我们不做生态级估计。**广度（已有 witness）：** 这个结构已经成为已发布平台的默认架构，包括 ServiceNow Now Assist（§1.1）、Microsoft Copilot Studio 和 Salesforce Agentforce。**轨迹（趋同）：** 多个独立努力正在竞相构建 P4 所指的衰减 substrate；目标正确，但还不是默认。

诚实的主张不是某个产品今天可被利用，而是缺口已经默认激活且正在扩张：结构部署得比能使它安全的通道更快。因此真正重要的贡献，是在这个缺口变成事件类别之前就可运行的检查（§7.1）。研究开放框架而不是黑盒产品也是有意选择：产品黑盒探测也许能复现 `ALLOW/DENY`，但不能隔离 P1--P4 中哪一个缺失，而这正是本文贡献；同时黑盒探测还会重新引入我们方法刻意移除的模型混杂因素。

8. 相关工作

一句话差异（§1.4）：既有工作主张应有 authority-bearing channel；我们在实现层测量到它默认不存在。我们把本文置于五条研究线中，包括近期关于 LLM agent 与 multi-agent security 的综述 [11, 21, 27]。这些工作收敛到同一个信息：安全组合需要完整性受保护的权限 substrate；但它们设计或假设该 substrate，而不是测量它是否在框架默认实现中存在。本文视角位于 **(object)** orchestration-framework implementation code、**(method)** deterministic isolated-vs-composed differential、以及 **(phenomenon)** principal substitution 的交点；附录 D（表 10）把每条线相对这个缺口定位。

概念框架。 Shi 等人的 Trust-Authorization Mismatch SoK [50] 用 Belief--Intention--Permission 视角，把静态权限与运行时可信度脱钩这一广泛威胁面统一起来，我们采用其词汇。它工作在不同层次：一个概率模型，明确排除 multi-agent coordination frameworks，并把 composition boundary 的实现层实证列为开放问题。本文处理的正是这个问题：在确定性场景中，同一个任务隔离执行安全、组合后不安全，其根因是 authority-bearing provenance loss → principal substitution。并行的概念工作多停留在治理或需求层，而不是已发布边界上的 authority-carrying channel [41, 52, 54, 62]。

权限升级与 confused-deputy 防御。 一条防御线直接处理 agent over-privilege：PFI [20] 区分 trusted 与 untrusted agents，Progent [51] 通过 tool-level policy DSL 强制最小权限，SEAgent [18] 构建 MAC/ABAC 框架，并且与本文最相关的是，它在真实 MAS runtime 上复现了 multi-agent confused-deputy PoC。我们**承认** multi-agent confused-deputy 现象发现的新颖性属于 SEAgent：它已经展示该行为。它问的是另一个问题，即给定 principal 后如何执行策略；其防御通过外部图重建 provenance。本文问的是更前置的问题：发起者是否在边界后仍然幸存，而这种重建正预设了这一点。

Provenance- and identity-aware authorization protocols。 互补的一条线为委托提出密码学 substrate：AuthGraph [56] 对齐 reasoning graph 与 authorization graph 以检测间接 prompt injection；AIP [43] 引入 invocation-bound capability token，把身份、衰减和 provenance 融合到 MCP/A2A；HDP [9] 把每一跳记录为 Ed25519 签名链路；KYA [45] 增加框架无关 trust layer；相关身份标准努力也从 advisory identity 走向 authenticated delegation [1, 4, 28, 35, 36, 53]。这些工作验证了我们的前提；本文位于更低一层：用实现层差分测量主流框架是否默认携带这些 substrate 所预设的通道（§5）。P3/P4 的 consumer-side enforcement 本身是标准系统安全实践：OS provenance、Linux credential passing 与 ambient capabilities、超越 OAuth on-behalf-of 的多跳微服务身份传播。P1--P4 是把既有实践投射到 agent boundary，而不是重新发明它。

组合诱发与多工具漏洞。 ChainFuzzer [58] 通过 greybox fuzzing 发现 workflow vulnerabilities，它关注的是 tool-to-tool dataflow taint：数据是否到达危险 sink，并没有 principal 概念。prompt-injection 与 tool-agent benchmarks 也类似 [10, 46, 63, 65]。最接近的检测线是 taint-style RCE discovery：LLMSmith [30] 恢复 source-to-sink 链并构造 exploit prompts（11 个 CVE），AgentFuzz [29] 使用 directed fuzzing（23 个 CVE）。这些工作研究的是对抗 payload 下的 data→sink 路径；本文研究良性输入下的 initiator→deputy 权限路径，访问控制本身正确但执行在错误主体上。隐私/抽取风险 [6] 与此正交。

协议与架构级评估。 三项评估工作相邻：AgentRFC [66] 把 MCP/A2A/ANP/ACP specifications 建模为 TLA$^+$ invariants（不是框架代码；实现测试仍在进行）；Agent-Fence [44] 在 Wrong-Principal Action

谓词下为 agent archetypes 打分；Agentproof [60] 静态验证 LangGraph/CrewAI/AutoGen/ADK 上的 workflow-graph topology；更广泛的 taxonomy 工作则盘点系统级威胁 [37]。我们**承认**术语重叠：principal substitution 是 Agent-Fence wrong-principal 类别的一个实现层子例。但本文位于更低一层：Agent-Fence 衡量 archetype break-rate，不能说明 P1--P4 中哪一个缺失；本文提供逐性质归因（完整 object × method × causal-claim 比较见附录 D 表 10）。

9. 结论

隔离状态下安全的 agents，可能仅因被组合而不安全：框架组合边界丢失发起者权限，使受保护资源处一个正确、未被绕过的检查针对 deputy 而非 initiator 执行，即 principal substitution。没有已发布身份原语携带 authority channel 必须共同满足的四个性质；这种 false assurance 在样本外得到确认，并且在我们的 census 中表现为默认缺失。因此，我们把检查打包为可运行 conformance test，而不是提出新协议。核心教训是：验证合取，而不是相信字段。

伦理考虑

本文是设计层测量，不是针对已部署产品的 exploit。在完全良性的威胁模型下（无恶意 agent、无注入 payload），我们展示主流框架在组合边界缺少承载发起者权限的通道。利益相关方包括框架维护者、构建在其上的开发者，以及数据位于受保护资源之后的最终用户；因此我们把工作组织为诊断和改进依据，而不是攻击手册。

无武器化 artifact。PoC 只在我们控制的真实但非生产授权后端上，以良性输入运行公开框架代码：本地 PostgreSQL RLS、两个 OS uid 下的 kernel POSIX ACL、私有 GitHub 测试仓库。In-vivo 检查也使用无注入、无 jailbreak 的良性请求。没有 PoC 针对第三方 live system，没有 exploit payload，也没有超过自有 fixture 中 canary 的外泄。

披露。因为我们命名具体框架，会在提交前通知 AutoGen、CrewAI 维护者，并共享 conformance test。我们将其视为 courtesy notification，而不是需要 embargo 的离散漏洞报告：其一，这是设计默认，不是有单个补丁的内存安全 bug；其二，相关缺口已经可从文档、issue tracker、已发布但部分的身份原语以及 ServiceNow 事件中看到。

总体评估。四性质 conformance test 与最小 conformant channel 能帮助维护者和部署者检测并关闭缺口；发布它利大于弊。本文作为测量与诊断报告，而非针对单一厂商的漏洞报告。

开放科学

我们承诺发布复现本文主张所需的全部 artifact。artifact 包括：CrewAI、LangGraph、AutoGen 的逐框架确定性组合四元组，以及保留 OpenAI Agents SDK 预测 harness；forgeability（P2）与 multi-hop origination（P1）差分，以及单变量 P3/P4 消融；输出逐性质矩阵的框架无关 conformance test；真实后端 harness（PostgreSQL RLS、GitHub fine-grained PAT）和 kernel POSIX ACL 设置；prevalence-census scanner、non-test site extractor、adjudication codebook 与 inter-rater classifier；以及 in-vivo harness、逐试验 transcript 和解析出的 model fingerprint。每个被测框架都 pin 到具体 commit 或 release（表 1，附录 A.1）。

提交时这些 artifact 作为匿名补充归档提供（双盲审稿期间隐藏链接）；接收后将在公共仓库以开源许可证发布。确定性组件首次运行即可复现（exit 0），除 pinned framework source 外无外部依赖。In-vivo 检查依赖我们不控制的第三方生产模型 API；因此我们包含记录下来的逐试验 transcript 与 model fingerprint（附录 B），使报告率即使在 live re-execution 受 provider model drift 影响时仍可检查。

References

[1] A2A Protocol. Agent2agent protocol specification. https://a2a-protocol.org/latest/specification/, 2026. Official protocol specification; accessed 2026-06.

[2] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems, 15(4):706--734, 1993.

[3] AppOmni AO Labs and Aaron Costello. When AI turns on its team: Exploiting agent-to-agent discovery via prompt injection. https://appomni.com/ao-labs/ai-agent-to-agent-discovery-prompt-injection/, November 2025. Accessed 2026-06.

[4] Varun Pratap Bhardwaj. Formal analysis and supply chain security for agentic AI skills, 2026.

[5] Arnar Birgisson, Joe Gibbs Politz, Ulfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In Proceedings of the Network and Distributed System Security Symposium (NDSS), 2014.

[6] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In 30th USENIX Security Symposium, 2021. arXiv:2012.07805.

[7] CrewAI. Issue #5360: Cryptographic identity for agents. https://github.com/crewAIInc/crewAI/issues/5360, 2026. GitHub issue; accessed 2026-06.

[8] CrewAI. Tool hooks. https://docs.crewai.com/en/learn/tool-hooks, 2026. Official documentation; accessed 2026-06.

[9] Asiri Dalugoda. HDP: A lightweight cryptographic protocol for human delegation provenance in agentic AI systems, 2026. Helixar Limited; also published as IETF Internet-Draft draft-helixar-hdp-agentic-delegation-00.

[10] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In Advances in Neural Information Processing Systems (NeurIPS), 2024. arXiv: 2406.13352.

[11] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. AI agents under threat: A survey of key security challenges and future pathways. ACM Computing Surveys, 57(7), 2025. Article 182; arXiv:2406.02630.

[12] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. Communications of the ACM, 9(3): 143--155, 1966.

[13] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. RFC 2693, IETF, 1999.

[14] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec), 2023. arXiv:2302.12173.

[15] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI), 2024. arXiv: 2402.01680.

[16] Dick Hardt. The OAuth 2.0 authorization framework. RFC 6749, IETF, 2012.

[17] Norm Hardy. The confused deputy: (or why capabilities might have been invented). ACM SIGOPS Operating Systems Review, 22(4):36--38, 1988.

[18] Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, Yudong Gao, Shuai Wang, and Yingjiu Li. Taming various privilege escalation in LLM-based agent systems: A mandatory access control framework, 2026.

[19] Michael B. Jones, Anthony Nadalin, Brian Campbell, John Bradley, and Chuck Mortimore. OAuth 2.0 token exchange. RFC 8693, IETF, 2020.

[20] Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. Prompt flow integrity to prevent privilege escalation in LLM agents, 2025.

[21] Juhee Kim, Xiaoyuan Liu, Zhun Wang, Shi Qiu, Bo Li, Wenbo Guo, and Dawn Song. The attack and defense landscape of agentic AI: A comprehensive survey, 2026. Accepted to USENIX Security 2026.

[22] Butler W. Lampson. Protection. ACM SIGOPS Operating Systems Review, 8(1):18--24, 1974. Originally Proc. 5th Princeton Symp. on Information Sciences and Systems, 1971.

[23] LangChain. LangGraph low-level concepts. https://langchain-ai.github.io/langgraph/concepts/low_level/, 2026. Official LangGraph documentation; accessed 2026-06.

[24] LangChain. LangGraph multi-agent systems. https://langchain-ai.github.io/langgraph/concepts/multi_agent/, 2026. Official LangGraph documentation; accessed 2026-06.

[25] Henry M. Levy. Capability-Based Computer Systems. Digital Press, 1984.

[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin,

Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Advances in Neural Information Processing Systems (NeurIPS), 2020. arXiv:2005.11401.

[27] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal LLM agents: Insights and survey about the capability, efficiency and security, 2024.

[28] Zibin Lin, Shengli Zhang, Guofu Liao, Dacheng Tao, and Taotao Wang. Binding agent ID: Unleashing the power of AI agents with accountability and credibility, 2025.

[29] Fengyu Liu, Yuan Zhang, Jiaqi Luo, Jiarun Dai, Tian Chen, Letian Yuan, Zhengmin Yu, Youkun Shi, Ke Li, Chengyuan Zhou, Min Yang, and Hao Chen. Make agent defeat agent: Automatic detection of taint-style vulnerabilities in LLM-based agents. In 34th USENIX Security Symposium, 2025. https://www.usenix.org/conference/usenixsecurity25/presentation/liu-fengyu.

[30] Tong Liu, Zizhuang Deng, Guozhu Meng, Yuekang Li, and Kai Chen. Demystifying RCE vulnerabilities in LLM-integrated apps. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 1716--1730, 2024. arXiv:2309.02926.

[31] Microsoft AutoGen. GroupChat reference. https://microsoft.github.io/autogen/0.2/docs/reference/agentchat/groupchat/, 2024. Official AutoGen 0.2 documentation; accessed 2026-06.

[32] Microsoft AutoGen. Discussion #7432: Message source enforcement. https://github.com/microsoft/autogen/discussions/7432, 2026. GitHub discussion; accessed 2026-06.

[33] Mark S. Miller, Ka-Ping Yee, and Jonathan Shapiro. Capability myths demolished. Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003.

[34] Mark Samuel Miller. Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. PhD thesis, Johns Hopkins University, 2006.

[35] Model Context Protocol. Specification. https://modelcontextprotocol.io/specification/, 2026. Official protocol specification; accessed 2026-06.

[36] Subramanya Nagabhushanaradhya. OpenID connect for agents (OIDC-A) 1.0: A standard extension for LLM-based agent identity and authorization, 2025.

[37] Tam Nguyen, Moses Ndebugre, and Dheeraj Arremsetty. Security considerations for multi-agent systems, 2026.

[38] OpenAI. GPT-4 technical report, 2023.

[39] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Advances in Neural Information Processing Systems (NeurIPS), 2022. arXiv:2203.02155.

[40] OWASP GenAI Security Project. OWASP top 10 for LLM applications 2025. https://genai.owasp.org/llm-top-10/, 2025. LLM01 Prompt Injection; LLM05 Improper Output Handling; LLM06 Excessive Agency; accessed 2026-06.

[41] KrishnaSaiReddy Patil. SentinelAgent: Intent-verified delegation chains for securing federal multi-agent AI systems, 2026. HIGH novelty-collision; new defense protocol + DelegationBench, presupposes the channel we show absent.

[42] Fabio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models, 2022. NeurIPS 2022 ML Safety Workshop.

[43] Sunil Prakash. AIP: Agent identity protocol for verifiable delegation across MCP and A2A, 2026.

[44] Sai Puppala, Ismail Hossain, Md Jahangir Alam, Yoonpyo Lee, Jay Yoo, Tanzim Ahad, Syed Bahauddin Alam, and Sajedul Talukder. Agent-fence: Mapping security vulnerabilities across deep research agents, 2026.

[45] Kolawole Quadri. KYA: A framework-agnostic trust layer for autonomous systems with verifiable provenance and hierarchical policy composition, 2026.

[46] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In International Conference on Learning Representations (ICLR), 2024. ToolEmu; arXiv: 2309.15817.

[47] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278--1308, 1975.

[48] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. IEEE Computer, 29(2): 38--47, 1996.

[49] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Advances in Neural Information Processing Systems (NeurIPS), 2023. arXiv:2302.04761.

[50] Guanquan Shi et al. SoK: Trust-authorization mismatch in LLM agent interactions, 2025.

[51] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2025.

[52] Vincent Siu, Jingxuan He, Kyle Montgomery, Zhun Wang, Neil Gong, Chenguang Wang, and Dawn Song. A framework for formalizing LLM agent security, 2026. terminological ``four properties" clash; disambiguate from our four channel-properties.

[53] Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, Cedric Deslandes Whitney, Dazza Greenwood, Alan Chan, and Alex Pentland. Authenticated delegation and authorized AI agents, 2025.

[54] Krti Tallam. Authorization propagation in multi-agent AI systems: Identity governance as infrastructure, 2026. MED-HIGH collision: ``not reducible to prompt injection"; benign behavior already triggers failures; requirements-level, no differential/failed-fix.

[55] The Hacker News. ServiceNow AI agents can be tricked into acting against each other via second-order prompts. https://thehackernews.com/2025/11/servicenow-ai-agents-can-be-tricked.html, November 2025. Accessed 2026-06.

[56] Peiran Wang, Ying Li, and Yuan Tian. Aligning provenance with authorization: A dual-graph defense for LLM agents, 2026. UCLA; arXiv:2605.26497v1 [cs.CR], 26 May 2026.

[57] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Advances in Neural Information Processing Systems (NeurIPS), 2022. arXiv: 2201.11903.

[58] Jiangrong Wu, Zitong Yao, Yuhong Nan, and Zibin Zheng. ChainFuzzer: Greybox fuzzing for workflow-level multi-tool vulnerabilities in LLM agents, 2026.

[59] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed H. Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation, 2023.

[60] Melwin Xavier, Vaisakh M A, Melveena Jolly, and Midhun Xavier. Agentproof: Static verification of agent workflow graphs, 2026.

[61] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, et al. The rise and potential of large language model based agents: A survey, 2023.

[62] Zijie Xu, Minfeng Qi, Shiqing Wu, Lefeng Zhang, Qiwen Wei, Han He, and Ningran Li. The trust paradox in LLM-based multi-agent systems, 2025. Full title: The Trust Paradox in LLM-Based Multi-Agent Systems: When Collaboration Becomes a Security Vulnerability.

[63] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024.

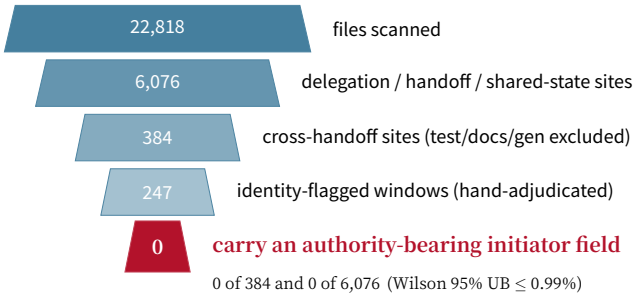


图 4: Prevalence census。在 22,818 个文件和 6,076 个 delegation sites 中，没有 site 携带 authority-bearing initiator field。两个分母都为零；Wilson 95% upper bound ≤ 0.99%。颜色加深表示逐步过滤；红色终点是结果：每个审计过的 cross-handoff site 都可能有 name/identity 字段，但**零个**携带 initiator authority。

[64] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR), 2023. arXiv:2210.03629.

[65] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In Findings of the Association for Computational Linguistics: ACL 2024, pages 10471--10506, 2024. arXiv:2403.02691.

[66] Shenghan Zheng and Qifan Zhang. AgentRFC: Security design principles and conformance testing for agent protocols, 2026.

附录 A. Prevalence census：可复现测量协议

本附录足够详细地说明 §6.2 的 prevalence census，使其可以重新运行。目的不是把结果呈现为一次 keyword scan，而是提升为一个 corpus、提取词表、判定规则和不确定性都明确且可独立检查的测量。脚本和原始输出位于 `experiments/w3-corpus-prevalence/`，包括 scan、non-test census、sampling、inter-rater 脚本，以及 results、summary 和对应 dumps。

A.1 Corpus 选择

Corpus 是 13 个固定仓库，来自已经克隆的主流 multi-agent frameworks 和知名 end-to-end multi-agent applications；我们不新增 clone。因此它是 **mainstream-skewed convenience sample，而不是均匀 GitHub 抽样**。prevalence 数字应解读为“在主流框架和代表性真实应用中”，而不是生态级统计推断（附录 C 也明确这一点）。纳入要求为：仓库 (i) 是 multi-agent orchestration framework 或已部署的 multi-agent application；(ii) 公开，且可从 Python、TypeScript/JavaScript 源码构建；(iii) 可 pin 到具体 commit，使测量稳定。每个仓库都记录 category、pinned commit、commit date、扫描扩展名和实际遍历文件数。

扩展名按项目选择：纯 Python 框架扫描 `.py`；full-stack 应用额外扫描 TypeScript/JavaScript 扩展。我们没有对文件或仓库设置 top-N cap：每个纳入仓库中所有目标扩展名文件都被遍历。目录剪枝只移除常规 vendored、generated 或 cache trees（例如 Git、node_modules、build/ dist、Python cache、virtualenv、Next.js output、vendor/ site-packages、mypy/pytest cache 和 migrations）；这些目录不包含框架 delegation logic。严格分母还通过明确审计过的 substring 规则排除 test/docs/examples/generated 路径；被排除的 1,842 个 handoff-type sites 被报告，而不是静默丢弃。

A.2 Site 提取词表

Delegation/ handoff/ shared-state sites（粗分母）。一个 site 是匹配 21 个 word-anchored pattern 之一的源码行；所有 pattern 都用 `\b` 锚定以避免 substring hits，并按组合机制分组：

- **delegation**（task/ work delegation）：delegate、allow-delegation、delegate-work tool、coworker、sub-agent 和 spawn-agent 相关 token。
- **handoff**（control transfer）：handoff、transfer-to/ create-handoff 系列 token、以及 goto command。
- **shared-state**（state-object construction/ merge）：message merge、state graph 和 send token。
- **serialization / message construction**（task-object reconstruction、role-rebinding、nested chats）：task/ crew construction、group chat、conversable agent、initiate-chat 和 nested-chat registration token。

这个词表在 corpus 中得到 **6,076** 个 delegation sites，即粗分母。匹配广泛命中野外代码中的 state graph、task construction、message merge、group chat、send/ initiate-chat、handoff、delegate/coworker 等 idiom，确认该结构是常见的，而不是我们构造出来的罕见形式。

Handoff-type call sites（严格分母）。严格 census 只保留 14 个表示真实 task/ control

repository	category	commit	commit date	extensions	files scanned
crewAI	framework	`051fa0c`	2026-06-03	.py	1,207
autogen	framework	`7cd0b10`	2026-06-05	.py	1,601
langgraph	framework	`43682f0`	2026-06-03	.py	445
metagpt	framework	`11cdf46`	2026-01-21	.py	890
superagi	framework	`c3c1982`	2025-01-22	.py	426
camel-ai	framework	`cc2de7a`	2026-06-02	.py	1,130
langroid	framework	`763d5cb`	2026-05-21	.py	424
autogpt	application	`2ca389e`	2026-06-02	.py/.ts/.tsx	2,796
dify	application	`c8abb11`	2026-06-04	.py/.ts/.tsx	9,523
openhands	application	`64ae075`	2026-06-04	.py/.ts/.tsx	1,904
taskweaver	application	`d44ddef`	2026-03-23	.py	167
flowise	application	`a4c4e49`	2026-06-04	.ts/.tsx	1,225
anythingllm	application	`43e0d9a`	2026-06-03	.js/.jsx	1,080
total					**22,818**

表 6: 每个仓库的 audited cross-handoff sites：delegation site 数和 authority-carrying 数。结果列全为 0。

repository	files	delegation sites	authority-carrying
crewAI	1,207	1,184	0
autogen	1,601	2,868	0
langgraph	445	1,292	0
metagpt	890	47	0
superagi	426	0	0
camel-ai	1,130	122	0
langroid	424	452	0
autogpt	2,796	56	0
dify	9,523	31	0
openhands	1,904	11	0
taskweaver	167	0	0
flowise	1,225	7	0
anythingllm	1,080	6	0
total	**22,818**	**6,076**	**0**

transfer 到下游 agent 的 pattern，并有意丢弃会膨胀粗分母但不是 authority-carrying call sites 的裸类型引用/import（例如 state graph、conversable agent、group chat 类型名）。严格 pattern 覆盖 delegate/ handoff/ transfer/ create-handoff、coworker、allow-delegation、delegate-work tool、task construction、goto/ send、initiate/nested chat、sub-agent 和 spawn-agent。经过 A.1 的排除后，剩下 **384** 个 non-test handoff-type sites。

Authority-field vocabulary（严格 16 词）。如果一个 authority/ provenance whole-word token 出现在 site 的 ±10 行窗口中，就把该 site 记为携带 authority field（下界指标）。这 16 个词覆盖 initiator、on-behalf-of、principal/ authority、source/ caller identity、delegator/ originator/ invoker、provenance，以及 requester/source/initiator/caller role 等变体。

Identifier-aware identity/authority family（用于判定的标记）。为避免漏掉非关键词 carrier，每个严格分母窗口都被切分为 identifier subwords（snake_case、camelCase、PascalCase 和 digit splits），并匹配一个扩展 identity/authority family。该 family 覆盖 principal 类词（user、account、owner、identity、subject、actor、requester、caller、initiator、delegator、originator、invoker、sender 等）、session/credential 类词（session、token、JWT、claim、credential、API key、bearer）、authorization 类词（auth/ authz/ authn、permission、scope、role、ACL、policy、privilege、capability）、multi-tenancy 类词（tenant、org、workspace 等）、carrier 类词（context、request、metadata、header、cookie）和 provenance 类词（source、origin、on-behalf、current、delegated、acting）。这把 384 个窗口中的 **247** 个标记为需要人工判定，故意取过宽的网，避免负结果被归因于词表过窄。

A.3 判定 codebook

Positive criterion（必要条件）。一个被标记窗口只有在携带的值同时满足下列条件时，才编码为 **POSITIVE**，即真正的 authority-bearing initiator field：(i) **initiator-bound**：命名起源外部主体，而不是相邻 peer、执行 agent 或 persona/role string；(ii) **runtime-set**：由可信代码作为 handoff call 上的 kwarg 或 dict key 绑定，而不是 comment/docstring 或请求文本；(iii) **integrity-protected**：不

从 attacker-influenceable content 派生；(iv) **sink-consumable**：跨 handoff 传播到下游 authorization decision，而不是只用于 logging/tracing/routing。冻结 codebook 将其操作化为：POSITIVE 当且仅当严格 originating-authority token 在 handoff call 同一窗口中作为代码参数绑定。247 个窗口全部编码为 NEGATIVE。

False-positive families（含代表性片段）。每个被标记但为 negative 的窗口都落入以下重复家族；它们都与身份相邻，但至少失败一个必要条件：

- **Transport/ endpoint authentication**（失败 initiator-bound）：`crewai/ a2a/ utils/ delegation.py: 175,274` 的 `auth: ClientAuthScheme` 认证的是到远端 A2A endpoint 的delegating client，不是到下游受保护资源的外部 initiator。
- **Resource-sandbox inheritance**（失败 initiator-bound，且授予的是更少而非 authority）：`autogpt/.../ execution_context.py` 的 `create_child_context` 给 child agent 一个受限 file store、减少的 budget 和 `parent_agent_id`；这是隔离/预算，不是 initiator principal。
- **Channel/ capability tag and task ownership**（失败 initiator-bound）：`autogen/ beta/ network/.../ delegate.py: 47` 的 `delegate(target, prompt, *, capability=None, …)` 只跨越 `prompt` 自由文本和一个 `capability` channel knob；`beta/ agent.py: 705` 的 `Task(owner_id=self.name, …)` 记录的是 owner agent 自己的名字。
- **Control-flow target / back-pointer**（失败 sink-consumable）：`RevertToInitiatorTarget` 是 group-chat routing target；`self.caller` 是 parent-task 指针，用于 done-signaling。
- **Web-auth endpoint, not an agent handoff**（失败 handoff predicate 本身）：Dify 中 workspace-ownership transfer 的 `transfer_token` 和 OAuth surface gate 的 `bearer/subject` 是 HTTP controllers，携带真实授权，但不跨 agent-to-agent boundary。
- **Advisory name string**（失败 integrity-protected）：`source/sender` 是 message/ event author 的 name strings，由 producing agent 设置；`role` 是 agent persona 或 message-role string。它们正是 §5 证明可伪造的 ambient claims。

低信号剩余项（216 个窗口）主要由 `context`（101；free-text task context）、`user`（43）、`role`（30；message-role/ persona）、`sender`（36；AutoGen message-sender name）、`current`（34；current speaker/ agent control flow）和 `group`（20；group-chat vocabulary）构成；没有一个是 sink-consumable、integrity-protected initiator authority。

A.4 分母、不确定性与 codebook 可复现性

两个分母同时报告以避免选择性呈现：严格分母给出更保守（更宽）的界，因此正文引用它，而不是用更紧的粗分母界遮掩。Wilson bounds 假设独立 Bernoulli trial，但这些 sites 并不独立：384 个严格 sites 聚簇在 13 个仓库内，并且同一框架内部共享 idiom，所以有效样本量小于名义计数，真实区间更宽。因此这些 bounds 应读作主流默认缺失的指示，而不是校准过的生态级比例；定性结果（13 个仓库所有审计 sites 都没有 authority-bearing channel）不依赖该区间。SuperAGI、TaskWeaver、Flowise、AnythingLLM 在我们的严格词表下贡献 0 个 handoff-type sites；这是覆盖事实，被报告而不是折入分母隐藏。

Codebook reproducibility。为说明全负判定不是个人判断，一个第二个、确定性的 codebook classifier 在不知道第一轮逐项 verdict 的情况下重新编码全部 247 个窗口，并注入 15 个合成 authority-carrying handoff sites 作为 positive controls，使 agreement statistic 在本来单类标签下也有定义。严格 codebook 精确复现判定（247/247 corpus windows NEGATIVE，15/15 controls detected；Cohen's $\kappa = 1.0$，Gwet's AC1 $= 1.0$，PABAK $= 1.0$）。故意宽松的 stress variant（只要 handoff 附近有 identity-family token 就 flag）会过度标记 19/247（$\kappa = 0.58$，AC1 $= 0.91$），但每一个都落入 codebook 已记录的 false-positive family，例如 sender、caller/self-caller、capability、source、user 和 subject。因此零结果通过了一个独立严格 coder，不是 lenient coding 的 artifact。我们也提供 shuffled blind sheet 供可选人工二次判定；确定性 codebook 是第二个自动实现，而不是第二个人类 coder，所以 $\kappa = 1.0$ 是按构造得到的 codebook reproducibility 数字，信息量更大的边界是上述 lenient variant 的 $\kappa = 0.58$。

附录 B. In-vivo 可复现元数据

本附录记录复现 §6.1 的 in-vivo experiment 所需元数据，使 `gpt-5.5/deepseek- chat/qwen- max` 结果对应公开可解析 endpoint，而不是软证据。原始逐试验 transcripts 和逐模型 protected-resource ledgers 位于 `experiments/ s2- invivo/ results- {gpt5.5,deepseek,qwen}.json`。每个模型都只通过提供商自己的官方 first-party API 调用；我们不控制任何上游服务，因此独立复现者可以通过同样 endpoint 重新运行实验。

Call-log summary。每次运行前，我们发出最小 probe 来确认 endpoint，并确认没有隐藏 system prompt 注入：bare probe 消耗 11--15 个 prompt tokens（见上表），与我们的消息内容本身一致；每个

分母	sites	authority-carrying	carrier rate	Wilson 95% upper bound
coarse keyword delegation sites	6,076	0	0.00%	$\leq 0.061\%$
strict non-test handoff-type sites	384	0	0.00%	$\leq 0.99\%$

表 7: In vivo：良性输入下的主体替换（§6.1）。每个 fire 都是低权限方发起、但由高权限代理者执行的敏感导出（每个场景十次试验）。

model (provider)	endpoint	salaries	payments	admins	overall
gpt-5.5 (OpenAI)	provider official API (`api.openai.com`)	10/10	10/10	10/10	**30/30**
deepseek-chat (DeepSeek)	provider official API (`api.deepseek.com`)	10/10	10/10	10/10	**30/30**
qwen-max (Alibaba)	provider official API (DashScope)	10/10	10/10	9/10	**29/30**

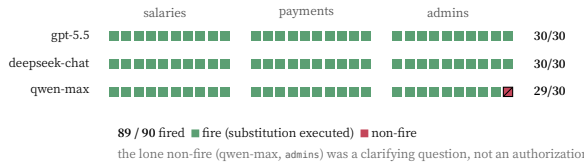


图 5: In-vivo 每次试验的 fire 情况（表 7 的可视化）：三个生产模型 × 三个良性场景 × 十次试验。每个 cell 是一次 trial；绿色表示敏感导出在高权限代理者处触发（principal substitution），红色表示未触发。良性输入、无注入情况下，普通模型在 **90 次中 89 次**走上框架允许路径。

表 8: In-vivo 端点和复现元数据（§6.1）。每个模型都通过提供商自己的 first-party API 到达；bare-probe token count 与我们的消息内容一致，用于确认没有隐藏 system prompt。逐场景 fires 见表 7。

model (provider)	requested id	TLS terminus	probe tok.
gpt-5.5 (OpenAI)	`gpt-5.5-2026-04-23`	`api.openai.com`	14
deepseek-chat (DeepSeek)	`deepseek-chat`	`api.deepseek.com`	11
qwen-max (Alibaba)	`qwen-max`	`dashscope.aliyuncs.com`	15

endpoint 都用标准 usage accounting 返回模型标识。OpenAI endpoint 通过 HTTPS forward proxy 访问以满足地域网络条件，但 TLS terminates at `api.openai.com`，所以中间层不能观察或修改 payload；DeepSeek 和 DashScope endpoint 直接访问。唯一 non-fire（qwen-max，`admin_users` 场景，9/10）是一个澄清问题，不是授权拒绝；该 trial 中框架仍然没有携带 initiator channel（§6.1）。

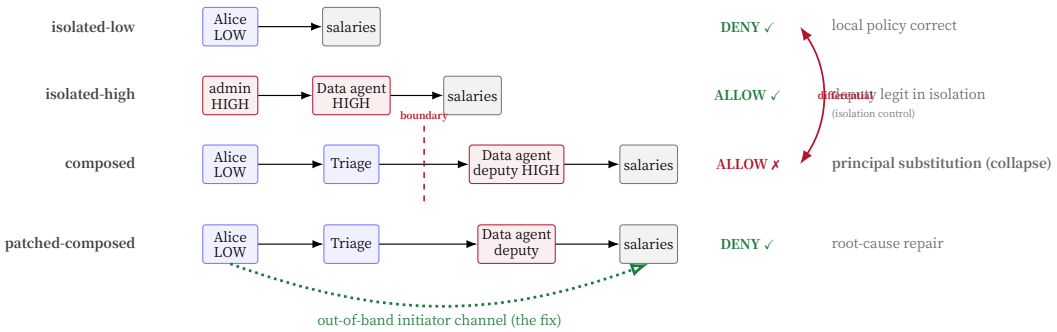
Snapshot pinning（诚实 caveat）。 三个 provider 对 served model 的 pin 精度不同。OpenAI 暴露 **date-pinned snapshot** `gpt-5.5-2026-04-23`，response 会原样 echo。`deepseek-chat` 是 rolling alias，API response 解析为 `deepseek-v4-flash`（system fingerprint `fp_8b330d02d0_prod0820_fp8_kvcache_20260402`，由 2026-06-10 metadata probe 记录）；DeepSeek 不提供可请求的 dated snapshot，因此我们报告解析出的模型标识和 fingerprint。`qwen-max` 也是 rolling alias，DashScope OpenAI-compatible endpoint 只 echo alias，所以原始运行的精确解析没有记录；但 Qwen-max family 发布 date-pinned snapshots。为证明 pinned model 上可复现，我们对 `qwen-max` family 的 date-pinned `max`-tier snapshot **`qwen3.7-max-2026-06-08`**（与原始运行日期匹配，同属 max tier；我们不能确认与 alias 未记录解析完全 byte-identical）重跑 Qwen leg，得到 **28/30** substitutions（salaries 9/10、payments 9/10、admins 10/10）。这与 29/30 alias run 一致，说明 rate 在 date-pinned artifact 上稳定（`experiments/s2-invivo/results-qwen-3.7max-2026-06-08.json`）。最小 per-provider metadata probe 记录 resolved model identifier、provider-stamped `created` timestamp 和 system fingerprint，位于 `experiments/s2-invivo/endpoint-metadata.json`。

Call date。 逐 provider artifact mtimes 与 git author-dates 共同约束运行发生在 **2026-06-08**（DeepSeek、Qwen 和最初 gpt-5 run）以及 **2026-06-09**（gpt-5.5 official-endpoint run）（`experiments/s2-invivo/`）；metadata probe 还捕获了 provider-stamped `created` timestamps。主结果 JSON 没有嵌入逐 call wall-clock timestamps；因此我们把这些 artifact/commit dates 作为文档化边界，并会在 camera-ready replication bundle 中从 provider account call logs 补充精确逐 call timestamps。

附录 C. 形式框架：因素、truth table 与命题

本附录给出 §3.4 末尾概述的完整形式框架。

因素。 Composition 因素 $c \in \{\text{ISO-LOW}, \text{ISO-HIGH}, \text{COMPOSED}\}$，表示请求 r 是直接以低/高权限主体到达 sink，还是跨 delegation/handoff boundary 到达。Initiator channel 因素 $\kappa \in \{\varnothing, \text{IN}, \text{OOB}\}$，表示 enforcement point 从何处恢复 initiator：无通道、框架携带的 in-band message/task text，或由可信 runtime 设置且框架不原生提供的 out-of-band channel。谓词 $\text{forgeable}(\kappa)$ 记录 adversary 是否能影响恢复值：$\text{forgeable}(\text{IN}) = \text{true}$（origin



Differential: rows isolated-low vs composed share the same LOW initiator (Alice) and resource (`employee_salaries`); adding composition alone flips DENY→ALLOW (the collapse). isolated-high is the isolation control; patched-composed is the root-cause repair. Color: blue=low-priv, red=high-priv/deputy, green=safe outcome / out-of-band fix.

图 6: 确定性差分。Isolated-low 与 composed 共享同一个低权限 initiator 和 sink；仅加入 composition 就把 DENY 翻转为 ALLOW（主体替换）。Isolated-high 是隔离控制；out-of-band initiator channel（patched-composed）恢复 DENY。

claims 按 §2.2 作为 attacker-influenceable text 到达），forgeable(OOB) = false（由 runtime 从真实 session 设置）。

固定装置。 模型是 §3.1 的 dutiful relay R：对任意声明 origin o（真实或伪造），它都发出同一个下游操作，即 $R(o) = op$ 并且对 o 为常量。这就是“移除模型混杂因素”的形式含义：relay 不可能成为解释 composed-only failure 的变量。代理者在所有单元格中应用同一个最小权限决策规则：

$$d_\kappa(r) = \text{sink\_allows}\big(\min(\text{authority}(\text{deputy}),\ \text{authority}(\text{claim}_\kappa(r)))\big),$$

其中 $\text{claim}_\kappa(r)$ 是通过通道 κ 恢复的 initiator，且 $\text{claim}_\varnothing(r) = \text{deputy}$（框架默认：没有通道时无从衰减到发起者，所以由 deputy 自身权限支配）。sink_allows 是平台原生 sink check，整个实验中保持不变。

Truth table（单个 sink；init 低权限，deputy 高权限；op 需要高权限）：

第 3→4 行只改变声明的 origin（c、κ 和 deputy code 都固定）；第 4→5 行只改变通道 κ。单变量对比使归因成为机械判断，而不是解释性判断。

命题 1（存在性：由定义蕴含，而非发现）。 在 relay R 和框架默认 $\kappa = \varnothing$ 下，规则化简为 $d_\varnothing(r) = \text{sink\_allows}(\min(\text{authority}(\text{deputy}), \text{authority}(\text{deputy})))$，所以 COMPOSED ⇒ ALLOW，同时 ISO-LOW ⇒ DENY，从而违反 (†)。这从定义推出：relay 转发，sink 检查 executor，没有通道携带 initiator。因此四元组是 principal substitution 的**展示**，而不是发现（隔离看，它就是 1988 confused deputy）。论文证据重量不在这里，而在命题 2--3；我们显式写出命题 1，是为了把四元组读成controlled witness，而不是读成核心发现。

命题 2（可伪造性，P2）。 固定 c = COMPOSED 与代理者规则，只改变 in-band claim：诚实 origin 得到 DENY，伪造 origin 得到 ALLOW。因为 relay 不随 origin 改变操作，唯一变量是 attacker-stated claim，故 in-band channel 可伪造。把 κ 从 IN 改为 OOB 恢复 DENY，说明维护 (†) 需要运行时设置、完整性保护的通道。

命题 3（起源性，P1）。 在无伪造的 $A \to B \to C$ 链中，运行时把 in-band source 重新盖章为产生该消息的 peer，所以 C 恢复到 B 而不是真正发起者 A，并按 B 权限衰减，可能得到 ALLOW；单跳控制 $A \to C$ 则 DENY。因此失败只在组合中显现。

命题 4（必要性，带完整 realized witnesses；不是不可能性结果）。 P1--P4 各自独立必要；移除每一个都有今天已发布的 primitive 作为 witness。A2A transport identity 和 AutoGen re-stamped `source` 丢 P1，in-band origin claim 丢 P2，CrewAI `fingerprint` 丢 P3，LangGraph `config.configurable`（满足 P1∧P2）丢 P3∧P4。因此在我们的测试中，{P1--P4} 的任何真子集都不能维护 (†)，所以 identity field 的存在是必要但不充分的（false assurance）。这是带 realized counterexamples 的必要性主张，**不是** impossibility theorem：本文 conformant out-of-band channel 和 §2.1 的已执行正控满足完整合取，说明安全通道确实可以存在。

Worked example（一个框架上的 E3 vs. E4）。 在 LangGraph/POSIX 中（initiator 为 `nobody`，deputy 为 `admin`，sink 是 mode-`0600` 文件）：composed 情况下，deputy 用自己的 kernel principal 读取文件 → ALLOW（substitution）。**E3**（P3，policy layer）增加一个 origin-aware check，读取保留的 initiator（`nobody`），并在 kernel read 之前拒绝；deputy credential 不变，原生 ACL 根本不会被触发。**E4**（P4，capability layer）把 deputy 的 effective principal 下调为 $\min(\texttt{admin}, \texttt{nobody}) = \texttt{nobody}$；未改变的 kernel ACL 随后返回 `EACCES`。两者通过不同机制恢复 DENY：一个对 initiator 做决策，另一个收缩 credential；二者都预设框架缺失的 P1∧P2 通道。

Honest limitations。

- **In-band channel 的建模。** 在 harness 中，in-

#	isolated factor	c	κ	stated origin	claim_κ	outcome	decides
1	baseline	ISO-LOW	—	—	init	**DENY**	local policy correct
2	deputy legit in isolation	ISO-HIGH	—	—	deputy	ALLOW	—
3	composition	COMPOSED	IN	honest init	init	DENY	defense appears to work
4	forgeability (vs 3: only origin)	COMPOSED	IN	forged high	high	**ALLOW**	**P2**
5	channel (vs 4: only κ)	COMPOSED	OOB	forged high	init	DENY	**P2**
6	consumer-side repair	COMPOSED	OOB	—	init	DENY	P3 / P4

band initiator claim 从 attacker-influenceable request text 中读出并 plumb 到 sink；该 plumbing 本身不是原生的，因为框架根本不给 sink 接 provenance。安全相关变量是source 的 forgeability，不是 plumbing 机制。

- **Sink fidelity**。三个 sink 都是真实授权后端：LangGraph 使用 kernel POSIX ACL，CrewAI 使用 live PostgreSQL 16 RLS engine，AutoGen 使用 live GitHub REST API 的 fine-grained PAT scopes。论证只用到一个性质，即对执行主体做 enforcement，这是三者的定义性属性。GitHub PAT 还是框架控制之外的外部 SaaS decision point，说明 P3 不需要 per-API shim：`patched-composed` deputy 向资源呈现被 $\min(\text{deputy}, \text{init})$ 缩窄的 credential（低权限 PAT），GitHub 自己返回 `404`（DENY），没有应用层 guard（§4.1）。
- **Determinism vs. realism**。Deterministic relay 是移除模型混杂因素的 controlled witness，不是具体部署 agent 的模型，也不是关于真实 LLM 会泄漏多少的主张。In-vivo confirmation（§6.1）覆盖三个 provider 的三个生产模型、三个良性场景、一个 OAuth-style sink；所有模型都通过 provider 官方 first-party API 到达。它说明普通模型在良性输入下会走框架允许路径，不是 root-cause claim；归因来自 deterministic differential 和 property matrix。
- **Prevalence corpus**。Corpus 是 cloned、mainstream-skewed sample，不是均匀 GitHub 抽样。384 个 non-test cross-handoff sites 的负 census 加上 identifier-aware matching 能界定但不能消除 idiom incompleteness。特别是 initiator 若通过 ± 10 行窗口之外的 runtime context（thread-local、middleware、run-scoped config）携带，会逃过窗口测试；因此我们额外运行 file-level scan，查找 contextvars / thread-local / middleware / run-scoped config 与 handoff、principal token 的共现。它产生 11 个 candidate files，全部按同一 codebook 判为 NEGATIVE，所以 out-of-window 形式是已测量的负结果，不是被构造性排除。
- **Falsifiable boundary**。如果某个框架默认暴露满足 §2.1 四性质的通道，本文主张对它就不成立。我们搜索后没有发现这样的通道；最接近的 protocol-level candidate A2A 认证的是单跳 immediate peer，而非 originating principal；CrewAI Enterprise“RBAC”管的是 human-user permissions，不是 runtime agent-to-agent handoff。
- **Attenuation semantics**。P4 写作 $\min(\text{authority}(\text{deputy}), \text{authority}(\text{init}))$，指资源原生 authority order 中的 greatest-lower-bound（RBAC role rank、POSIX uid、OAuth scope intersection），不是整数算术。更丰富的 ABAC predicates、per-operation obligations 或按 initiator keyed 的 allowlists 都是同一 P4 的有效实例：它们都在决策处消费幸存 initiator，并拒绝 initiator 不能直接调用的操作。
- **Enterprise platforms: witnessed, not measured**。ServiceNow Now Assist 是外部披露、vendor-confirmed incident（§1.1）；Microsoft Copilot Studio 和 Salesforce Agentforce 作为具名默认架构进入讨论。我们没有对这些 closed products 运行差分。黑盒产品探测或许能复现 ALLOW/DENY，但不能隔离 P1--P4 中哪一个缺失，还会重新引入模型混杂因素。
- **修复是 validator，不是 benchmarked protocol**。我们验证 out-of-band channel 的 conformance（P1--P4 恢复正确拒绝），而不是 runtime cost。性能、可部署性和可用性是 delegation protocol 的设计目标；本文有意不提出协议（§7.2）。
- **Provenance lifecycle: synchronous only**。四元组测试覆盖 in-the-moment、single-session、synchronous delegation；不测试 durable boundary 上的 provenance lifecycle，例如 asynchronous/resumable runs 中 initiator 超出 session 生命周期、replay/checkpoint-resume 重新绑定陈旧 initiator reference、以及 recovered initiator 的 tenant isolation。这些是真实 out-of-band channel 的实现问题，会扩展 conformance result（P1--P4），不与它矛盾。

§5 conformance matrix 的证据基础。每个 ✓/✗ 的建立方式如下：runtime 表示由已执行差分决定（§4.4）；structural 表示 native-API / authorization-path 代码分析；doc 表示 primitive 自身规范。

附录 D. 扩展相关工作比较

Protocol/standard efforts 目标覆盖哪些 P1--P4。Failed-fix matrix（§5）报告的是已发布框架默认满

表 9: failed-fix matrix 每个单元格的证据来源：runtime 表示运行时差分，structural 表示 native-API / authorization-path 代码分析，doc 表示原语规范。P2（以及除 A2A 由 doc 决定外的 P1）由 runtime 决定；P3 除 LangGraph runtime probe 外为 structural。**P4 对两个设计为 identity/provenance 机制的原语是 runtime-witnessed**：attenuation differential 把零权限 initiator 放在 deputy 决策点，操作仍为 allow；一行 min(deputy, init) 则翻转为 deny。因此“它不衰减”是执行出来的拒绝证据，不是只从代码读出。P4 只有对 LangGraph opportunistic metadata 和 A2A 才保持 structural/doc。

primitive	P1 orig.	P2 unforg.	P3 reach.	P4 atten.
AutoGen `source`	runtime	runtime	structural	runtime
CrewAI `fingerprint`	runtime	runtime	structural	runtime
A2A transport identity	doc	doc	doc	doc
LangGraph `config.configurable`	runtime	runtime	runtime	structural
out-of-band channel (ours)	runtime	runtime	structural	runtime

图 7: 权限放大：低权限发起者直接可达的特权操作（蓝色）与组合后通过高权限代理者可达的操作（红色新增 band）。在三个 fixture 上，组合把发起者有效可达操作提升 2–3×。该图仅说明本文 fixture，不是生态总体估计。

足什么。为完整性，我们把提出中的 substrates 按设计意图（analytical，不是 runnable framework defaults）映射到同样四个性质；结论与本文 thesis 一致：每个机制都目标覆盖某个子集，并且预设本文测量为缺失的通道，所以没有一个是 deployer 免费获得的默认能力。

开发者今天可以尝试 bolt on OAuth token-exchange / on-behalf-of（RFC 8693）或 attenuated macaroon；失败 seam 正是本文差分隔离出的 seam：如果 in-band 携带，它可由攻击者陈述（失败 P2）；如果作为 framework metadata 携带，它会在边界丢失（P1）或永远不被资源读取（P3）；也没有框架在决策处默认执行 min(deputy, init)（P4）。OAuth/macaroon 在其原生场景中成立，正是因为 token 由 runtime 铸造、完整性受保护、out-of-band 携带，并由 resource server 重新推导 narrowed scope；这就是 §2.1 的 realizability witness。一个 framework-agnostic interposer 若把该 witness out-of-band 携带，不需要改框架源码，但它正是 census 测量为缺失的非默认通道（§6.2）。bolt-on 是默认系统没有支付的成本；一个能工作的 interposer 是“通道存在但未发布为默认”的 realized form，因而确认而不是反驳 default-absence。

表 10: 与相关工作的三维比较：研究对象、方法、因果主张/现象，以及与本文重叠之处。

Work	Object	Method	Causal claim / phenomenon	Overlap with us
Ours	Framework **implementation** paths (CrewAI/ LangGraph/AutoGen: handoff, delegation, serialization)	**Deterministic isolated-vs-composed differential** (benign input, single run)	**Authority-bearing provenance loss → protected-resource RBAC on wrong principal (principal substitution)**	—
SoK Trust-Auth-Mismatch [50]	Survey of 200+ papers; protocols (MCP/A2A) abstractly	Secondary synthesis + B-I-P formal lens + Rice-theorem undecidability	Trust–authorization decoupling (probabilistic, runtime trust)	Root-cause **naming** only; no implementation, no empirics
SEAgent [18]	MAS runtimes (AIOS-AutoGen broadcast)	MAC/ABAC enforcement + PoC case study	Confused deputy via prompt-level persuasion + broadcast topology	**Phenomenon discovery**（本文承认）; not implementation root-cause, not differential
PFI [20] / Progent [51]	Single-agent tool use	Trust separation / policy DSL enforcement	Over-privileged tool calls	Defense; presupposes substrate
AuthGraph [56]	Single agent, single trajectory	Dual-graph alignment defense (AgentDojo)	Indirect prompt injection	Single-trajectory; multi-agent out of scope
AIP [43] / HDP [9] / KYA [45]	MCP/A2A/ cross-framework delegation	Capability tokens / signed hop chains / trust layer	Unverifiable delegation identity & provenance	Defense protocol; **we cite as supporting**, patch ≠ new protocol
ChainFuzzer [58]	Tool-to-tool dataflow	Greybox fuzzing + payload mutation	Source-to-sink dataflow taint reachability	"composition" framing only; no principal/authority
LLMSmith [30]	LLM-app code, source→sink call chains	Static call-chain recovery + prompt exploit	Untrusted data reaches RCE/injection sink	Taint/RCE discovery; no principal, payload-driven
AgentFuzz [29]	20 agent applications (runnable)	Directed greybox fuzzing (non-deterministic)	Taint-style 0-day RCE/ injection exploitability	Vuln discovery via payload; argues against static, no authority notion
AgentRFC [66]	Protocol **specifications**	TLA$^+$ invariants + conformance replay	Spec-level composition safety via abstract bridge	Spec layer; implementation testing "ongoing, no findings"
Agent-Fence [44]	8 agent **archetypes** (black-box)	Mean security break rate, multi-turn adversarial	Authorization Confusion / Wrong-Principal Action	**Terminology**（承认"auth. confusion"）; no root-cause
Agentproof [60]	Workflow **graph topology**	Static graph checks + DFA temporal policies	Structural defects, missing human gates	Same frameworks; topology ≠ authority

表 11：每个 proposed substrate 在设计上目标覆盖哪些 P1–P4（分析性，不是表 4 那样的已发布框架测量）。✓ 表示目标覆盖该性质；“—”表示不处理。每个机制都目标覆盖一个子集，并预设本文测量为缺失的通道。

effort (layer)	P1	P2	P3	P4	note
AIP capability tokens [43]	✓	✓	—	✓	identity, provenance, attenuation；protocol layer，不是 framework default
HDP signed hop chains [9]	✓	✓	—	—	Ed25519 hop provenance；不处理 resource-side attenuation
KYA trust layer [45]	✓	—	✓	—	framework-agnostic interposition，把 identity 携带到 decision
MCP signed tool outputs [35]	—	✓	—	—	保护 output 完整性，不保护 call 处 initiator identity
attestation-based chains	—	✓	—	—	unforgeable hop provenance；P3 只有在资源消费它时才成立