PORTA：身份代理中的系统性漏洞发现

Zixu Zhou
jasonz.zhou@utoronto.ca
多伦多大学 多伦多，安大略省，加拿大

Xiangyu Guo
xiangyu.guo@mail.utoronto.ca
多伦多大学 多伦多，安大略省，加拿大

Ilya Grishchenko
ilya.grishchenko@utoronto.ca
多伦多大学 多伦多，安大略省，加拿大

David Lie
david.lie@utoronto.ca 多伦多大学 多伦多，安大略省，加拿大

arXiv:2611.33015v1 [cs.CR] 24 Nov 2026

摘要

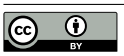
身份代理（如 Keycloak、Vault 和 Dex）会将上游身份提供商提供的身份验证证明转换为下游服务的会话。由于身份代理在身份验证和信任中介中扮演着核心角色，代理中的单个漏洞就可能危及其后所有下游服务。事实上，近期事件表明，即使是广泛采用的平台也无法免受此类问题的影响，这凸显了开展系统性分析的迫切需求。为构建此类分析，本研究首先定义了代理运行的不变量。现有身份验证规范并未完全涵盖这些不变量，主要是因为代理具有独特的双重身份，即同时充当依赖方和身份提供商。这些不变量概括了身份代理在整个工作流中必须遵守的属性；我们考察的此前 60 个通用漏洞披露均至少违反了一项不变量。在这些工作流中，我们还进一步定义了高层级操作类别。基于这些信息，我们的自动化工具 PORTA 利用大语言模型（LLM）生成与高层级操作类别相对应的应用程序接口调用序列，以及与指定不变量相一致的预言机断言。PORTA 在七种流行的开源代理中发现了 33 个此前未报告的安全漏洞。我们研究了这些新漏洞，并将其归纳为八种反复出现的模式。我们已负责任地披露所有发现的漏洞：其中 29 个已获确认，14 个已修复，19 个已分配通用漏洞披露编号，其中 10 个被评为严重。

CCS 概念

• **安全与隐私** → **授权**；**Web 协议安全**；**Web 应用安全**；软件安全工程。

关键词

身份代理、单点登录、OAuth、OpenID Connect、安全测试、安全不变量、身份验证、信任转换、大语言模型



本作品根据知识共享署名 4.0 国际许可协议授权。*CCS '26*，荷兰海牙© 2026 版权所有归所有者/作者所有。ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846557

ACM 参考文献格式：
Zixu Zhou、Xiangyu Guo、Ilya Grishchenko 和 David Lie。2026。PORTA：身份代理中的系统性漏洞发现。发表于*2026*年 *ACM SIGSAC* 计算机与通信安全会议论文集（*CCS '26*），*2026*年*11*月*15*日至*19*日，荷兰海牙。ACM，纽约，纽约州，美国，共15页。https://doi.org/10.1145/3830454.3846557

1 引言

在单点登录（SSO）过程中，企业和云租户不会针对每个身份提供方（IdP）分别对用户进行身份验证，而每个身份提供方可能依赖各自的身份验证协议，例如用于企业联合身份认证的安全断言标记语言（SAML）[27]、用于 Web 和移动端单点登录的 OpenID Connect（OIDC）[32]，以及用于 API授权的开放授权（OAuth）[14]。相反，组织通常会运营一个内部身份提供方，让员工只需进行一次身份验证，之后便可在不同系统中重复使用该身份验证。此过程涉及通过异构协议转发最初的身份验证事件。为此，必须将一个信任域中的认证凭证转换为另一个信任域中的有效会话。

执行这种转换的软件称为*身份代理* [6, 15, 23, 31, 33, 40]。具体而言，代理会接收上游认证凭证（例如 SAML 断言或 OIDC令牌），验证其密码学完整性，提取其中包含的身份和授权声明，将这些声明映射到下游主体，并为下游服务签发新的凭证。

近期的测量研究 [18] 证实，代理式单点登录在互联网上十分普遍，而且单个代理可能同时服务数十个依赖方（即接受该代理令牌的应用程序和服务）。

自然，身份代理中的漏洞会影响所有信任它的下游服务。此外，下游服务无法检测或防御此类故障，因为当它们看到代理的输出时，上游证明已经被替换。因此，代理内部的凭证处理逻辑构成了单点故障。

与此同时，代理操作违背了所有身份验证协议规范中一项至关重要的假设，即
认证协议中的每个实体都被分配一个单一角色。请考虑 SAML规范 [27] 中的一条款：“如果存在 NotOnOrAfter，依赖方必须在该时间之后拒绝该断言。”对于从单一受信任 IdP 获取断言的依赖方（RP）而言，遵循 IdP 选择省略该

字段的做法是正确的。对于代理而言，遵循字段缺失情况会使攻击者能够删除该字段并禁用检查。事实上，身份代理区别于传统依赖方或 IdP 之处在于其双重角色：它同时作为上游 IdP 的依赖方，并作为下游客户端的 IdP。

单一角色假设也被 Fett 等人及其他研究者对身份验证协议的现有分析所采用 [7–9] ，这些分析研究了身份验证协议的安全属性。其他既有工作要么测试单一身份验证协议角色（针对 OAuth 授权服务器的 OAuch [29] ，以及针对应用平台OAuth 的 COVScan [24] ），要么研究代理与其环境之间的外部交互 [18]。

在本研究中，我们将身份代理视为结合了上游依赖方角色与下游身份提供商角色的系统。

首先，我们通过对八份单点登录规范中的 40 项规范性要求以及针对十个代理平台的 60 个既有通用漏洞披露进行汇聚分析，推导出一组五项安全不变量。每项不变量都至少可追溯至一条规范条款，并且至少在两个独立开发的代理中被通用漏洞披露所违反。我们并不声称这五项不变量构成了代理运行正确性的完整定义，而是认为它们有助于检测安全漏洞。

为大规模检测不变量违规，我们设计了一种包含四个阶段的自动化方法：(1) 平台适配 使用大语言模型从代理的 API文档中提取平台配置文件，将抽象操作映射到具体的 HTTP端点及其参数；(2) 序列生成 使用大语言模型根据不变量定义和平台配置文件生成攻击序列和预言机断言；(3) 执行 在隔离的 Docker容器中，针对运行中的代理实例执行每个序列；以及 (4)判定针对捕获的 HTTP响应，以机械方式评估预言机断言。

我们将这种方法实现为一个名为 PORTA 的工具，并在七个平台代理上对其进行评估。

我们首先在先前通用漏洞披露的一个子集上评估 PORTA，选取了 11 个适用的通用漏洞披露，涉及所选的七个平台代理。在三次独立运行中，PORTA 找到了其中 10 个。

我们还开展了漏洞发现活动，以识别此前未知的漏洞。我们采用相同的实验设置进行了三次独立运行，每次均在 3 小时内完成。自动去重前，PORTA 产生了 926 项不变量违规。每次运行内去重后保留 252 项违规，其中经人工审查确认 226 项，驳回 26 项。随后，源代码分析将这 226 项已确认发现映射为三次运行中共 33 个不同的安全缺陷。其中 15 个缺陷违反了认证协议规范中的规范性要求。其余 18 个是具有安全影响的缺陷：虽然不变量遭到违反，但相关规范并未明确要求进行相应检查，这表明双重角色代理场景中存在规范不完整性。

我们研究了新近发现的漏洞，并观察到类似的失效模式：每种模式都在两个到六个由不同团队独立开发、使用四种编程语言编写的代理代码库中反复出现。我们将这些重复出现归因于假设不匹配：易受攻击的代码保留了在以下假设下正确的编码习惯即所有参与方都扮演单一角色；然而，如上文所述，双角色身份代理并非如此。总计，我们确定了八种不同的假设不匹配模式，并为33个发现项中的每一项指定了一种主导模式。

在33项发现结果中，有15项（45%）的根本原因与所选语料库中的任何通用漏洞披露都不匹配，这表明不变量集合能够推广到其推导所依据的通用漏洞披露之外。我们负责任地向受影响的维护者披露了所有违规问题；其中29项已得到确认，14项已修复，19项已分配CVE标识符。其中至少10项被评为严重级别。

总而言之，本文的贡献如下：

- 我们根据规范和60个既有通用漏洞披露，为身份代理双重角色操作推导出一组包含五项安全不变量的集合。
- 我们设计了一种自动化方法，利用大语言模型根据不变量定义生成序列和预言机断言，并针对捕获的平台行为，以确定性方式评估这些断言。
- 我们将该方法实现为工具PORTA，并在七个代理平台上对其进行评估，发现了此前报告的11项适用安全缺陷中的10项，并识别出33项此前未报告的安全缺陷。
- 我们将发现的漏洞归类为八种跨平台失效模式，其根源是对单一角色代理操作的假设。负责任披露促成了 29 项致谢、14 个补丁和 19 个通用漏洞披露编号；其中至少有 10 个属于严重漏洞。

PORTA 的源代码和评估工件已在 [1] 提供。

2 背景

身份代理。 身份代理位于外部身份提供方（IdP）与一个或多个下游服务之间。不同于只使用上游身份证明的普通依赖方，代理既使用上游证明，也重新签发证明：它会验证上游凭证，提取其中携带的身份和授权声明，将这些声明映射到下游域，并为下游服务签署新的凭证。这种双重角色——面向上游 IdP 时充当服务提供商（SP），面向下游服务时充当 IdP——正是代理既强大又脆弱的原因。任一环节出现故障都可被直接利用：验证错误会让攻击者伪造上游证明，而签发错误则可能发放具有过高权限的下游会话。Keycloak、Authentik 和 Dex 等平台广泛承担这一角色，在数十个下游应用之间中介企业单点登录 [18]。

身份代理协议。 代理主要出现在两种协议场景中。在*SAML*代理中，代理从上游身份提供商接收带签名的 SAML 2.0断言，验证其签名和条件，并签发自己的下游断言或 OIDC令牌 [27]。在*OIDC* 和 *JWT* 联邦中，代理通过 OAuth 2.0授权码流程 [14] 获取上游身份令牌，或验证直接提供的 JSON Web 令牌（JWT） [21],，然后签发下游会话。令牌交换 [22]是第三种模式，在这种模式中，代理在单一协议内将一种令牌类型转换为另一种令牌类型。线格式有所不同，但安全义务

是相同的：代理不得接受其无法验证的内容，不得授予未经授权的权限，也不得因协议步骤的顺序安排而受到操纵。图1展示了代理在信任链中的位置及其双重的 SP/IdP角色。上游证明与下游会话之间的阴影区域即为信任转换边界，这正是本文的重点。

3 威胁模型

攻击者能力。 我们考虑三种攻击者位置，每种都反映了联合身份部署中的现实场景。第一，攻击者可能运营一个上游身份提供方，该身份提供方已在代理中注册为受信任的身份源。在多租户代理部署中，任何组织都可以注册自己的身份提供方，因此攻击者使用自己的密钥签署 SAML 断言和 JWT，而代理则配置为信任这些密钥。攻击者不需要恢复属于其他身份提供方的密码学密钥，也不需要伪造其签名；所有攻击凭据都源自攻击者自己的身份提供方。第二，攻击者可能截获一个合法上游证明（例如通过遭入侵的端点或泄露的 URL），并尝试对其进行重放或修改。第三，攻击者可能是一个合法的代理用户，仅拥有有限权限，却试图提升访问权限，例如将窄范围令牌兑换为更广泛的令牌，或访问其他租户的资源。

不在范围内。 我们方法的范围由若干假设决定。代理已正确配置一个或多个合法上游身份提供商的信任锚；攻击者无法读取或修改这些合法身份提供商的配置、密钥或断言。攻击者无法访问代理的数据库、文件系统或内部网络。攻击者无法修改代理的源代码或配置。所有管理员配置均受信任，因管理员配置错误导致的漏洞不在范围内。攻击者无法访问管理 API。例如，如果管理员明确启用宽松设置（例如 trust-email=true 或 validateSignature=false ），则由此产生的行为是设计使然，而非漏洞。此外，我们不测试拒绝服务状况、客户端攻击（跨站脚本攻击、跨站请求伪造、点击劫持）或上游身份提供商自身的漏洞。

部署假设。 我们在每个代理的默认配置下进行测试，并确保管理员已正确配置至少一个上游身份提供商的信任。每次测试运行都从干净状态开始：代理运行在隔离的容器中，该容器会在协议运行之间销毁并重新创建，从而确保不会残留先前测试的状态。

攻击者目标。 所有目标均从信任代理输出的下游服务的视角定义。如果攻击者能够：(a) 冒充另一个主体进行身份验证（身份伪造），(b) 获得在上游证明过期或被撤销后仍然有效的会话（重放），(c) 获得超出上游身份提供方授权范围的权限（权限提升），或 (d) 利用身份冲突接管其他用户的账户（账户接管），则攻击者便达成了目标。

4 代理的信任转换不变量

为了测试身份代理能否正确转换上游域与下游域之间的信任，我们首先需要精确定义身份代理的这种正确性。现有协议规范（如 SAML、OIDC 和 OAuth）定义了各个角色的正确性，即身份提供方必须声明什么，以及依赖方必须验证什么，但并未定义同时承担这两种角色的代理应如何做到正确。因此，代理开发者即使分别按照这些规范正确实现每种角色，最终生成的转换流水线整体仍可能不安全。本节旨在通过一组正确性属性填补这一空白。这些属性 (i) 以所有主要单点登录规范中的具体规范性条款为依据，(ii) 已被多个独立开发的代理中真实存在的通用漏洞披露所违反，且 (iii) 可通过黑盒 API 交互进行测试。

我们将这些属性称为转换不变量。这些不变量在本文后续内容中发挥三种作用。首先，它们明确了代理的正确性要求，为开发者、审计人员和标准组织提供了一份具体的代理测试清单。其次，它们为 PORTA 的设计提供了组织框架（§5）：每个不变量都成为攻击序列的生成目标和判定流水线的测试目标。第三，它们为解读我们的发现结果提供了结构性视角（§7）：我们发现的每一类跨平台故障都对应一个特定的不变量边界，这直接表明该缺口是结构性的，而非偶然现象。

4.1 形式化模型

我们将身份代理建模为一个信任转换函数

$$F : \mathcal{A}_{\text{up}} \times C \longrightarrow \mathcal{A}_{\text{down}} \cup \{\perp\}, \tag{1}$$

其中，A_{up} 是上游凭证工件的空间（例如 SAML 断言、JWT 和证书），A_{down} 是下游会话的空间，C 是代理配置（例如信任锚、声明映射规则和策略约束），$\perp$ 表示拒绝。F 分解为图 2 所示的四个连续阶段：Verify 检查密码学有效性，并依据 C 中的信任锚验证元数据；Extract 从经过验证的工件中提取结构化表示，其中包括主体身份、身份验证上下文和授权声明；Map 根据 C 的映射规则，将该表示转换到下游域；Issue 则对下游凭证进行序列化和签名。

这些阶段之间的每个边界都会引入一项正确性要求，而单角色身份提供方或依赖方实现中并不存在此类要求。我们将这些义务形式化为五个安全不变量。方程 (1) 中的函数 F 表示一个转换跳：一个上游身份提供商、一个代理和一个下游会话。实际部署通常会组合多个转换跳，其中一个代理的输出会成为下一个代理的上游输入。本文将形式化处理范围限定在单跳情形，并将组合分析留待未来工作。我们定义的五个不变量分别适用于每个转换跳，本身并不涵盖跨跳属性，例如传递信任转移。

4.2 推导方法

为推导出有原则依据的不变量集合，我们对两个独立来源进行了汇聚分析。

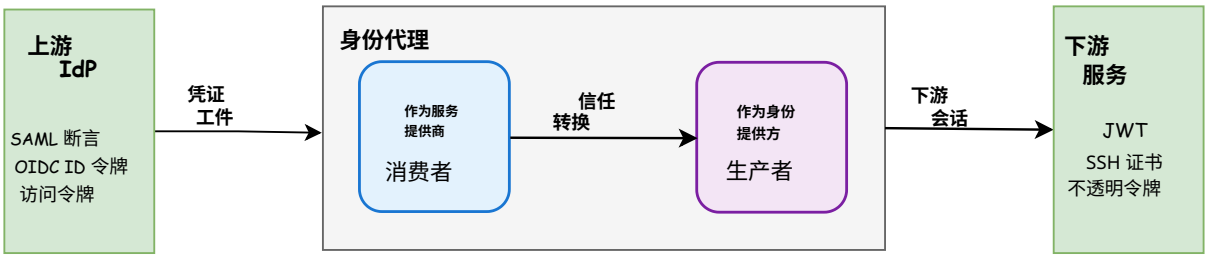


图1：身份代理同时充当面向上游身份提供商的服务提供商（SP/消费者），以及面向下游服务的身份提供商（IdP/生产者）。下游会话的安全性完全取决于信任转换步骤的正确性。

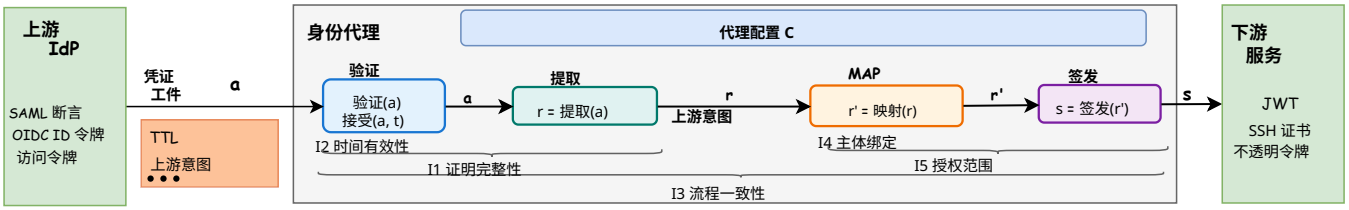


图2：四阶段信任转换流程，以及每个不变量（I1–I5）的适用位置。

来源 1：协议规范。我们从八份规范中汇编了 40 条与安全相关的规范性要求（must 和 shall 条款）：SAML 2.0 核心规范 [27] 和配置文件 [28], OpenID Connect 核心规范 1.0 [32], RFC 7519 (JWT) [21], RFC 7515 (JWS) [20], RFC 8693 (OAuth 2.0 令牌交换) [22], RFC 6749 (OAuth 2.0 授权框架) [14],以及 NIST SP 800-63C（数字身份指南：联合与断言） [12]。这 40 条款规定了身份验证证明处理、会话签发和授权委派，对应于 F 操作的各个阶段。

来源2：以往的通用漏洞披露。我们通过 NVD、GitHub 安全公告和平台变更日志，为 10 个身份代理平台（Keycloak、Dex、Vault、Teleport、Ory Hydra/Fosite/Kratos、Conjur、Shibboleth 和 OpenStack Keystone）收集了 60 个代理逻辑通用漏洞披露。完整列表见工件 [1]。我们根据威胁模型（§3）采用了严格的筛选标准：仅保留身份验证和授权逻辑漏洞；排除跨站脚本攻击、跨站请求伪造、拒绝服务攻击、内存安全和依赖项漏洞。对于每个通用漏洞披露，违反的要求由修复提交本身确定：代码变更揭示了缺失或不正确的检查，而该检查可直接映射到上述 40 条规范性条款之一。例如，CVE-2018-14637（Keycloak）的修复增加了 NotOnOrAfter 验证，可明确映射到 SAML 核心规范 §2.5.1.2。

根据缺失或不正确的安全检查对 60 个通用漏洞披露进行分组后，我们得到五个互不重叠的集群，共同涵盖全部 60 个通用漏洞披露。我们将每个集群所违反的属性称为 不变量，并在 §4.3 中对其进行形式化定义。只有满足以下三个条件的集群，我们才将其保留为不变量：1）至少有一条协议规范中的可引用条款为其提供依据；2）至少有两个独立开发的平台中的通用漏洞披露违反了该属性；3）无需源代码访问，即可通过黑盒 API 交互对其进行测试。

4.3 不变量定义

现在，我们将分别定义每个不变量，包括其形式化规范、所防护的流水线阶段（图2）、相关协议条款以及一个代表性通用漏洞披露。

我们使用两种字体。以小型大写字母书写的代理操作，表示代理实际对凭证执行的计算。我们定义的其他函数以斜体表示，用于简洁地规定不变量。

I1：证明完整性 (验证 → 提取)。如果提取阶段在处理凭据工件 a时成功完成，且该工件来自成功的验证阶段，那么由 a 派生的表示 r 在内容上必须与 a 匹配。凭据工件 a 和表示 r 都是记录，其中包含以下语义上下文：主体身份、认证上下文（例如身份验证方法和保证级别）以及授权声明（例如组成员关系）。我们假定存在一个 SC 函数，可输出 a 和 r的语义上下文。

$$\textsc{Verify}(a) \wedge r = \textsc{Extract}(a) \implies SC(a) = SC(r) \qquad (2)$$

该不变量基于 SAML Core §5.4.2（断言必须受到完整性保护）、OIDC Core §3.1.3.7（签名和签发者验证）以及 RFC 7515 §5.2（JWS签名验证）。I1 有两个子模式。*I1-a*（已签名 ≠ 已使用）：代理验证一个元素，却从另一个未经验证的元素中提取身份。这是经典的 XML 签名包装攻击 [34] 及其 JWT 对应攻击。*I1-b*（错误验证器）：代理应用了错误的验证策略，例如信任由攻击者控制的 JWKS端点，或接受非 CA证书作为信任锚。

代表性通用漏洞披露：CVE-2024-8698（Keycloak）：SAML 签名验证逻辑验证了一个 XML元素，但代理却从另一个未签名的元素中读取身份声明，位于

同一响应中。攻击者可以构造一个响应，其中包含一个合法签名的元素，以及一个携带攻击者自选身份声明的未签名元素；代理验证前者，却授予反映后者内容的会话，从而能够冒充任何本地账户。

I2：时间有效性（验证阶段）。在验证阶段，如果代理在 $t+\Delta$ 时接受了 a，其中 a 于 t 时签发，则 Δ 位于 $TTL(a)$ 范围内。设 k 为同一工件已成功接受的次数，包括当前这次接受。如果 a 携带一次性使用标记，则代理仅可在 $k=1$ 时接受它。

$$\begin{aligned}\textsc{Accept}(a, t+\Delta, k) \implies 0 \le \Delta \le TTL(a) \\ \wedge \left(OneTimeUse(a) \implies k = 1\right)\end{aligned} \tag{3}$$

这一不变量以 SAML 核心规范 §2.5.1.2（NotOnOrAfter）、§2.5.1.5（一次性使用）、OIDC Core §3.1.3.7（exp、随机数）以及 RFC 7519 §4.1.4 （ exp ）中的规范为依据。

代表性 *CVE*：CVE-2018-14637（Keycloak）：SAML 代理端点忽略了断言过期条件，使攻击者能够重放已过期的断言，从而创建新的下游会话。

I3：流程一致性（贯穿整个流水线）。如果代理根据表示 r' 签发下游会话 s，那么 s 必须由流水线的完整执行得出。必须存在一个代理已成功验证的上游制品 a、一个代理从 a 中提取的表示 r，以及一个代理从 r 映射得到的下游表示 r'。除非同一源制品的每个前序阶段都已成功完成，否则代理不得生成会话。

$$\begin{aligned}s = \textsc{Issue}(r') \implies \\ \textsc{Verify}(a) \wedge r = \textsc{Extract}(a) \wedge r' = \textsc{Map}(r)\end{aligned} \tag{4}$$

这排除了跳过某个阶段的执行路径、接受拼接自另一个会话的状态，或通过替代入口点绕过已配置的身份验证因素。

这一不变量并非现有规范中明确完整阐述的属性。相反，它是针对代理的概括，其依据是以下规范条款：OIDC Core §3.1.2.1（state）、§3.3.2.6（一次性授权码）、SAML Profiles §4.1.4.2 [28]（具体而言，即 InResponseTo 条款），以及 RFC 6749 §4.1.3（关于绑定客户端_id/redirect_uri 的讨论）。

典型漏洞披露：CVE-2021-3827（Keycloak）：SAML ECP 绑定路径绕过了其他认证流程（包括多因素认证），使攻击者无需完成所需的身份验证步骤即可获取会话。

I4：主体绑定（映射阶段）。如果两个上游表示的主体身份组件 r_1, r_2 在代理的配置下彼此不同（即配置未将它们声明为同一身份），代理的映射阶段必须将它们映射到不同的下游身份 $r\prime_1, r'_2$。我们假设存在一个函数 ID，用于从表示中提取主体身份。

$$\begin{aligned}ID(r_1) \neq ID(r_2) \wedge r'_1 = \textsc{Map}(r_1) \wedge r'_2 = \textsc{Map}(r_2) \\ \implies ID(r'_1) \neq ID(r'_2)\end{aligned} \tag{5}$$

这一不变量基于 OIDC Core §2（sub 必须唯一标识最终用户）、RFC 7519 §4.1.2（sub）以及 RFC 8693 §2.1（subject_令牌用于标识被代表行事的主体）。

典型漏洞披露：CVE-2020-16250（Vault）：AWS IAM 身份验证方法接受了被篡改的身份值，使一个 AWS 主体能够以另一个主体的身份通过身份验证。

I5：授权范围（映射 $\rightarrow$ 签发）。如果代理从凭据工件 a中提取表示 r，将 r映射为表示r'（其中包含主体 ID（$r\prime$）），并为 ID（$r\prime$）签发会话 s，那么 s所携带的权限将受到附加到 a的上游意图以及代理配置授予 ID（r'）的权限的约束。该不变量规定，授予的权限必须同时符合来自上游身份提供方上游意图 的原始意图和代理配置 C。

$$\begin{aligned}r = \textsc{Extract}(a) \wedge r' = \textsc{Map}(r) \wedge s = \textsc{Issue}(r') \\ \implies Perms(s) \subseteq UpstreamIntent \cap Policy(C, ID(r'))\end{aligned} \tag{6}$$

这里的上游意图 表示上游意图希望授予的一组权限，这些权限被编码到代理接收的凭据工件 a 中。此外，我们假设存在一个函数 $Policy$，它以主体 ID(r') 和代理配置 C 为输入，并生成 C 中通用权限的具体实例。最后，我们假设存在一个函数 $Perms$，用于从已建立的会话中提取一组权限。具体来说，考虑一个 JWT a，其上游意图希望授予对两个应用程序接口及其中特定操作的访问权限：{(finance-api，read:reports)，(comp-api，exec:task)}。在我们的示例中，代理的配置使得策略(C $ID(r')$) 返回 {(finance-api，read:reports)，(hr-api，write:reports)}。对于此示例，不变量 I5 允许授予 (finance-api，read:reports) 权限。相比之下，根据该不变量，授予 (comp-api，exec:task) 不符合策略，因此不被允许；同样，由于访问 HR 功能 (hr-api，write:reports) 不在意图范围内，也不被允许。

该不变量基于 SAML Core §2.5.1.4（受众限制）、OIDC Core §3.1.3.7（aud 必须包含客户端_id）、RFC 7519 §4.1.3（aud）以及 RFC 8693 §2.1（受众和范围参数）。

代表性通用漏洞披露：CVE-2024-5798（Vault）：JWT 身份验证方法未根据角色绑定受众验证 aud 声明，因此允许为一个服务签发的令牌在另一个服务进行身份验证。

4.4 覆盖范围与验证

我们的数据集中，每个通用漏洞披露都至少违反了五个不变量之一。表 1 展示了 60 个受调查通用漏洞披露中的不变量分布。不变量 I1 是最大的一组（涉及 7 个平台上的 19 个通用漏洞披露），这与 SAML [34] 和 JWT 处理领域长期存在签名相关漏洞的情况一致。不变量 I5 是第二大组（18 个通用漏洞披露），反映出在令牌交换与声明映射边界上执行范围约束普遍存在困难。不变量 I2 和 I3 涉及的通用漏洞披露较少，但各自覆盖 4 至 5 个平台。一个潜在的顾虑是循环论证：如果这些不变量是从通用漏洞披露中推导出来的，那么利用它们查找漏洞或许只是在找出那些相同的通用漏洞披露。

表 1：60 个通用漏洞披露在五个不变量中的分布。"B #"统计独立开发的代理代码库数量。"S #"统计每个不变量在全部八项规范中所对应的规范性要求数量。

不变量	CVE	B #	S #	示例通用漏洞披露
I1 证明完整性	19	7	10	CVE-2024-8698
I2 时间有效性	7	5	8	CVE-2018-14637
I3 流程一致性	10	4	9	CVE-2021-3827
I4 主体绑定	6	2	6	CVE-2020-16250
I5 授权范围	18	4	7	CVE-2024-5798
总计	**60**	**10**	**40**	

然而，我们的评估（§6.3）显示，产生的发现项中有 45% 符合此前没有通用漏洞披露记录的模式，这些发现项集中在通用漏洞披露覆盖最少的两个不变量（I3 和 I4）上。我们指出，这些不变量仅构成代理正确性规范的一部分；我们并未声称它们是完备的。

5 系统设计

本节介绍 PORTA，这是一个能够自动测试身份代理是否违反第 §4 节所定义不变量的系统。其核心挑战在于，每个代理都提供不同的应用程序接口，但待测试的安全属性（I1–I5）与平台无关。PORTA 通过四阶段流水线（图 3）解决这一问题：(1) 平台适配 将每个代理的应用程序接口抽象为通用配置，从而只需编写一次测试逻辑；(2) 序列生成 利用大语言模型根据不变量生成潜在的攻击序列和预言机断言；(3) 执行 针对在线代理实例运行每个序列；(4) 判定 根据预言机评估捕获的行为，以判断是否发生违规。大语言模型仅参与阶段 1 和阶段 2；阶段 3 和阶段 4 完全具有确定性。

5.1 平台适配

待测试的经纪服务可能使用不同的管理接口、端点路径和认证流程。直接针对每个应用程序接口编写测试序列，就需要为每个平台分别编写逻辑；而添加新平台则意味着必须从头编写新测试。为应对这一挑战，PORTA 引入了抽象操作。抽象操作是一种必须针对每个平台进行实例化的通用操作。尽管实例化操作的名称和调用机制可能因平台而异，但抽象操作在每种情况下都会产生相同的语义效果，例如成功创建用户账户。PORTA 定义了 33 个抽象操作（例如 saml_login、token_exchange、create_user）。[1] 其中 9 个操作来自 RFC 6749 等协议规范，其余 24 个是我们研究的七个主流代理平台共有的管理操作。包含规范参考文献的完整列表见工件 [1]。

给定一组抽象操作，我们的方法会自动为每个平台生成一个平台配置文件。该配置文件将每个抽象操作与平台的 HTTP 方法、URL 路径、请求参数和响应字段进行关联（图 3 中的步骤 ❶）。

[1]注意：PORTA 定义了 33 个抽象操作，并且恰好发现了 33 个漏洞，这纯属巧合。

具体而言，对于每个平台，大语言模型会接收该平台的 API文档，并以固定的结构化格式生成配置文件（提示词框架见工件 [1] ）。由于大语言模型可能会生成错误的端点路径或参数，因此每个平台都会运行三次配置文件生成流程，并合并三次输出：每个端点的 HTTP 方法和路径通过精确字符串多数表决确定（至少三次运行中的两次必须一致），而请求体参数和响应字段则取并集。如果三次运行对某一路径的结果都不一致，则会暂不采用该操作，并在使用前通过在线代理进行验证。在我们测试的七个平台中，均未出现这种情况。

5.2 不变量引导的序列生成

在生成步骤中，大语言模型会接收三个输入（图 3 中的步骤 ❷）：平台配置文件，其中列出了可用的抽象操作以及如何在该平台上调用这些操作；目标不变量定义（§4.3），其中说明了攻击必须违反的属性；以及我们为每个平台手动编写的一个示例序列，用于展示生成器预期输出的结构格式，包括设置步骤和执行步骤。该示例仅向大语言模型提供一个通用的序列模板。

首先，对于每个不变量 I1–I5 和每个平台，我们都会要求大语言模型提出一对抽象操作序列：一个候选攻击序列，用于针对给定平台探测该不变量（例如，再次重放同一个 SAML 断言以测试 I2）；以及一个合规序列，用于建立预期的基线（例如，执行正确的 SAML登录并生成有效的下游会话）。接下来，在获得抽象操作序列后，我们会要求大语言模型使用配置文件（§5.1）将其解析为特定于平台的具体HTTP请求。这种抽象/具体两层架构遵循基于模型的测试范式，并已用于安全研究，以测试同一协议的多个实现 [5, 17, 38]。

为避免重复执行，生成的序列会在调度前进行去重，例如，针对同一不变量且攻击步骤在结构上相同的序列。每次活动针对每个不变量、每种协议绑定生成 10 个序列，因此在去重前，每个平台约有 100 个候选序列。

除序列之外，大语言模型还会生成一个 预言机断言。预言机断言是将通用不变量实例化为一个布尔表达式，该表达式基于从已执行序列中捕获的值。例如，对于一个通过同一代理登录两个不同上游主体的序列，I4 不变量 $D(p_1, p_2)$ ⇒映射(p_1) ≠ 映射(p_2) 的主体可实例化为断言 entity_id_first_user != entity_id_second_user。

5.3 执行引擎

在完成平台特定的序列和不变量解析（§5.2）后，我们针对在线代理执行大语言模型生成的序列，以生成证据。执行引擎在隔离的 Docker容器中，针对运行中的代理实例执行每个序列（图 3 中的步骤❸–❹）。每次运行前，容器都会被销毁，并根据固定版本镜像重新创建（表 2 列出了所有版本），从而确保环境干净，不会残留先前测试中的会话、令牌或配置。

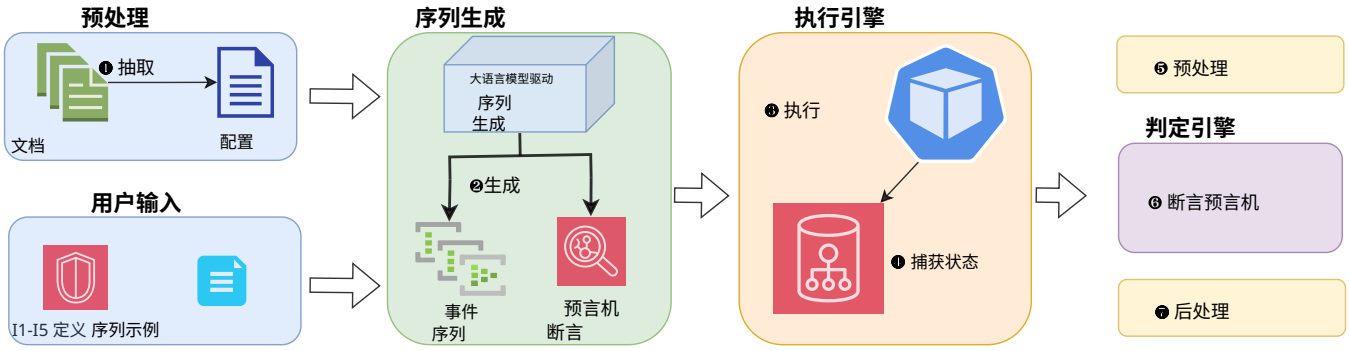


图 3：PORTA 流水线的系统概览。

执行期间，引擎会发出 HTTP 请求，并将从某个响应中捕获的值串接到后续请求中；例如，将授权调用返回的授权码传入后续的令牌交换。

在每个需要凭证的步骤中，引擎都会按需生成凭证。对于 SAML，引擎会生成签名断言。对于 JWT，引擎会生成令牌，其在 RFC7519[21] 中定义的标准声明（例如 iss、aud、exp 和 sub）取值由序列指定。对于合规序列，每个凭证都由可信密钥签名；对于攻击序列，则由攻击者控制的密钥签名。平台适配器负责处理特定于协议的流程，例如 OIDC 授权码交换和多步骤身份提供商意图 API，而不会向序列生成器暴露这些细节。

传输错误和模板替换失败会中止序列，并将其排除在所有发现项计数之外。但是，合规序列中某个步骤返回非 200 状态码，并不会因此终止序列。共同生成的预言机断言会观察失败步骤，并发出相应的发现项供人工分类。

5.4 判定引擎

PORTA 的判定引擎会评估预言机断言。每条预言机断言都是与具体序列共同生成的布尔表达式，用于编码不变量成立时必须满足的条件。

由于断言是目标不变量（§5.2）在每个序列中的具体实例化，因此，错误断言对应于代理违反不变量。断言预言机涵盖全部五个不变量，并解释了 PORTA 发现的大多数漏洞。

预处理。 PORTA 在启动判定引擎之前会执行预处理步骤。每当 HTTP 响应到达时，PORTA 都会应用轻量级语法检查，以标记单个响应中可见的违反情况，而无需等待序列完成（步骤❺）。这些检查包括：格式正确的输入产生的服务器错误（HTTP 5xx）；签发的 JWT 令牌的 iss 声明与代理配置的可信签发者不匹配；以及缺少已授权随机数的 OIDC id_令牌。此预处理仅报告可直接依据 RFC 级要求确定的违反情况，对总体发现结果的贡献比例较小。

后处理。 除了判定引擎之外，PORTA 还有两个用于发现结果后处理的来源——回退机制和变异扫描。

对于不变量I4和I5，PORTA 增加了一种不依赖大语言模型生成的断言的回退机制（步骤❼）。它会查询代理的用户信息或内省端点，并将上游制品中的主体身份和权限与代理实际授予的会话所携带的主体身份和权限进行比较。身份不匹配会标记 I4，权限不匹配会标记 I5。

PORTA 还会将一个经过人工整理的 102 项凭证变异库（在工件 [1], 中提供，并按目标不变量分组）应用于具体的合规序列。我们创建此变异库时参考了 60 个通用漏洞披露（§4.2）以及有文档记录的 SSO 攻击模式，例如 XML 签名包装攻击 [34] 和带有 alg=none 标头的 JWT 令牌，该标头表示未签名令牌。每项变异都会单独应用，以便对变异凭证的接受情况能够隔离出代理遗漏的具体验证检查。示例包括移除签名、更改签发者、将过期时间设置为过去、替换受众，以及去除 SAML <Conditions> 元素。该变异库补充了使用大语言模型生成的序列进行的测试，其中包含旨在触发五个不变量各自违反情况的变异。

变异扫描会重用第 2 阶段生成的合规序列，并一次更改一个凭证。每次变异都会针对单个元素——用户、提供方、客户端、组织、策略和身份验证状态——然后执行登录、SAML 回调或令牌交换。这样，PORTA 就可以测试跨组织的配置变更、多因素身份验证路径、身份冲突和令牌交换。

6 评估

6.1 实验设置

我们测试了七种开源身份代理平台，涵盖两种协议绑定（OIDC/JWT 和 SAML），共形成 13 种不同的平台/协议配置（Vault 的 SAML 身份验证仅在企业版中提供；我们测试的是仅支持 OIDC 的开源社区版）。表 2 列出了这些平台、版本及 GitHub 星标数，并以此作为部署普及度的参考。每个平台都在独立的 Docker 容器中运行；每次协议测试之间，都会根据固定版本镜像重新创建容器。

我们进行了三次独立实验，每次实验都使用单独生成的一组平台配置档案（§5.1），因此三次运行的方差

表 2：PORTA 测试的目标平台。

平台	版本	协议	GitHub 星标
Keycloak [31]	26.5.6	OIDC、SAML	33.8k
Casdoor [4]	2.362.0	OIDC、SAML	13.3k
Dex [6]	2.45.1	OIDC、SAML	10.7k
Authentik [23]	2026.2.1	OIDC、SAML	20.9k
Vault [15]	1.21.4	OIDC	35.4k
Zitadel [40]	4.13.0	OIDC、SAML	13.5k
Logto [33]	1.37.1	OIDC、SAML	11.9k

反映了包括配置文件生成中的大语言模型差异在内的完整流水线。每次实验针对每个不变量和协议绑定测试 10 个序列。

所有大语言模型阶段（配置文件生成、序列生成和 OIDC 断言生成）均使用单一配置：gpt-5_2-2025-12-1、默认温度和 top-p = 1.0。从序列生成到判定分配，三次运行均实现了全自动化。每次运行内部都会自动进行发现结果去重。与触发漏洞的序列相关的源代码识别、归纳为共同的根本原因，以及报告准备工作，均在多次运行之间手动完成。

6.2 已知漏洞的召回率

为衡量召回率，我们在 60 个 CVE 语料库的一个子集上评估 PORTA。在该语料库中，有 34 个 CVE 针对我们测试集中的应用程序（Keycloak、Dex 和 Vault）。其中，18 个 CVE 涉及 OIDC 或 SAML，即 PORTA 支持的协议。不过，其中有 7 个需要协议绑定、并发、外部参与者或执行器目前不支持的操作。因此，最终有 11 个 CVE 用于召回实验。工件 [1]详细介绍了筛选过程。对于每个选定的 CVE，召回实验均使用受影响系统固定的已知存在漏洞的补丁前版本，相关信息记录在工件 [1]中。

在三次独立实验中，在执行前移除结构上相同的序列（§5.2）后，PORTA 每次运行平均生成 453.0 ± 23.6 个序列。

只有当生成的序列触发 CVE 报告中描述的行为时，我们才认为发现了该 CVE。表 3 报告显示，在三次实验中分别发现了 11 个 CVE 中的 10 个、7 个和 8 个。合并三次实验的结果后，在该子集中检测到 11 个 CVE 中的 10 个（90.9%）；当考虑全部 18 个 OIDC 和 SAML CVE 时，端到端检测率为 18 个中的 10 个（ 55.6% ）。CVE-2024-5798 仅在三轮实验中的一轮被检测到。唯一从未被检测到的 CVE 是 CVE-2018-14637。在此情况下，生成的测试旨在使用已过期的 SAML 断言。然而，断言构建器将 NotOnOrAfter 设置为断言创建后 30 分钟，因此断言在发送时仍然有效，未能触发该漏洞。召回实验的 API 成本为 53.53 美元。

6.3 漏洞发现结果

在选定平台（§6.1）上进行的三轮评估中，PORTA 针对每个不变量和协议绑定生成 10 条序列；经过过滤结构相同的序列（§5.2）后，每轮平均生成 500.7 ± 12.5 条序列。

三次运行的 API 成本总计为 69.5 美元。

表 3：在三次独立实验中检测已知 CVE。Y 和 N 分别表示 PORTA 是否检测到该 CVE。最后一列给出了检测到该 CVE 的实验次数。

CVE	实验 1	实验 2	实验 3	已发现
CVE-2018-14637	N	N	N	0/3
CVE-2022-1245	Y	N	Y	2/3
CVE-2023-0264	Y	Y	Y	3/3
CVE-2023-2422	Y	Y	Y	3/3
CVE-2023-2585	Y	Y	Y	3/3
CVE-2023-6291	Y	Y	N	2/3
CVE-2023-6544	Y	Y	Y	3/3
CVE-2024-8698	Y	N	Y	2/3
CVE-2025-1391	Y	Y	Y	3/3
CVE-2024-5798	Y	N	N	1/3
CVE-2025-3879	Y	Y	Y	3/3
已检测	**10/11**	**7/11**	**8/11**	

表4：PORTA 在七个代理平台上的三次独立实验评估。序列：**实验中生成的序列数量。**执行率：**已完成序列的百分比。**发现结果：**去重前的候选漏洞数量。**去重后：**去重后的候选漏洞数量。**真/假正例（*TP/FP*）：**经人工分类确认/排除的漏洞数量。**精确率：$TP/(TP+FP)$。

	实验1	实验2	实验3	$\mu \pm \sigma$
序列	492	515	495	500.7 ± 12.5
执行率	72.4%	69.5%	76.0%	72.6 ± 3.3%
发现结果	206	439	281	308.7 ± 118.9
去重后	54	96	102	84.0 ± 26.2
真正例	44	89	93	75.3 ± 27.2
假正例	10	7	9	8.7 ± 1.5
精确率	81.5%	92.7%	91.2%	**88.5 ± 6.1%**

在三次独立运行中（表4），PORTA 检测出 926 个候选漏洞。随后，PORTA 使用自动去重的确定性处理阶段，合并具有以下值的发现结果：变异扫描发现结果的变异规则 + 不变量；断言违反发现结果的不变量 + 判定路径 + 断言文本的哈希。去重后，我们得到 252 个候选漏洞（每次运行 84.0 ± 26.2 个）。

我们根据流水线顺序，将每个候选发现项归因于最先触发的主要检测层。带有断言预言机的判定引擎（步骤 ❻）是主要机制，每次运行产生 70.0 ± 21.2 个发现结果，约占所有发现结果的 83 %。后处理（步骤 ❼）包含两个组件：回退机制贡献 7.0 ± 4.6 个发现结果，变异扫描每次运行产生 4.0 ± 1.7 个发现结果。预处理（步骤 ❺）贡献其余部分，每次运行 3.0 ± 1.0 个。

我们对 252 个候选漏洞进行了人工分析，并确认其中有 226 个；三轮测试中共出现 88.5 ± 6.1 % 个误报，对应的精确率为 26。我们进一步进行了根因分析，发现已确认的 226 个

个漏洞由 §7 中讨论的代码中的 33 个不同安全缺陷导致。我们还发现，26 个假正例有四种不同的根本原因；假正例的详细分类见 §6.4。

表5总结了各平台的漏洞分类情况。我们将所有已识别的缺陷分为两类：规范违规和规范未充分规定的漏洞。这种分类反映的是规范的严谨程度，而非可利用性：一些规范未充分规定的缺陷，例如通过未经验证的电子邮件造成的身份冲突，在实际中比某些规范违规更容易被利用。

我们还将每个根本原因漏洞与60 个通用漏洞披露进行匹配：如果某个通用漏洞披露的修复提交加入了与我们的发现中所指出的缺失验证检查相同的检查，则该发现项被视为“匹配”。

规范违规和规范未充分规定的漏洞。如果观察到的行为与规范中的规范性“必须”要求相矛盾，则该缺陷属于规范违规。例如，对于时间有效性（I2），Authentik 的 JWT 客户端断言端点会接受一个 exp 声明已在十分钟前过期的令牌（规范违规：该声明存在，但被忽略）。33 个根本原因漏洞中，有 15 个属于规范违规。示例包括：由于缺少防重放缓存而导致的 SAML 断言重放（I2，Keycloak、Casdoor 和 Dex）；未验证 SAML AudienceRestriction（I5，Zitadel、Casdoor 和 Authentik）；禁用了 OIDC 签发者验证（I1，Authentik）；令牌端点未强制执行 JWT 客户端断言过期时间（I2，Authentik）；以及通过反向检查逻辑绕过 OIDC nonce（I2，Logto）。

三次运行结果的并集涵盖了全部 15 项规范违规，其中 15 项中的 12 项至少被两次运行发现，5 项则被三次运行全部发现。仅在一次运行中发现的三项缺陷分别是：仅在第一次运行中发现的两个 Authentik OIDC 时效有效性缺陷（I2）（当时大语言模型生成了所需的过期绕过序列），以及仅在第二次运行中发现的一个 Casdoor SAML 证明完整性缺陷（I1）。

规定不足的 缺陷在实际应用中并不安全，但规范中的规范性表述为可选或含糊不清。

例如，Zitadel 的 JWT 处理程序仅在 exp 缺失时跳过过期时间检查（规定不足：RFC 7519 将 exp 标记为可选）。

已识别的 33 个根因漏洞中，有 18 个属于规定不足。值得注意的例子包括：非原子配置强制执行造成竞态窗口（I3，Dex 和 Keycloak）；缺少 <Conditions> 元素时会禁用受众验证，而规范允许但不建议这种行为（I5，Keycloak 和 Dex）；以及通过不区分大小写的主体匹配导致的身份冲突（I4，Vault）。

在18个规范不足的缺陷中，全部18个缺陷至少被两次运行发现，其中9个缺陷被三次运行全部发现，其比例高于规范违规，这反映出规范不足的缺陷由更广泛的攻击模式触发，而大语言模型能够在独立运行中可靠地生成这些攻击模式。

匹配与未匹配模式。在进行 CVE 匹配时，我们将识别出的每个代码缺陷与 §4.2 中的 60 个 CVE 语料库进行比较。“匹配”的示例是：CVE-2018-14637 的补丁通过添加 NotOnOrAfter 强制执行来修复 Keycloak，而我们的发现项则是 Casdoor 同样缺少 NotOnOrAfter 强制执行。若数据集中没有任何 CVE 修复涉及某项发现，则该发现项为“未匹配”。

表5：各平台的发现结果。规范。：规范违规（规范性必须要求）。规范不足。：规范未涵盖的安全缺陷。匹配：根因模式与 §4.2 数据集中某个平台上的现有 CVE 相匹配。未匹配：模式与数据集中的任何 CVE 都不匹配。

平台	规范	规范不足	总计	匹配	未匹配
Keycloak	1	4	5	4	1
Casdoor	6	3	9	5	4
Dex	1	4	5	2	3
Authentik	4	1	5	2	3
Vault	0	1	1	0	1
Zitadel	2	1	3	2	1
Logto	1	4	5	3	2
总计	**15**	**18**	**33**	**18**	**15**

同一问题。按照这一标准，33 个发现项中有 15 个不匹配，另外 18 个匹配，但这些问题是在此前未曾报告过的平台上确认的（即漏洞在平台之间“转移”）。不匹配项包括 15 项规范违规中的 5 项，以及 18 项规范说明不足缺陷中的 10 项。不匹配的发现项集中在违反不变量 I3（七项中有六项）和 I4（四项中有两项）的情况；这两个不变量的 CVE 记录最为稀少（表 1）。

负责任披露。我们于 2026 年 3 月 4 日开始向所有受影响平台的维护者进行负责任披露。本文撰写时，已披露的 33 个缺陷中有 29 个得到确认，14 个已修复，19 个已分配 CVE 标识符。至少 10 个已分配标识符的漏洞，其 CVSS 评分达到或超过 9.0，属于严重级别。

6.4 误报分析

每次运行的误报数量（表4）保持稳定（8.7 ± 1.5, σ μ = 17 26 /%）。我们将误报归纳为四种不同的误分类机制。具体而言，我们根据触发误报的不变量违反及其误触发原因，对假正例进行分组。

后处理（10 个假正例，I4，步骤❼）。I4 回退机制（步骤❼，§5.4）会将上游制品中的主体与代理授予的主体进行比较。在 Casdoor 的 10 个案例中，代理以不同的显示格式返回了同一用户（例如，发送的用户名为 i2_logout_user，而授予的显示名称为 I2 Logout User），导致字符串比较被触发，尽管实际上并未发生身份混淆。

失败的合规序列（8 个假正例，多个不变量，步骤❻）。序列生成步骤（§5.2）会生成一对攻击序列和合规序列，并为二者配备用于检查不变量的预言机断言。当合规序列失败，即返回 4xx 状态码（通常是因为大语言模型使用了错误的端点或遗漏了参数）时，可能会错误地触发预言机。例如，当令牌请求失败且未捕获到请求的作用域时，检查授予的作用域是否超出请求的作用域这一检查可能会被错误触发。

预处理（4 个误报，I1，步骤❺）。4 个误报是格式错误输入（标头中的 NUL 字节、空的 subject 声明）导致的 HTTP 500 服务器错误；逐响应检查（步骤❺）保守地将其标记为证明完整性违规

（I1）。每种情况下，代理都在输入解析期间崩溃，未能执行其身份验证逻辑，因此没有无效凭据被接受。

测试脚手架（4 个误报，步骤 ❻）。其余四个误报源于测试基础设施故障：Authentik 返回了空身份字符串，而 Vault 因缺少别名名称而拒绝了 JWT。预言机断言（步骤 ❻）将这些基础设施故障视为不变式违规，因为捕获的响应值偏离了预期。

6.5 消融研究

为衡量 PORTA 背后的设计选择对结果的影响，我们将完整系统与三种关闭部分流水线功能的配置进行比较。这些属于配置级消融，而非严格的组件隔离实验：每种配置都会改变流水线的一部分，因此产生的差异反映的是改变该部分对整个系统的影响，而不是单独该部分在隔离状态下的作用。每种配置使用与漏洞发现评估（§6.3）相同的目标平台、相同的代理配置文件集，以及三次独立运行。

表6报告了系统三个消融变体的汇总结果。移除不变量定义和攻击示例后，我们得到 PORTA 的 Blind 版本。它只能使用平台配置文件和通用的“发现安全漏洞”提示词；确定性裁决流水线保持不变。在生成提示词中保留不变量定义、但移除手动编写的示例后，我们得到 No-example 版本。LLM-judge 版本保留完整的不变量引导生成流程，但将确定性裁决流水线替换为一次大语言模型调用，由其判断每个捕获的序列是否表现出违规。LLM-judge 不仅输出序列是否存在漏洞，还会为其判定给出置信度。这三个消融实验分别隔离了三个独立维度：不变量集合决定系统测试什么，示例决定测试的精确程度，而确定性预言机决定判定是否值得信任。

盲测。 Blind 版本在全部三次运行中均未产生任何真正例，其 19 个发现项全部为误报。其中 74% 是被错误标记为证明完整性违规的 HTTP 500 服务器错误：由于没有目标不变量，大语言模型生成的是会导致请求解析器崩溃的格式错误输入，而不是能够检验身份验证逻辑的输入。这并不是序列数量问题；Blind 每次运行实际生成的序列多于完整系统（636 个对 501 个），但成功执行的序列比例更低（56.5% 对 72.6%），因为无引导序列针对的是错误的端点，并使用了不正确的请求格式。

无示例。 无示例版本在相近的序列预算下识别出 33 个发现结果中的 26 个（§6.3 中发现结果的一个子集），但其执行率从 72.6% 降至 63.3%，每次运行精确率下降 13 个百分点（从 88.5% 降至 75.6%），每次运行的真正例下降 43%（从 75.3 降至 43.0）。无示例版本始终无法发现的 7 个发现结果（见 §6.3）都需要多步骤设置或不明显的参数，包括跨组织令牌交换、JWT 客户端断言的时间绕过，以及身份冲突。仅有不变量定义过于抽象，无法引导大语言模型找到这些模式。

大语言模型评审。 用大语言模型评审器替代确定性裁决流水线，同时保持生成流水线不变。大语言模型评审在数量相当的序列上以相近的执行率运行，因此结果的任何差异都可归因于裁决层，而非输入。在这些匹配条件下，大语言模型评审每次运行产生的发现结果数量几乎是完整系统的三倍，但其中只有 20.3 ± 2.1 个是真正例，精确率为 8.4%，而完整系统为 88.5%。大语言模型评审还遗漏了 33 个纳入的发现结果中的 21 个，包括 Casdoor 的所有 SAML 受众验证和断言重放发现结果，以及 Authentik 的大多数 I1 和 I3 发现结果。大语言模型评审的全部 12 个发现结果都是 PORTA 在 §6.3 中发现结果的子集。

对误报进行人工甄别后发现，大语言模型的发现结果中有 63% 是虚构的，也就是说，大语言模型报告了具体的攻击描述（例如“意图拼接绕过”“缺少应用标签强制执行”），但根据平台源代码进行人工验证后发现，这些描述并不对应任何实际的安全相关行为。确定性的断言预言机构造上无法产生虚构的发现结果：断言空间由执行前编写的类型化状态布尔表达式构成，并且每个表达式都会针对具体捕获值进行机械求值。

表6中的结果汇总了在所有经过测试的大语言模型置信度级别下的发现结果。对大语言模型采用高置信度设置可获得最佳结果，因为精确率达到 38.2%，但与完整系统相比仍存在 $2.3\times$ 差距。在工件 [1], 中，我们详细介绍了不同置信度配置下的结果，以及向评判器提示中加入 I1–I5 不变量定义是否能够改善性能。与表6中报告的配置相比，所有这些配置的表现都更差。

7 安全发现与洞见

7.1 漏洞分类

本节研究 §6.3 中报告的实际缺陷究竟是孤立的实现错误，还是遵循相似的模式。具体而言，我们从两个维度分析这 33 个漏洞：被破坏的安全属性（故障族），以及导致违规的编码假设（假设不匹配模式）。这两个维度彼此正交：族描述出了什么 问题；模式则描述代码为何 会如此编写。

跨平台故障族。表7按被破坏的安全属性对 33 个发现项中的 30 个进行归类，并展示了最常见的七个族；每个族都影响了两个至六个独立开发的平台。其余三个发现项（Casdoor 的跨组织令牌交换、Casdoor 对已撤销令牌的接受，以及 Logto 的 Nonce 谓词反转）均为单例，因此未归入任何族。

族名称描述了被破坏的安全属性。违反不变量 I1、I2 和 I5 的族主要导致规范违规（S），因为 OIDC、SAML 和 JWT 规范中明确规定了这些不变量的必须条款。七个 I3 发现项中的六个以及全部四个 I4 发现项都属于规范不足的缺陷（U），规范并未明确涵盖这些情况。

与这些族相关的典型代码构造（表7）并非代码复用的实例。七个平台采用四种不同的编程语言（Java、Go、Python、TypeScript）编写，

表6：三次独立运行的消融结果。

配置	生成	判定	每次运行的序列数	执行率	每次运行的发现数	每次运行的真阳性数	精确率	发现数量
盲测	通用提示	确定性	636.0 ± 1.0	56.5 ± 3.1%	6.3 ± 3.5	0.0	0.0%	0 / 33
无示例	仅不变项	确定性	492.0 ± 7.9	63.3 ± 1.7%	57.0 ± 3.6	43.0 ± 6.0	75.6 ± 11.5%	26 / 33
大语言模型评审	不变量 + 示例	LLM	475.3 ± 7.0	71.5 ± 2.2%	245.3 ± 22.7	20.3 ± 2.1	8.4 ± 1.6%	12 / 33
Full	不变量 + 示例	确定性	500.7 ± 12.5	72.6 ± 3.3%	84.0 ± 26.2	75.3 ± 27.2	88.5 ± 6.1%	33 / 33

表7：跨平台漏洞族。不变量。**：被违反的不变量。**平台**：该族中独立开发的平台数量。**规范违规/规范不足的缺陷**：该族中的规范违规/规范不足的缺陷。**代表性条款**：对应的规范性条款。**典型代码形式**：导致该缺陷的重复出现的代码构造。

族	Inv.	平台	S / U	代表性条款	典型代码形式
未验证受众	I5	6	4 / 4	RFC 7519 §4.1.3；SAML §2.5.1.4	验证_aud：False 或不进行受众检查
时间有效性	I2	4	3 / 2	SAML §2.5.1.2；RFC 7519 §4.1.4	如果字段 != nil { validate() }
流程 / 非原子配置	I3	4	0 / 5	—	流程开始时已加载配置，但未重新检查
断言重放	I2	3	3 / 0	SAML §2.5.1.5	无持久化断言 ID 缓存
未验证签发者	I1	3	2 / 1	OIDC §3.1.3.7.4；SAML §2.2.3	验证_iss：假或来自响应的证书
通过联合身份认证绕过多因素认证	I3	2	1 / 1	NIST 800-63C §5.1 [12]	联合身份认证活动代码路径中缺少多因素认证检查
身份冲突	I4	3	0 / 4	—	不区分大小写或基于电子邮件的关联

由不同的开发团队独立开发，并且每个族都包括构建于互不相关的第三方协议库之上的代理：Casdoor 的 SAML 协议栈构建于 gosaml2 之上，Dex 构建于 crewjam/saml 之上，Keycloak 构建于其内置的 Java SAML 核心之上，Authentik 构建于自定义的 Python SAML 解析器之上，Logto 构建于 samlify 之上，Vault 构建于 HashiCorp cap/jwt 插件之上，而 Zitadel 则构建于其自身的 SAML SP 实现之上。

假设不匹配模式。 我们观察到，每个新发现的漏洞都是由一种假设不匹配促成的：某种对于单角色 RP 正确、但在同一代码于代理中运行时却会变得不正确的编码模式。设想这样一种场景：单角色 RP 从一个预先配置且已由运营者明确信任的身份提供方接收凭据。在此场景中，若干编码实践是正确的：接受身份提供方选择省略某个可选字段的做法，在启动时缓存一次身份提供方证书，仅验证活动代码路径所需的内容，并将进行中的协议状态视为单个事务。代理代码若将这些实践应用于上游凭据，就会把攻击者提供的输入当作来自预先配置的受信任源，而其威胁模型禁止这种做法：上游身份提供方可能由攻击者控制，断言可能已被篡改，配置可能在流程进行期间发生变化，或者两个上游身份可能在代理的本地命名空间中发生冲突。

我们按照假设不匹配的形式对这 33 个漏洞进行分类。为此，两位作者分别阅读相关源文件，并为每个发现项标注一种主要的假设模式（每项发现的分类及源文件链接见工件 [1]）；其中 3 处分歧经讨论后解决（Cohen's $\kappa = 0.89$；这三处分歧均发生在"静默跳过"和"显式禁用"这两个最相近的类别之间；详情见工件 [1]）。分类模式集并未预先确定，而是在分析过程中逐渐收敛为表 8 所示的八个类别。按假设不匹配模式划分，发现项数量最多的两组——字段缺失时静默跳过（9 项）和显式禁用验证（8 项）——合计略多于所有发现项的一半。两者都直接源于将 RP 风格的"照单全收"语义带入给定内容由攻击者控制的场景。其次是过时的配置快照（5 项）和跨命名空间身份坍缩（4 项）。I3 类发现涉及代理在完整流程中的行为，而所引用的规范仅规定了该流程中的局部检查。I4 类发现涉及跨身份提供方的命名空间隔离，而没有任何单角色规范对此作出定义。因此，这两组大多导致规范不足的缺陷（§7.2）。

假设不匹配类别与安全类别正交。单个族可以混合多种模式（受众类别结合了显式禁用和静默跳过），而单个模式也可能破坏多个族（静默跳过会同时破坏受众、时间有效性和条件处理）。每个发现项的完整映射见工件 [1]。

7.2 规范不足的缺陷

对这 33 个缺陷进行的规范强度划分揭示了现有规范和实践对哪些不变量类别的规定最不充分。在全部 4 个 I4 缺陷和 7 个 I3 缺陷中的 6 个缺陷中，相关要求的规定都不充分，也就是说，被违反的要求要么是可选的、未记录的，要么存在争议。剩余的一个 I3 缺陷违反了一项规范性要求条款。相比之下，I2（时间有效性和重放）的 10 个缺陷中有 7 个对应规范违规，I5（受众和授权范围）的 9 个缺陷中有 5 个对应规范违规。这种梯度反映了规范覆盖度的差异：I1、I2 和 I5 对应明确的必须条款，因此违规与否一目了然。相反，I3 不变量概括了若干规范性条款，因此现有规范中并未明确写出这一不变量。因此，该不变量的大多数违规预计会被归类为规定不足。I4 对应跨身份提供方的命名空间隔离，而没有任何规范对此作出规定。这些是不变量类别中，在身份代理生态系统里结构性规定最不充分的类别。

表 8：假设不匹配模式：**单角色依赖方假设泄漏到双角色代理代码中的方式**。发现数量：**具有该模式的漏洞数量**。依赖方假设：**适用于单角色依赖方的防御性编码假设**。代理影响：**不匹配造成的后果**。

假设不匹配模式	发现数量	依赖方假设	代理影响
字段缺失时静默跳过	9	遵循 IdP 对可选字段的省略	攻击者移除字段以禁用检查
显式禁用验证	8	一个受信任 IdP 使该检查变得多余	已禁用的检查是唯一的命名空间分隔符
过时的配置快照	5	单步请求没有进行中的状态	多步骤流程跨越配置变更
跨命名空间身份坍缩	4	在同一身份提供方内，规范化只是为了方便	在各身份提供方之间合并了不同的主体
没有持久化反重放状态	3	浏览器会话本身不可重放	代理在不同会话中接受相同的证明
分支覆盖缺口	2	本地登录路径首先得到了强化	联合路径绕过本地策略
信任引导自反性	1	证书从带外元数据中缓存	证书从待验证的证明中读取
非状态谓词	1	在受信任 IdP 测试中，两个谓词一致	攻击者省略该字段；谓词退化

7.3 案例研究

我们通过五个案例研究展示新发现的漏洞。前四个属于规范违规。第五个是一个规定不足的缺陷，我们将其呈现为端到端攻击场景，以说明规定不足并不意味着难以利用。对于每个案例，我们都会概述代理的行为、PORTA 构造的攻击以及安全影响。

案例 A：JWT 受众绕过（I5，Zitadel；假设不匹配模式：显式禁用验证）。Zitadel 的通用 JWT IdP 会验证上游 JWT 的签名、签发者和过期时间，但从不验证受众声明是否与配置的下游服务匹配。由受信任 IdP 签名且 aud=service-B 的 JWT，可在配置为 service-A 的代理处成功通过身份验证。PORTA 构造一个 aud=service-A 的控制登录（即代理配置为保护的服务）和一个 aud=service-B 的攻击登录（另一个服务）；代理错误地接受了两者。

影响：跨服务横向移动，即，为某个下游服务签发的令牌可在同一代理后方的所有其他服务中通过身份验证。

这违反了 RFC 7519 §4.1.3 的要求：处理 JWT 的每个接收方都必须验证自身是否出现在 aud 声明中。

案例 B：签发者验证禁用（I1，Authentik；假设不匹配模式：显式禁用验证）。Authentik 的 OIDC 源处理程序在关闭签发者验证的情况下解码上游 JWT。来自攻击者控制的 IdP、且 iss 声明不匹配的令牌，会被当作来自已配置的信任锚而接受。PORTA 构造一个来自受信任 IdP 的合法登录，以及一个由攻击者 IdP 签名的攻击登录；两者具有相同的身份声明，代理却都接受了。

影响：完全绕过信任锚，即任何签发者都会被视为已配置的受信任 IdP。

这违反了 OIDC Core §3.1.3.7.4，该规范要求令牌中的 iss 声明必须与提供商配置的签发者标识符完全匹配。

案例 C：Nonce 谓词反转（I2，Logto；假设不匹配模式：非状态谓词）。Logto 的 OIDC 单点登录连接器仅在上游令牌包含 nonce 声明时才对其进行验证，而不是在会话于授权请求中发送 nonce 时始终进行验证。攻击者通过在伪造的 id_令牌中省略 nonce，完全绕过了检查。PORTA 生成一个在请求中包含 nonce 的身份验证，然后向回调提交一个不含 nonce 声明的 id_令牌；代理接受了该令牌。

影响：随机数原本用于将 id_令牌绑定到授权请求并防止重放，但现在已失效。

这直接违反了 OIDC Core §3.1.3.7.11。

案例 D：通过 SAML 联邦绕过多因素认证（I3，Casdoor；假设不匹配模式：分支覆盖缺口）。Casdoor 会在密码和 OAuth 登录路径上强制执行多因素认证，但不会在 SAML 联邦路径上调用多因素认证检查。因此，已启用多因素认证的用户可以通过 SAML 上游身份提供商完成登录，而无需提供第二因素。PORTA 构造一个已启用多因素认证的用户，先通过密码路径执行控制登录（该路径会正确要求多因素认证），然后通过 SAML 路径执行相同的登录，而该路径一步即可返回完全认证会话。

影响：多因素认证变成可选项，即攻击者在宽松的上游身份提供商处获取用户密码和联邦身份后，可以选择联合路径绕过本地第二因素要求。

NIST 800-63C §5.1 要求不得利用联合身份认证绕过身份验证要求。

案例 E：通过未验证电子邮件自动关联实现账户接管（I4，Logto；规范不充分）。我们纳入此案例，是为了说明规范不充分缺陷也可能对应着完全可复现的端到端攻击。只要电子邮件地址匹配，Logto 的单点登录处理程序就会将传入的单点登录身份关联到现有本地账户，而不要求上游身份提供方设置 email_verified。我们将此问题归类为规范不充分，因为没有任何单点登录规范以规范性条款要求账户关联必须满足 email_verified。攻击者利用管理员已在 Logto 租户中配置为受信任单点登录连接器的上游身份提供方发起攻击。攻击者在该身份提供方注册 victim@corp.com（该身份提供方不会验证电子邮件所有权），并通过该身份提供方登录；Logto 会悄然将攻击者控制的单点登录身份关联到受害者现有的本地账户，使攻击者在下次登录时获得完整的会话访问权限。PORTA 通过以下方式捕获这一问题：先以受害者身份进行密码登录，然后

提交来自模拟身份提供商的单点登录登录请求，该请求声明受害者的电子邮件地址，但不包含 email_verified；随后观察由此生成的会话是否解析为相同的本地用户标识符。

影响：任何邮箱地址可在宽松的第三方身份提供商处注册的本地 Logto 账户都可能遭到账户接管。

7.4 代理规范建议

根据表 8 报告的模式，我们提出一系列针对代理规范的高级建议，分为三类：缺失字段、配置变更和身份冲突。

缺失字段。 如果当前代理策略要求检查时间、受众、随机数或断言条件，则必须拒绝缺少所需字段的输入。字段缺失不得导致跳过检查。这项建议源于发现项 I2 和 I5：当字段缺失时，验证会被跳过。

配置变更。如果在登录完成之前，提供商、信任、客户端或多因素认证配置发生变更，代理必须在签发会话之前重新检查受影响的条件，否则就应拒绝该流程。这项建议源于 I3 发现结果：登录会在旧配置下继续进行。

身份冲突。来自不同上游身份提供商的身份必须保持区分，除非明确的关联规则声明它们属于同一账户。如果使用电子邮件进行关联，该规则必须要求提供已验证电子邮件声明。这项要求源于 I4 身份冲突发现结果以及 Logto 案例研究（§7.3）。

8 讨论与局限性

序列生成器的假设覆盖率。

序列生成器通过抽样攻击策略来探索不变量遭到违反的情形，而不是枚举所有可能的违反方式。

例如，在某个序列中，映射器对电子邮件大小写的规范化不一致，因此两个账户被映射到同一身份，可能会触发针对 I4 的预言机。

然而，对于同一个代理，PORTA 可能无法识别出会因用户标识符冲突而触发同一 I4 不变量的场景。

召回实验。 由于该实验展示了对一部分已知 CVE 的检测率，因此它只能作为 PORTA 假阴性率的下限估计。此外，PORTA 当前的实现也受到其实现选择的限制。

具体而言，CVE-2018-14637 需要一个已过期的断言，而当前尚不支持这种情况，因为 PORTA 的构建器始终会生成过期时间在未来的断言。

不变量范围。 这五个不变量是根据具有代表性的 60 个通用漏洞披露集合推导得出的。其他语料库可能会发现额外的代理需求，或以不同方式归类相同案例。PORTA 会测试本文定义的这五个不变量，但这些不变量可能无法涵盖所有可能的代理安全属性。

默认配置范围。 所有发现结果均针对固定版本的最简配置部署。企业安装通常会引入自定义属性映射器、声明转换脚本、外部用户存储和组成员资格插件。在默认配置下满足全部五个不变量的平台，在定制后可能无法满足这些不变量。我们的结果描述的是每个平台代理所提供的转换逻辑，而不是任何特定生产部署的安全性。

9 相关工作

协议级形式化分析。大量研究为单点登录协议角色建立了形式化安全保证。Fett 等人基于 Web 基础设施模型，为 OAuth 2.0 [8] 和 OpenID Connect [9] 建立了全面的模型，之后又将其扩展至金融级 API [7] 和 FAPI 2.0 [16]。在 SAML 方面，Groß 分析了浏览器/工件配置 [13],，Armando 等人则对基于 SAML 的 Google Apps 单点登录流程进行了形式化分析并攻破了该流程 [2]。这些研究在假设各角色行为正确的对手模型下，为单个协议角色（身份提供方、依赖方、授权服务器、客户端）定义了安全属性。它们既没有建模代理同时承担多个角色时会发生什么，也没有定义代理在验证–提取–映射–签发转换边界上所应履行的正确性义务。我们的工作正是针对这一空白：我们探讨双重角色代理在跨协议边界将上游证明转换为下游会话时必须保持哪些不变量。

单点登录实证安全与实现缺陷。Wang 等人 [36] 以及 Sun 和 Beznosov [35]开展的流量引导研究表明，商业化部署的 OAuth 和 OpenID 实现充斥着逻辑错误。SSOScan [39] 和 SAAT [10] 等自动化依赖方侧测试工具随后证实，集成故障在首次披露多年后仍然存在。Ghasemisharif 等人 [11] 采用了互补性的实证方法，测量了已部署网站中的单点登录账户劫持和会话管理故障。Somorovsky 等人的 XML 签名包装攻击研究 [34] 是我们 I1（证明完整性）不变量的直接技术先驱：该研究证明，“已验证对象 ≠ 被使用对象”的故障模式在 SAML 实现中具有系统性。我们的工作沿着这一方向从两个方面进行了扩展：我们将研究范围从单个不变量和单一协议推广到五个通用不变量，并将分析单元从单个依赖方或身份提供方实现转移到代理的转换流水线。

身份提供商测试、合规性与恶意身份提供商攻击。 Mainka、Mladenov 和 Schwenk [25] 提出了恶意身份提供商攻击者模型，并用它研究了 OpenID 和 OpenID Connect 部署中的单点登录客户端。他们随后发表的 SoK [26] 系统梳理了 OIDC 安全属性及其评估方法。OAuch [29] 依据标准和最佳实践要求测试 OAuth 授权服务器，展现了可执行合规判定器的价值。之后，Philippaerts 等人将这一方法扩展到开源身份提供商中间件的评估 [30]。Bisegna 等人 [3] 推出了 Micro-Id-Gym，用于测试身份管理实现；Yang 等人 [37] 则对 JWT 库实现进行模糊测试，以发现时序和签名验证缺陷。PORTA 与 OAuch 一样秉持合规判定器理念，但它针对的是跨协议、跨角色的转换流水线，而非单一角色的协议合规性。OAuch 的检查清单围绕授权服务器对单个依赖方所负的义务构建，因此不涉及向被测系统提供输入的上游身份提供商，也无法测试多步骤流程原子性或跨 IdP 命名空间隔离。这

我们发现的 33 个违反 I3 和 I4 不变量的发现项中，有 11 个恰恰属于 OAuch 合规模型无法表示的类别。

经纪式与中介式单点登录。最接近的先前研究直接针对经纪层。Innocenti 等人 [18] 研究了公共 Web 上的代理式单点登录，量化了经纪服务部署普及率，并指出经纪服务、上游身份提供商与下游网站之间的外部重定向链组合中存在漏洞。Luo 等人 [24] 提出了跨应用令牌攻击，并为 OAuth 2.0 平台开发了 COVScan 测试工具；中介层表面上与之相似，但其目标是应用到平台的账户关联，而非经纪服务内部转换链。Jannett 等人的 DISTINCT 攻击 [19]利用双单点登录窗口之间的浏览器内通信，针对的是浏览器端组合，而非服务器端经纪逻辑。我们的工作与这些研究互为补充：Innocenti 等人研究经纪服务如何与其环境组合，而我们研究代理的内部转换逻辑是否维持了我们定义的五个不变量。据我们所知，尚无先前研究针对跨多个平台和协议绑定的经纪服务转换流水线内部所涉及的正确性义务，提供基于实现的系统规范。

10 结论与未来工作

身份代理在身份提供方与下游服务之间实现信任转换函数。

在本项工作中，我们从 40 项规范性协议要求和 60 个既往通用漏洞披露中，形式化定义了五项转换不变量（I1–I5），并构建了 PORTA，用于测试已部署的代理是否满足这些不变量。

我们将 PORTA 的发现结果归纳为 33 项此前未报告的经纪服务安全缺陷。修订时，其中 29 项已获确认，14 项已修补，19 项已分配 CVE 标识符。至少 10 项已分配的漏洞属于严重级别。

未来，我们计划与标准组织合作，将 I3 和 I4 不变量纳入规范，因为我们的工具发现，它们的大多数违规目前尚未被规范涵盖。此外，我们计划将转换模型从单一代理扩展到经纪服务链，以便研究不变量违规如何在多个转换跳之间传播。

致谢

本研究部分由 NSERC 探索资助项目 RGPIN-2026-07548 和 NSERC-CSE 研究社群资助项目 ALLRP 588144-23 提供资助。OpenAI 慷慨提供了大语言模型访问权限的额度。David Lie 获加拿大一级研究讲席 CRC-2019-00242 和 Schwartz Reisman 技术与社会研究所的支持。通过 NSERC-CSE 研究社群资助项目获得资助的研究人员不代表加拿大通信安全局或加拿大政府。他们作为本倡议一部分开展的任何研究、发表的任何意见或持有的任何立场，均不代表加拿大政府的官方观点。

参考文献

[1] 2026. PORTA：工件。https://github.com/dlgroupuoft/PORTA。[2] Alessandro Armando、Roberto Carbone、Luca Compagna、Jorge Cuellar 和 Llanos Tobarra。2008 年。SAML 2.0 Web 浏览器单点登录的形式化分析：破解 Google Apps 的基于 SAML 的单点登录。载于 第六届 *ACM* 安全工程中的形式化方法研讨会论文集 （美国弗吉尼亚州亚历山德里亚）（*FMSE '08*）。美国纽约州纽约市计算机协会，1–10。doi:10.1145/1456396.1456397

[3] Andrea Bisegna、Matteo Bitussi、Roberto Carbone 和 Silvio Ranise。2024 年。增强身份管理实现的安全测试：引入 Micro-Id-Gym 语言和 Micro-Id-Gym 测试工具。IEEE Security and Privacy 22, 6 2024 50 61 10 1109 2024 3450277（2024 年 11 月），–。doi:[4] 2024。/MSEC。。Casbin。。Casdoor：开源身份与访问管理平台。https://github.com/casdoor/casdoor。访问日期：2026-04-08。[5] Yi Chen、Di Tang、Yepeng Yao、Mingming Zha、XiaoFeng Wang、Xiaozhong Liu、Haixu Tang 和 Baoxu Liu。2023 年。Sherlock on specs：通过自动推理构建 LTE 一致性测试。载于第 32 届 USENIX安全研讨会论文集（美国加利福尼亚州阿纳海姆）（SEC '23）。USENIX协会，美国，第 198 号文章，共 17 页。[6] Dex IdP。2024 年。Dex：OpenID Connect（OIDC）身份和 OAuth 2.0 提供商。https://github.com/dexidp/dex。访问日期：2026-04-08。[7] Daniel Fett、Pedram Hosseyni 和 Ralf Küsters。2019 年。OpenID 金融级 API 的全面形式化安全分析。载于 2019 年 IEEE安全与隐私研讨会（SP）。453 471 10 1109 2019 00067 –。doi:。/SP。。Daniel [8] 2016 Fett、Ralf Küsters 和 Guido Schmitz。。OAuth 2.0 的全面形式化安全分析。载于第 2016 届 ACM SIGSAC计算机与通信安全会议论文集（奥地利维也纳）（CCS '16）。美国纽约州纽约市计算机协会，1204–1215。doi:10.1145/2976749.2978385[9] Daniel Fett、Ralf Küsters 和 Guido Schmitz。2017 年。Web 单点登录标准 OpenID Connect：深入的形式化安全分析与安全指南。载于 2017 年 IEEE 第 30 届计算机安全基础研讨会（CSF）。189–202。doi:10.1109/CSF.2017.20[10] Mohammad Ghasemisharif、Chris Kanich 和 Jason Polakis。2022 年。面向单点登录部署中账户与会话管理漏洞的自动化审计。载于 2022 年 IEEE安全与隐私研讨会（SP）。1774–1790。doi:10.1109/SP46214.2022.9833753[11] Mohammad Ghasemisharif、Amruta Ramesh、Stephen Checkoway、Chris Kanich 和 Jason Polakis。2018 年。单点注销，你在哪里？对 Web 上单点登录账户劫持和会话管理的实证分析。载于第 27 届 USENIX安全研讨会论文集（美国马里兰州巴尔的摩）（SEC'18）。USENIX协会，美国，1475–1492。[12] Paul A. Grassi、Justin P. Richer、Sarah K. Squire、James L. Fenton、Ellen M. Nadeau、Naomi B. Lefkovitz、Jamie M. Danker、Yee-Yin Choong、Kristen K. Greene 和 Mary F. Theofanos。2017 年。数字身份指南：联合与断言。NIST 特别出版物 800-63C。美国国家标准与技术研究院。doi:10.6028/nist.sp.800-63c [13] Thomas Groß。2003 年。SAML 单点登录浏览器/工件配置文件的安全分析。载于第 19 届年度计算机安全应用会议论文集（ACSAC '03）。IEEE计算机协会，美国，298。[14] Dick Hardt。2012 年。OAuth 2.0 授权框架。RFC 6749。doi:10.17487/RFC6749[15] HashiCorp。2024 年。Vault：基于身份的机密与加密管理系统。https://github.com/hashicorp/vault。访问日期：2026-04-08。[16] Pedram Hosseyni、Ralf Küsters 和 Tim Würtele。2024 年。OpenID FAPI 2.0 协议族的形式化安全分析：伴随标准化过程。ACM Trans. Priv. Secur. 28，1，第 4 号文章（2024 年 11 月），共 36 页。doi:10.1145/3699716[17] Syed Rafiul Hussain、Omar Chowdhury、Shagufta Mehnaz 和 Elisa Bertino。2018 年。LTEInspector：一种针对 4G LTE 的系统化对抗性测试方法。载于第 25 届年度网络与分布式系统安全研讨会，NDSS 2018，美国加利福尼亚州圣迭戈，2018 年 2 月 18–21 日。互联网协会。https://www.ndss-symposium.org/wp-content/uploads/2018 02 2018_ / /ndss 02 3 A-Hussain paper.pdf[18] Tommaso Innocenti、Louis Jannett、Christian Mainka、Vladislav Mladenov 和 Engin Kirda。2025 年。“安全性取决于最薄弱的环节”：论 Web 上经纪式单点登录的安全性。载于 2025 年 IEEE安全与隐私研讨会（SP）。1009 1027 10 1109 61157 2025 00024 –。doi:。/SP。。Louis Jannett、[19] Vladislav Mladenov、Christian Mainka 和 Jörg Schwenk。2022 年。DISTINCT：利用浏览器内通信实施双窗口单点登录中的身份盗窃。载于 2022 年 ACM SIGSAC计算机与通信安全会议论文集（美国加利福尼亚州洛杉矶）（CCS '22）。美国纽约州纽约市计算机协会，1553–1567。doi:10.1145/3548606.3560692[20] Michael B. Jones、John Bradley 和 Nat Sakimura。2015 年。JSON Web 签名（JWS）。RFC 7515 10 17487 7515 。doi:。/RFC Michael B. Jones、John [21] Bradley 和 Nat Sakimura。2015 年。JSON Web 令牌（JWT）。RFC 7519。doi:10.17487/rfc7519[22] Michael B. Jones、Anthony Nadalin、Brian Campbell、John Bradley 和 Chuck Mortimore。2020 2 0 8693 10 17487 8693 。OAuth。令 [23] 2024 牌交换。RFC。。doi:。/rfc Jens Langhammer。。authentik：你所需的身份验证粘合层。https://github.com/goauthentik/authentik。访问日期：2026-04-08。[24] Kaixuan Luo、Xianbo Wang、Pui Ho Adonis Fung、Wing Cheong Lau 和 Julien Lecomte。2025 年。通用跨应用攻击：利用并保护集成平台中的 OAuth 2.0。载于第 34 届 USENIX安全研讨会论文集

研讨会（美国华盛顿州西雅图）（*SEC '25*166，第 18 页）。USENIX协会，美国，第若干文章。[25] Christian Mainka、Vladislav Mladenov 和 Jörg Schwenk。2016 年。不要相信我：利用恶意身份提供商分析和攻击单点登录。载于*2016* 年 *IEEE*欧洲安全与隐私研讨会（*EuroS&P*）。321–336。doi:10.1109/EuroSP.2016.33[26] Christian Mainka、Vladislav Mladenov、Jörg Schwenk 和 Tobias Wich。2017 年。SoK：单点登录安全——OpenID Connect 的评估。载于*2017* 年 *IEEE*欧洲安全与隐私研讨会（*EuroS&P*）。251–266。doi:10.1109/EuroSP.2017.32[27] OASIS安全服务技术委员会。2005 年。*OASIS*安全断言标记语言（*SAML*）*V2.0* 的断言与协议。OASIS标准 saml-core-2.0-os。OASIS。https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf [28] OASIS安全服务技术委员会。2005 年。*OASIS*安全断言标记语言（*SAML*）*V2.0* 的配置。OASIS标准 saml-profiles-2.0-os。OASIS。https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf [29] Pieter Philippaerts、Davy Preuveneers 和 Wouter Joosen。2022 年。OAuch：探索 OAuth 2.0生态系统中的安全合规性。载于第 *25* 届国际攻击、入侵与防御研究研讨会论文集（塞浦路斯利马索尔）（*RAID '22*）。美国纽约州纽约市计算机协会，460 481 10 1145 3545948 3545955 –。doi：。/。Pieter Philippaerts、[30] Jan Vanhoof、Tom Van Cutsem 和 Wouter Joosen。2024 年。你的 OAuth 中间件是否存在漏洞？评估开源身份提供商的安全性。载于*GLOBECOM 2024——2024* 年 *IEEE*全球通信大会。3607–3612。doi:10.1109/GLOBECOM52923.2024.10901500[31] 红帽。2024 年。Keycloak：开源身份与访问管理。https://github.com/keycloak/keycloak。访问日期：2026-04-08。[32] N. Sakimura、J. Bradley、M. Jones、B. de Medeiros 和 C. Mortimore。2023 年。OpenID Connect Core 1.0，纳入勘误集 2。OpenID基金会规范。https://openid.net/specs/openid-connect-core-1_0.html [33] Silverhand Inc. 2024 年。Logto：面向开发者的更优身份基础设施。https://github.com/logto-io/logto。访问日期：2026-04-08。[34] Juraj Somorovsky、Andreas Mayer、Jörg Schwenk、Marco Kampmann 和 Meiko Jensen。2012 年。论破解 SAML：成为你想成为的任何人。载于第 *21* 届 *USENIX*安全研讨会论文集（美国华盛顿州贝尔维尤）（*Security'12*）。USENIX协会，美国，21。[35] San-Tsai Sun 和 Konstantin Beznosov。2012 年。魔鬼藏在（实现）细节中：OAuth单点登录系统的实证分析。载于*2012* 年 *ACM*计算机与通信安全会议论文集（美国北卡罗来纳州罗利）（*CCS '12*）。美国纽约州纽约市计算机协会，378 390 10 1145 2382196 2382238 –。doi：。/。Rui Wang、Shuo [36] Chen 和 XiaoFeng Wang。2012 年。通过 Facebook 和 Google 将我签入你的账户：对商业部署的单点登录网络服务进行流量引导的安全研究。载于*2012* 年 *IEEE*安全与隐私研讨会论文集（*SP '12*）。IEEE计算机协会，美国，365–379。doi:10.1109/SP.2012.30[37] Jingcheng Yang、Enze Wang、Jianjun Chen、Qi Wang、Yuheng Zhang、Haixin Duan、Wei Xie 和 Baosheng Wang。2026 年。令牌定时炸弹：评估 JWT实现以发现漏洞。载于网络与分布式系统安全研讨会论文集（*NDSS*）。美国。doi:10.14722/ndss.2026.24069[38] Ronghai Yang、Guanchen Li、Wing Cheong Lau、Kehuan Zhang 和 Pili Hu。2016 年。基于模型的安全测试：OAuth 2.0实现的实证研究。载于第 *11* 届 *ACM* 亚洲计算机与通信安全会议论文集（中国西安）（*ASIA CCS '16*）。美国纽约州纽约市计算机协会，651 662 10 1145 2897845 2897874 –。doi：。/。[39] 2014 Yuchen Zhou 和 David Evans。。SSOScan：用于单点登录漏洞的 Web 应用自动化测试。载于第 *23* 届 *USENIX*安全研讨会论文集（美国加利福尼亚州圣迭戈）（*SEC'14*）。USENIX协会，美国，495–510。[40] ZITADEL。2024 年。ZITADEL：身份管理平台。https://github.com/zitadel/zitadel。访问日期：2026-04-08。

开放科学

评估本文贡献所需的全部工件均已发布 [1]。

审稿人应注意的依赖项。端到端运行完整流水线需要用于配置提取和序列生成阶段的 OpenAI 应用程序接口密钥；论文中报告的三次运行的总成本约为漏洞发现实验（§6.3）69.5 美元，以及召回实验（§6.2）53.53 美元。不希望获取应用程序接口密钥的审稿人仍然可以通过将保存的 sequences.json 文件重放到正在运行的 Docker 容器中，复现执行和判定阶段；这不需要访问大语言模型。我们在 README.md 中说明了如何执行 PORTA 的重放模式。

伦理考量

我们遵循 ACM CCS 伦理指南。所有实验均针对由我们控制的、运行于隔离 Docker 容器中的开源软件。实验未涉及任何生产系统或用户数据。对于本文及随附工件中报告的所有发现结果，我们均遵循负责任披露实践，并已与相关方完成披露流程。

生成式人工智能的使用

我们将本研究中生成式人工智能的使用分为三类进行披露。

PORTA 中的大语言模型组件*PORTA*。PORTA 流水线（§5）在三个内部阶段使用大语言模型（gpt-5_2-2025-12-1，默认温度，top-p = 1.0）：平台配置文件生成（§5.1）、攻击序列生成（§5.2）和预言机断言生成。每项大语言模型输出都会通过针对运行中的代理实例进行确定性执行来验证：配置文件需经过三次独立运行的多数表决合并，并在使用前针对平台进行验证；序列和预言机断言则通过确定性判定流水线（§5.4）对捕获的 HTTP 跟踪记录进行评估。报告中的任何发现项都不单独依赖大语言模型的判断。

辅助源代码查找的人工智能工具。 在对 252 个候选发现项进行人工分类（§6.3）时，我们使用 Claude 代理作为语义搜索工具，检索各平台的源代码。代理根据 PORTA 运行中捕获的 HTTP 请求/响应跟踪记录，定位与所观察行为相关的源文件和函数。所有根因分析、漏洞分类（规范违规还是规范不充分缺陷）、不变量和代码级模式的归类，以及最终包含 33 个发现项的目录，均由作者手动完成。

写作辅助。 我们使用生成式人工智能工具对论文文本进行语法和风格润色。对于经人工智能工具改进或检查的所有技术内容、主张、图表、表格和论证，作者均已在提交前进行审阅和核实。

通用漏洞披露编号

CVE-2026-15611, **CVE-2026-15612**,CVE-2026-15615, **CVE-2026-15616**, **CVE-2026-15617** , CVE-2026-41577 , CVE-2026-55669 , CVE-2026-56664 , CVE-2026-56665 , CVE-2026-56666 , **CVE-2026-9090** , CVE-2026-9091 , **CVE-2026-9092** , **CVE-2026-9093** ,**CVE-2026-9094** , CVE-2026-9095 , CVE-2026-9096 , **CVE-2026-9097**,**CVE-2026-9098** 加粗的条目表示通用漏洞评分系统 ≥ 9.0（严重）。