PORTA: Systematic Discovery of Vulnerabilities in Identity Brokers

Zixu Zhou
jasonz.zhou@utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Xiangyu Guo
xiangyu.guo@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Ilya Grishchenko
ilya.grishchenko@utoronto.ca
University of Toronto
Toronto, Ontario, Canada

David Lie
david.lie@utoronto.ca
University of Toronto
Toronto, Ontario, Canada

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

Abstract

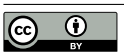
Identity brokers such as Keycloak, Vault, and Dex translate authentication proofs from an upstream identity provider into sessions for downstream services. Due to their central role in authentication and trust mediation, a single vulnerability in a broker compromises every downstream service behind it. In fact, recent incidents have demonstrated that even widely adopted platforms are not immune to such issues, highlighting the critical need for systematic analysis. To build such an analysis, this work first defines invariants for broker operation. These invariants are not completely covered by existing authentication specifications, primarily because brokers are unique in their duality, i.e., acting simultaneously as a relying party and an identity provider. The invariants capture the properties that identity brokers must uphold throughout their workflows, as all 60 previous CVEs we consider violate at least one invariant. Within these workflows, we further define high-level action categories. Equipped with this information, our automated tool PORTA leverages a large language model (LLM) to generate API invocation sequences corresponding to the high-level action categories, and oracle assertions aligned with the specified invariants. PORTA identifies 33 previously unreported security vulnerabilities in seven popular open-source brokers. We study these novel vulnerabilities and trace them to eight recurring patterns. We responsibly disclosed all discovered vulnerabilities: 29 have been confirmed, 14 patched, and 19 assigned CVE numbers, with 10 CVEs rated as critical.

CCS Concepts

• **Security and privacy → Authorization**; **Web protocol security**; **Web application security**; *Software security engineering*.

Keywords

identity broker, single sign-on, OAuth, OpenID Connect, security testing, security invariants, authentication, trust translation, large language models



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846557

ACM Reference Format:
Zixu Zhou, Xiangyu Guo, Ilya Grishchenko, and David Lie. 2026. PORTA: Systematic Discovery of Vulnerabilities in Identity Brokers. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846557

1 Introduction

During Single Sign-On (SSO), enterprises and cloud tenants do not authenticate users separately against each identity provider (IdP), each of which may rely on its own authentication protocol—such as Security Assertion Markup Language (SAML) [27] for enterprise federation, OpenID Connect (OIDC) [32] for web and mobile SSO, and Open Authorization (OAuth) [14] for API authorization. Instead, organizations typically operate an internal IdP that allows employees to authenticate once and subsequently reuse that authentication across different systems. This process involves relaying the initial authentication event across heterogeneous protocols. Such relaying requires translating an authentication proof from one trust domain into a valid session within another.

The software that performs this translation is an *identity broker* [6, 15, 23, 31, 33, 40]. Specifically, a broker receives an upstream authentication proof (e.g., a SAML assertion or an OIDC token), verifies its cryptographic integrity, extracts the identity and authorization claims it carries, maps those claims to a downstream principal, and issues a new credential for the downstream service.

Recent measurement work [18] confirms that brokered SSO is pervasive on the internet and that a single broker may sit in front of dozens of relying parties (the applications and services that accept the broker's tokens) at once.

Naturally, a bug inside an identity broker affects every downstream service that trusts it. Moreover, the downstream services cannot detect or defend against the failure because by the time they see the broker's output, the upstream proof has already been replaced. The broker's internal credential-processing logic is therefore a single point of failure.

At the same time, broker operations violate a crucial assumption present in all authentication protocol specifications, namely, that each entity within an authentication protocol is assigned a single role. Consider a clause from the SAML specification [27]: "if `NotOnOrAfter` is present, the relying party MUST reject the assertion after that time." For a relying party (RP) that gets assertions from a single trusted IdP, honoring the IdP's choice to omit the

field is correct. For a broker, honoring the absent-field case enables an attacker to strip the field and disable the check. Indeed, what distinguishes an identity broker from a conventional relying party or IdP is its *dual role*: it simultaneously acts as a relying party to an upstream IdP and as an IdP to downstream clients.

The single role assumption is also assumed by existing analyses of authentication protocols by Fett et al. and others [7–9] which study security properties of the authentication protocols. Other prior works either test single authentication protocol role (OAuch [29] for OAuth authorization servers, COVScan [24] for app-platform OAuth) or study the external interaction between brokers and their environment [18].

In this work, we study identity brokers as systems that combine an upstream relying party role with a downstream identity provider role.

First, we derive a set of five security invariants from a convergent analysis of 40 normative requirements across eight SSO specifications and 60 previous CVEs for ten broker platforms. Each invariant traces to at least one specification clause and is violated by CVEs in at least two independently developed brokers. We do not claim that these 5 invariants are a complete definition of correctness for broker operation, but rather that they are useful for detecting security vulnerabilities.

To detect invariant violations at scale, we design an automated approach with four stages: (1) *platform adaptation* uses an LLM to extract a platform profile from the broker's API documentation, mapping abstract operations to concrete HTTP endpoints and their parameters; (2) *sequence generation* uses an LLM to produce attack sequences and oracle assertions from the invariant definitions and the platform profile; (3) *execution* runs each sequence against a live broker instance in an isolated Docker container; and (4) *verdict* mechanically evaluates the oracle assertions against captured HTTP responses.

We implement this approach as a tool we call PORTA, and evaluate it on seven broker platforms.

We first evaluated PORTA on a subset of the previous CVEs, selecting 11 applicable CVEs affecting the seven chosen broker platforms. Across three independent runs, PORTA found 10 of the 11 CVEs.

We further conducted a vulnerability discovery campaign to identify previously unknown vulnerabilities. Using the same experimental setup, we performed three independent runs, each completing within 3 hours. PORTA produces 926 invariant violations before automatic deduplication. Deduplication within each run retains 252 violations, of which manual review confirms 226 and rejects 26. Source code analysis then maps the 226 confirmed findings to 33 distinct security flaws across the three runs. 15 of these flaws violate normative requirements in the authentication protocol specifications. The remaining 18 are security-significant flaws where the invariant is violated but the relevant specification does not explicitly mandate a check, pointing to specification incompleteness in the dual-role broker context.

We studied newly identified vulnerabilities and observed similar failure patterns each recurring in two to six independently developed broker codebases, written in four programming languages by separate teams. We attribute these recurrences to *assumption mismatches*: vulnerable code preserves coding habits that are correct under the assumption that all parties are single-role, which is not the case for the dual-role identity brokers as discussed above. In total, we identify eight distinct assumption-mismatch patterns and assign each of the 33 findings a dominant pattern.

From 33 findings, 15 (45%) have root causes that do not match any CVE in the selected corpus, indicating that the invariant set generalizes beyond the CVEs from which it was derived. We responsibly disclosed all violations to the affected maintainers; 29 have been confirmed, 14 patched, and 19 were assigned a CVE identifier. At least 10 of them were rated as critical.

In summary, the paper makes the following contributions:

- We derive a set of five security invariants for identity broker dual-role operation from specifications and 60 previous CVEs.
- We design an automated approach that uses an LLM to generate sequences and oracle assertions from invariant definitions, and evaluates the assertions deterministically against captured platform behavior.
- We implement the approach as a tool PORTA and evaluate it on seven broker platforms, finding 10 of the 11 previously reported applicable security flaws and identifying 33 previously unreported security flaws.
- We categorize the discovered vulnerabilities into eight cross-platform failure patterns driven by a single role broker operation assumption. Responsible disclosure led to 29 acknowledgments, 14 patches, and 19 CVE assignments; at least 10 of them are critical.

The source code of PORTA and the evaluation artifacts are available [1].

2 Background

Identity Broker. An identity broker sits between an external Identity Provider (IdP) and one or more downstream services. Unlike a plain relying party that only *consumes* an upstream identity proof, a broker both consumes and *re-issues*: it verifies the upstream credential, extracts the identity and authorization claims it carries, maps those claims to the downstream domain, and signs a fresh credential for the downstream service. This dual role, acting as a Service Provider (SP) toward the upstream IdP and as an IdP toward downstream services, is what makes brokers both powerful and fragile. A failure in either half is directly exploitable: a verification mistake lets an attacker forge upstream proofs, and an issuance mistake can hand out downstream sessions with inflated privileges. Platforms such as Keycloak, Authentik, and Dex are widely deployed in this role, mediating enterprise SSO across dozens of downstream apps [18].

Identity Broker Protocols. Brokering appears in two main protocol contexts. In *SAML brokering*, the broker receives a signed SAML 2.0 assertion from the upstream IdP, validates its signature and conditions, and issues its own downstream assertion or OIDC token [27]. In *OIDC and JWT federation*, the broker obtains an upstream ID token via the OAuth 2.0 Authorization Code Flow [14] or validates a directly presented JSON Web Tokens (JWT) [21], then issues a downstream session. Token exchange [22] is a third pattern in which the broker translates one token type into another within a single protocol. The wire formats differ, but the security obligations

are the same: the broker must not accept what it cannot verify, must not grant what was not authorized, and must not be manipulated through the sequencing of protocol steps. Figure 1 illustrates the broker's position in the trust chain and its dual SP/IdP role. The shaded region between the upstream proof and the downstream session is the *trust translation boundary*, which is the focus of this paper.

3 Threat Model

Attacker capabilities. We consider three attacker positions, each reflecting a realistic scenario in federated identity deployments. First, the attacker may *operate an upstream IdP* that is registered as a trusted identity source on the broker. In multi-tenant broker deployments, any organization can register its own IdP, so the attacker signs SAML assertions and JWTs with its own keys and the broker is configured to trust those keys. The attacker does not need to recover the cryptographic keys belonging to other IdPs or forge their signatures; all attack credentials originate from the attacker's own IdP. Second, the attacker may *intercept a legitimate upstream proof* (e.g., via a compromised endpoint or a leaked URL) and attempt to replay or modify it. Third, the attacker may be a *legitimate broker user* with limited privileges who attempts to escalate access, for example by exchanging a narrow-scope token for a broader one, or by accessing resources of a different tenant.

Out of scope. The scope of our approach is determined by several assumptions. The broker is correctly configured with trust anchors for one or more legitimate upstream IdPs; the attacker cannot read or modify the configuration, keys, or assertions of those legitimate IdPs. The attacker cannot access the broker's database, file system, or internal network. The attacker cannot modify the broker's source code or configuration. All administrator configurations are trusted, and vulnerabilities due to administrator misconfigurations are out of scope. The attacker cannot access administrative APIs. For instance, if an administrator explicitly enables a permissive setting (e.g., `trust-email=true` or `validateSignature=false`), the resulting behavior is by design, not a vulnerability. Furthermore, we do not test denial-of-service conditions, client-side attacks (XSS, CSRF, clickjacking), or bugs in the upstream IdP itself.

Deployment assumptions. We test each broker in its *default configuration*, with at least one upstream IdP trust correctly provisioned by the administrator. Each test run starts from a clean state: the broker runs in an isolated container that is destroyed and recreated between protocol runs, ensuring no residual state from prior tests.

Attacker goals. All goals are defined from the perspective of the downstream service that trusts the broker's output. The attacker succeeds if they can: (a) authenticate as a different principal (identity spoofing), (b) obtain a session that remains valid after the upstream proof has expired or been revoked (replay), (c) gain permissions exceeding what the upstream IdP authorized (privilege escalation), or (d) take over another user's account through identity collision (account takeover).

4 Trust Translation Invariants for Brokers

To test whether identity brokers correctly translate trust between upstream and downstream domains, we first need a precise definition of what such correctness means for a broker. Existing protocol specifications (e.g., SAML, OIDC, and OAuth) define correctness for individual roles, namely what an IdP must assert and what a relying party must verify, but not for the broker that occupies both roles at once. A direct consequence is that broker developers reading these specifications can implement each role correctly in isolation and still produce a translation pipeline that is insecure as a whole. The goal of this section is to fill that gap with a set of correctness properties that (i) are grounded in concrete normative clauses across all major SSO specifications, (ii) are violated by real-world CVEs in multiple independently developed brokers, and (iii) are testable through black-box API interaction.

We call these properties *translation invariants*. The invariants serve three roles in the rest of the paper. First, they make the broker's correctness requirements *explicit*, giving developers, auditors, and standards bodies, a concrete checklist for broker testing. Second, they organize the design of PORTA (§5): each invariant becomes a generation target for attack sequences and a testing target for the verdict pipeline. Third, they provide a structural lens for interpreting our findings (§7): every cross-platform failure family we discover aligns with a specific invariant boundary, which is direct evidence that the gap is structural rather than incidental.

4.1 Formal Model

We model an identity broker as a trust translation function

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

where $\mathcal{A}_{\text{up}}$ is the space of upstream credential artifacts (e.g., SAML assertions, JWTs, and certificates), $\mathcal{A}_{\text{down}}$ is the space of downstream sessions, C is the broker configuration (e.g., trust anchors, claim-mapping rules, and policy constraints), and $\perp$ denotes rejection. F decomposes into four sequential stages shown in Figure 2: Verify checks cryptographic validity and metadata against the trust anchors in C; Extract produces a structured representation of the verified artifact, including the principal identity, authentication context, and authorization claims; Map translates this representation to the downstream domain according to C's mapping rules; and Issue serializes and signs the downstream credential.

Each boundary between these stages introduces a correctness requirement that does not arise in single-role IdP or RP implementations. We formalize these obligations as five security invariants. The function F in Equation (1) represents a single translation hop: one upstream IdP, one broker, and one downstream session. Real deployments often compose multiple hops, in which the output of one broker becomes the upstream input of the next. We restrict the formal treatment in this paper to the single-hop case and leave compositional analysis to future work. The five invariants we define apply to each hop independently and do not by themselves capture cross-hop properties such as transitive trust transfer.

4.2 Derivation Methodology

To derive a principled invariant set, we conducted a convergent analysis of two independent sources.

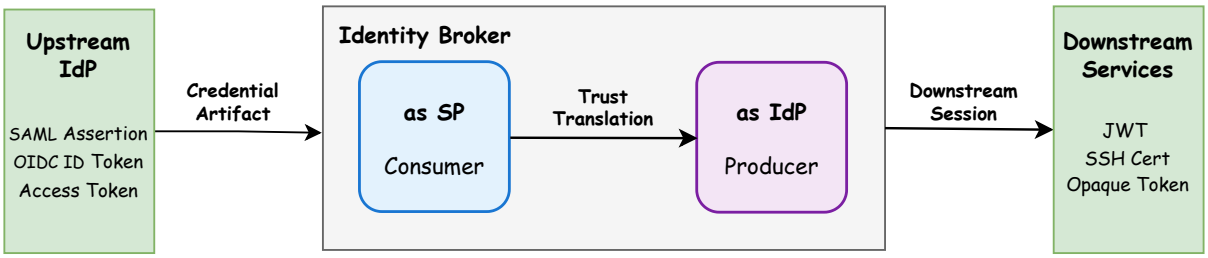


Figure 1: An identity broker simultaneously acts as a Service Provider (SP/Consumer) toward the upstream IdP and as an Identity Provider (IdP/Producer) toward the downstream service. Security of the downstream session depends entirely on the correctness of the *trust translation* step.

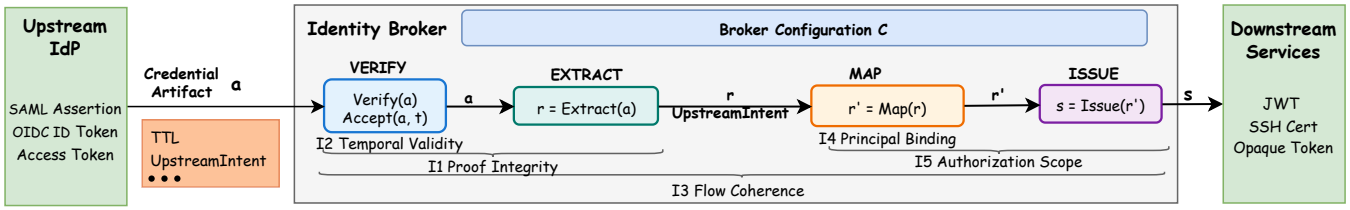


Figure 2: The four-stage trust translation pipeline and where each invariant (I1–I5) applies.

Source 1: Protocol specs. We compiled 40 security-relevant normative requirements (MUST and SHALL clauses) from eight specifications: SAML 2.0 Core [27] and Profiles [28], OpenID Connect Core 1.0 [32], RFC 7519 (JWT) [21], RFC 7515 (JWS) [20], RFC 8693 (OAuth 2.0 Token Exchange) [22], RFC 6749 (OAuth 2.0 Authorization Framework) [14], and NIST SP 800-63C (Digital Identity Guidelines: Federation and Assertions) [12]. These 40 clauses govern authentication proof handling, session issuance, and authorization delegation, which correspond to the stages of F operation.

Source 2: Previous CVEs. We collected 60 broker logic CVEs for 10 identity broker platforms (Keycloak, Dex, Vault, Teleport, Ory Hydra/Fosite/Kratos, Conjur, Shibboleth, and OpenStack Keystone) via NVD, GitHub Security Advisories, and platform changelogs. The full list is in the artifact [1]. We applied strict filtering consistent with our threat model (§3): only authentication and authorization logic flaws were retained; XSS, CSRF, DoS, memory safety, and dependency bugs were excluded. For each CVE, the violated requirement is determined by the *fix commit itself*: the code change reveals which check was absent or incorrect, and this check maps directly to one of the 40 normative clauses above. For example, the fix for CVE-2018-14637 (Keycloak) adds `NotOnOrAfter` validation, mapping unambiguously to SAML Core §2.5.1.2.

Grouping the 60 CVEs by their missing or incorrect security check produced five non-overlapping clusters that collectively account for all 60 CVEs. We refer to the properties violated by each cluster as *invariants* which are formalized in §4.3. We retain a cluster as an invariant only if it satisfies three criteria: 1) it is motivated by at least one citable clause in a protocol specification; 2) it is violated by CVEs in at least two independently developed platforms; and 3) it is testable through black-box API interaction without source-code access.

4.3 Invariant Definitions

We now define each invariant with its formal specification, the pipeline stage it guards (Figure 2), related protocol clauses, and a representative CVE.

We use two kinds of fonts. Broker operations, written in small caps, denote computations the broker actually performs on a credential. Additional functions we defined to concisely stipulate the invariants are in italic.

I1: Proof Integrity (VERIFY → EXTRACT). If the EXTRACT stage finishes successfully when processing the credential artifact a, obtained from the successful VERIFY stage, then the representation r derived from a must match a in content. Credential artifact a and representation r are records that include the following semantic context: the principal identity, the authentication context (e.g., authentication method and assurance level), and the authorization claims (e.g., group memberships). We assume existence of an SC function which outputs semantic context of a and r.

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

This invariant is grounded in SAML Core §5.4.2 (assertions must be integrity-protected), OIDC Core §3.1.3.7 (signature and issuer validation), and RFC 7515 §5.2 (JWS signature verification). I1 has two sub-patterns. *I1-a (Signed ≠ Consumed):* the broker verifies one element but extracts identity from a different, unverified element. This is the classic XML Signature Wrapping attack [34] and its JWT equivalents. *I1-b (Wrong Verifier):* the broker applies the wrong verification policy, e.g., trusting an attacker-controlled JWKS endpoint or accepting a non-CA certificate as a trust anchor.

Representative CVE: CVE-2024-8698 (Keycloak): the SAML signature validation logic verified one XML element, but the broker consumed identity claims from a different, unsigned element in the

same response. An attacker can craft a response containing a legitimately signed element alongside an unsigned element carrying attacker-chosen identity claims; the broker verifies the former but grants a session reflecting the latter, enabling impersonation of any local account.

I2: Temporal Validity (Verify stage). During Verify stage, if the broker accepts a at time $t + \Delta$ where a was issued at time t, then Δ lies within $TTL(a)$. Let k be the number of times the same artifact a has been successfully accepted, including the current acceptance. If a carries `OneTimeUse`, the broker may accept it only when $k = 1$.

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

This invariant is grounded in specifications SAML Core §2.5.1.2 (`NotOnOrAfter`), §2.5.1.5 (`OneTimeUse`), OIDC Core §3.1.3.7 (`exp`, `nonce`), and also in RFC 7519 §4.1.4 (`exp`).

Representative CVE: CVE-2018-14637 (Keycloak): the SAML broker endpoint ignored assertion expiration conditions, allowing replay of expired assertions to mint fresh downstream sessions.

I3: Flow Coherence *(spans entire pipeline).* If the broker issues a downstream session s from representation r', then s must be derived by the complete execution of the pipeline. There must exist an upstream artifact a that the broker successfully verified, a representation r that the broker extracted from a, and a downstream representation r' that the broker mapped from r. The broker must not produce a session unless every preceding stage has succeeded on this same source artifact.

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

This rules out execution paths that skip a stage, accept state spliced from a different session, or bypass a configured authentication factor through an alternative entry point.

This invariant is not a property fully stated in existing specifications. Instead, it is a broker-specific generalization motivated by the following specification clauses: OIDC Core §3.1.2.1 (`state`), §3.3.2.6 (single-use authorization codes), SAML Profiles §4.1.4.2 [28] (specifically, `InResponseTo` clause), and RFC 6749 §4.1.3 (discussion on binding `client_id`/`redirect_uri`).

Representative CVE: CVE-2021-3827 (Keycloak): the SAML ECP binding path bypassed other authentication flows including MFA, allowing an attacker to obtain a session without completing the required authentication steps.

I4: Principal Binding (Map stage). If the principal identity component of two upstream representations r_1, r_2 are distinct under the broker's configuration (i.e., the configuration does not declare them as the same identity), the broker's Map stage must send them to distinct downstream identities r_1', r_2'. We assume a function *ID* which extracts the principal identity from representation.

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

This invariant is grounded in OIDC Core §2 (`sub` must uniquely identify the end-user), RFC 7519 §4.1.2 (`sub`), and RFC 8693 §2.1 (`subject_token` identifies the acted-upon principal).

Representative CVE: CVE-2020-16250 (Vault): the AWS IAM auth method accepted manipulated identity values, allowing one AWS principal to authenticate as a different one.

I5: Authorization Scope (Map → Issue). If the broker extracts representation r from credential artifact a, maps r to representation r' (with $ID(r')$ principal), and issues session s for $ID(r')$, then the permissions carried in s are bounded by the upstream intent attached to a and by what the broker's configuration grants to $ID(r')$. The invariant stipulates that the granted permissions need to be aligned with both the original intent coming from the upstream IdP *UpstreamIntent* and the broker configuration C.

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

Here *UpstreamIntent* denotes a set of permissions that the upstream intent wants to grant, they are encoded into the credential artifact a that the broker receives as input. Further, we assume a function *Policy*, which takes as an input principal ID(r') and the broker configuration C and produces grounded instantiations of the generic permissions in C. Finally, we assume a function *Perms* which extracts a set of permissions from the established session. As a concrete illustration, consider a JWT a whose *UpstreamIntent* wants to grant access to two APIs and specific actions within them: {(finance-api, read:reports), (comp-api, exec:task)}. The broker in our example is configured such that $Policy(C, ID(r'))$ returns {(finance-api, read:reports), (hr-api, write:reports)}. For this example, the invariant I5 allows to grant (finance-api, read:reports) permission. By contrast, granting (comp-api, exec:task) is not permitted by the invariant as non-compliant with the policy, as well as the access to HR functionality (hr-api, write:reports), since it is not in the intent.

This invariant is grounded in SAML Core §2.5.1.4 (`AudienceRestriction`), OIDC Core §3.1.3.7 (`aud` must contain `client_id`), RFC 7519 §4.1.3 (`aud`), and RFC 8693 §2.1 (`audience` and `scope` parameters).

Representative CVE: CVE-2024-5798 (Vault): the JWT auth method did not validate the `aud` claim against the role-bound audience, allowing a token minted for one service to authenticate at another service.

4.4 Coverage and Validation

At least one of the five invariants is violated by each of the 60 CVEs in our dataset. Table 1 shows the invariant distribution over the 60 surveyed CVEs. Invariant I1 is the largest group (19 CVEs, 7 platforms), consistent with the long history of signature-related vulnerabilities in both SAML [34] and JWT processing. Invariant I5 is the second largest (18 CVEs), reflecting the widespread difficulty of enforcing scope constraints across token exchange and claim-mapping boundaries. Invariants I2 and I3 are smaller but span 4–5 platforms each. A potential concern is circularity: if the invariants are derived from CVEs, using them to find bugs may merely find those same CVEs.

Table 1: Distribution of 60 CVEs across the five invariants. "B #" counts independently developed broker codebases. "S #" counts the normative requirements each invariant traces to across all eight specifications.

Invariant	CVE	B #	S #	Example CVE
I1 Proof Integrity	19	7	10	CVE-2024-8698
I2 Temporal Validity	7	5	8	CVE-2018-14637
I3 Flow Coherence	10	4	9	CVE-2021-3827
I4 Principal Binding	6	2	6	CVE-2020-16250
I5 Authorization Scope	18	4	7	CVE-2024-5798
Total	**60**	**10**	**40**	

However, our evaluation (§6.3) shows that 45% of the resulting findings match patterns with no prior CVE record, concentrated in the two invariants (I3 and I4) with the sparsest CVE coverage. We note that our invariants form only a partial specification of correctness for brokers, we do not claim they are complete.

5 System Design

This section presents PORTA, a system that automatically tests identity brokers for violations of the invariants defined in §4. The core challenge is that each broker exposes a different API, yet the security properties to test (I1–I5) are platform-independent. PORTA addresses this with a four-phase pipeline (Figure 3): (1) *platform adaptation* abstracts each broker's API into a common profile so that test logic is written once; (2) *sequence generation* uses an LLM to produce potential attack sequences and oracle assertions from the invariants; (3) *execution* runs each sequence against a live broker instance; and (4) *verdict* evaluates captured behavior against the oracle to decide whether a violation occurred. The LLM participates only in Phases 1 and 2; Phases 3 and 4 are fully deterministic.

5.1 Platform Adaptation

Brokers under test can use different admin APIs, endpoint paths, and authentication flows. Writing test sequences directly against each API would require separate logic for every platform, and adding a new platform would mean writing new tests from scratch. To address this challenge, PORTA introduces *abstract operations*. An abstract operation is a generic operation that must be instantiated for each platform. Although the instantiated operation's name and invocation mechanism may differ across platforms, the abstract operation produces the same semantic effect in each case, such as successfully creating a user account. PORTA defines 33 abstract operations (e.g., `saml_login`, `token_exchange`, `create_user`).[1] Nine of the 33 operations come from protocol specifications like RFC 6749 and the remaining 24 are administration operations shared by all seven popular broker platforms that we studied. The full list with specification references is in the artifact [1].

Having a set of abstract operations, our method automatically generates a *platform profile* for each platform. The profile connects each abstract operation to a platform's HTTP method, URL path, request parameters, and response fields (Step ❶ in Figure 3). Specifically, for each platform, an LLM receives that platform's API documentation and produces the profile in a fixed structured format (see the artifact [1] for the prompt skeleton). Because the LLM may produce incorrect endpoint paths or parameters, profile generation runs three times per platform and the three outputs are merged: each endpoint's HTTP method and path are decided by exact-string majority vote (at least two of three runs must agree), and body parameters and response fields are taken as a union. If all three runs disagree on a path, the operation is withheld and validated against the live broker before use. This case did not arise on any of the seven platforms we tested.

[1]Note: it is pure coincidence that PORTA defines 33 abstract operations and happens to find 33 vulnerabilities.

5.2 Invariant-Guided Sequence Generation

In the generation step, the LLM receives three inputs (Step ❷ in Figure 3): the platform profile, which lists the available abstract operations and how to call them on this platform; the target invariant definition (§4.3), which states the property the attack must violate; and one manually written example sequence per platform that we compose to illustrate the structural format the generator is expected to produce, including setup steps and execution steps. The example only provides the LLM with one generic sequence schema.

First, for each invariant I1–I5 and each platform, the LLM is asked to propose a pair of abstract operation sequences: a candidate attack sequence that probes this invariant for the given platform (e.g., replaying the same SAML assertion a second time to test I2), and compliant sequence that establishes the expected baseline (e.g., a correct SAML login that produces a valid downstream session). Next, after obtaining abstract-operation sequences, the LLM is tasked to resolve them to platform-specific concrete HTTP requests using the profile (§5.1). This two-layer abstract/concrete architecture follows the model-based testing paradigm and has been used in security research to test multiple implementations of the same protocol [5, 17, 38].

To avoid redundant execution, generated sequences are deduplicated before dispatch, e.g., sequences targeting the same invariant with structurally identical attack steps. Each campaign targets 10 sequences per invariant per protocol binding, yielding approximately 100 candidate sequences per platform before deduplication.

Alongside the sequences, the LLM produces an *oracle assertion*. An oracle assertion is an instantiation of a generic invariant as a boolean expression over values captured from the executed sequence. For example, the body of the I4 invariant $D(p_1, p_2) \implies \textsc{Map}(p_1) \neq \textsc{Map}(p_2)$ can be instantiated, for a sequence that logs in two distinct upstream principals through the same broker, as assertion `entity_id_first_user != entity_id_second_user`.

5.3 Execution Engine

After the platform-specific sequence and invariant resolution (§5.2), we execute LLM-generated sequences against a live broker to produce evidence. The Execution Engine runs each sequence against a live broker instance in an isolated Docker container (Steps ❸–❹ in Figure 3). Before each run, the container is destroyed and recreated from a pinned image (Table 2 lists all versions), ensuring a clean state with no residual sessions, tokens, or configuration from prior tests.

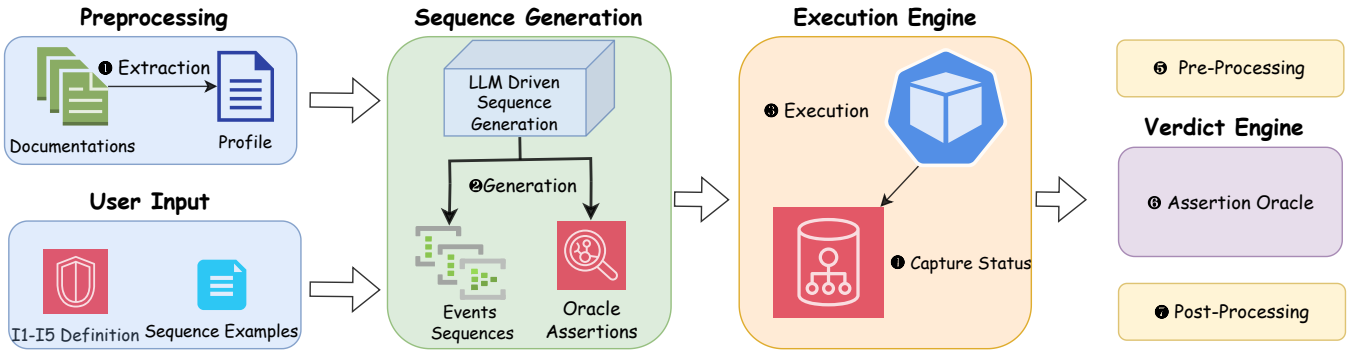


Figure 3: System overview of PORTA pipeline.

During execution, the engine issues HTTP requests and chains values captured from one response into subsequent requests, for example feeding an authorization code returned by the authorize call into the following token exchange.

On each step that requires a credential, the engine produces it on demand. For SAML, the engine generates a signed assertion. For JWT, it mints a token whose standard claims defined in RFC7519[21] (e.g., `iss`, aud, exp, and sub) take values the sequence specifies. Each credential is signed by a trusted key for the compliant sequence and by an attacker-controlled key for the attack sequence. Platform adapters handle protocol-specific flows such as OIDC authorization code exchange and multi-step IdP intent APIs without exposing these details to the sequence generator.

Transport errors and template-substitution failures abort the sequence and exclude it from all finding counts. A step in the compliant sequence that returns a non-200 status, however, still does not terminate the sequence. The co-generated oracle assertion observes the failed step and emits the corresponding finding for manual triage.

5.4 Verdict Engine

PORTA's verdict engine evaluates oracle assertions. Each oracle assertion is a boolean expression, co-generated with the concrete sequences, that encodes what must be true if the invariant holds.

Because the assertion is the per-sequence instantiation of the target invariant (§5.2), a false assertion corresponds to an invariant violation by the broker. The assertion oracle covers all five invariants and accounts for the majority of vulnerabilities discovered by PORTA.

Pre-processing. PORTA has a pre-processing step before engaging the verdict engine. As each HTTP response arrives, PORTA applies lightweight syntactic checks that flag violations visible on a single response, without waiting for the sequence to complete (Step ❺). These include server errors (HTTP 5xx) on well-formed inputs, an issued JWT token whose `iss` claim does not match the broker's configured trusted issuer, and an OIDC `id_token` missing the `nonce` that was authorized. This pre-processing reports only violations grounded directly in RFC-level requirements and contributes a small fraction of overall findings.

Post-processing. Apart from the verdict engine, PORTA has two post-processing sources of findings—fallback and mutation sweep. For invariants I4 and I5, PORTA adds a fallback that does not rely on the LLM-generated assertions (Step ❼). It queries the broker's user-info or introspection endpoint and compares the principal identity and the permissions in the upstream artifact against those carried by the session the broker actually granted. A mismatch in identity flags I4 and a mismatch in permissions flags I5.

PORTA also applies a manually curated library of 102 credential mutations (provided in the artifact [1], grouped by targeted invariant) to the concrete compliant sequence. We create this library, drawing from the 60 CVEs (§4.2) and documented SSO attack patterns such as XML Signature Wrapping [34] and JWT tokens with the `alg=none` header that signals an unsigned token. Each mutation is applied individually so that acceptance of the mutated credential isolates which specific validation check the broker omits. Examples include removing a signature, changing an issuer, setting an expiration time in the past, replacing an audience, and stripping the SAML `<Conditions>` element. The library complements the testing with LLM-generated sequences. It contains mutations that aim to trigger violations of each of five invariants.

The mutation sweep reuses the compliant sequence produced in Phase 2 and changes one credential at a time. Each mutation targets a single element–users, providers, clients, organizations, policies, and authentication state—before performing a login, SAML callback, or token exchange. This lets PORTA test configuration changes, MFA paths, identity collisions, and token exchange across organizations.

6 Evaluation

6.1 Experimental Setup

We test seven open-source identity broker platforms across two protocol bindings (OIDC/JWT and SAML), yielding 13 distinct platform/protocol configurations (Vault's SAML authentication is available only in the Enterprise edition; we test the open-source Community edition, which supports OIDC only). Table 2 lists the platforms, versions, and GitHub stars as a proxy for deployment prevalence. Each platform runs in an isolated Docker container recreated from a pinned image between protocol runs.

We run three independent experiments, each using a separately generated set of platform profiles (§5.1), so that three-run variance

Table 2: Target platforms tested by PORTA.

Platform	Version	Protocols	GitHub Stars
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

reflects the full pipeline including LLM variation in profile generation. Each experiment targets 10 sequences per invariant per protocol binding.

All LLM phases (profile generation, sequence generation, and oracle assertion generation) use a single configuration: `gpt-5_2-2025-12-1`, default temperature, top-$p = 1.0$. The three runs are fully automated from sequence generation to verdict assignment. Automatic finding deduplication is performed within each run. Identifying the source code relevant to the vulnerability-triggering sequence, consolidation into common root causes, and report preparation are performed manually across runs.

6.2 Recall on Known Vulnerabilities

To measure recall, we evaluate PORTA on a subset of the 60-CVE corpus. In this corpus, 34 CVEs are for the applications in our test set (Keycloak, Dex, and Vault). Of those, 18 CVEs involve OIDC or SAML, the protocols that PORTA supports. However, seven of these require a protocol binding, concurrency, an external actor, or an operation that the executor currently does not support. This leaves 11 CVEs used for the recall experiment. The artifact [1] gives details on the selection process. For each selected CVE, the recall experiment uses a pinned known-vulnerable, pre-patch version of the affected system, as documented in the artifact [1].

In the three independent experiments, PORTA generates a mean of 453.0 ± 23.6 sequences per run after removing structurally identical sequences before execution (§5.2).

We consider a CVE found only when a generated sequence triggers the behavior described in the CVE report. Table 3 reports 10 of 11, 7 of 11, and 8 of 11 CVEs found in the three experiments. The union detects 10 of 11 CVEs (90.9%) for this subset and 10/18 (55.6%) end-to-end when all 18 OIDC and SAML CVEs are considered. CVE-2024-5798 is detected only in one of the three rounds. The only CVE never detected is CVE-2018-14637. In this case, the generated tests were intended to use expired SAML assertions. However, the assertion builder set `NotOnOrAfter` 30 minutes after assertion creation, so the assertions were still valid when sent and the vulnerability was not triggered. The API cost of the recall experiment is $53.53.

6.3 Vulnerability Discovery Results

In our three-run evaluation on the selected platforms (§6.1), PORTA produces 10 sequences per invariant per protocol binding, with a mean of 500.7 ± 12.5 sequences per run after filtering structurally identical sequences (§5.2).

The total API cost for all three runs is $69.5.

Table 3: Detection of known CVEs in three independent experiments. Y and N indicate whether PORTA detected the CVE. The last column gives the number of experiments in which it was detected.

CVE	Exp. 1	Exp. 2	Exp. 3	Found
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
Detected	**10/11**	**7/11**	**8/11**	

Table 4: Evaluation of PORTA on seven broker platforms across three independent experiments. *Sequences*: # of sequences generated in the experiment. *Execution rate*: Percent of finished sequences. *Findings:* # of candidate vulnerabilities before deduplication. *Deduplicated:* # of candidate vulnerabilities after deduplication. *True/False positives (TP/FP):* # of vulnerabilities confirmed/rejected by manual triage. *Precision:* $TP/(TP + FP)$.

	Exp 1	Exp 2	Exp 3	$\mu \pm \sigma$
Sequences	492	515	495	500.7 ± 12.5
Execution rate	72.4%	69.5%	76.0%	72.6 ± 3.3%
Findings	206	439	281	308.7 ± 118.9
Deduplicated	54	96	102	84.0 ± 26.2
True positives	44	89	93	75.3 ± 27.2
False positives	10	7	9	8.7 ± 1.5
Precision	81.5%	92.7%	91.2%	**88.5 ± 6.1%**

Across three independent runs (Table 4), PORTA detects 926 candidate vulnerabilities. PORTA then uses an automatic deduplication deterministic pass that collapses findings sharing the following values: mutation rule + invariant (for mutation-sweep findings); invariant + verdict path + hash of the assertion text (for assertion violation findings). After deduplication, we obtain 252 candidate vulnerabilities (84.0 ± 26.2 per run).

We attribute each candidate finding to a primary detection layer that fired first in the pipeline order. The verdict engine with assertion oracles (Step ❻) is the main mechanism with 70.0 ± 21.2 findings per run, about 83% of all findings. The post-processing (Step ❼) has two components: the fallback contributes 7.0 ± 4.6 findings and the mutation sweep produces 4.0 ± 1.7 findings per run. The pre-processing (Step ❺) contributes the remaining 3.0 ± 1.0 per run.

We performed manual analysis of 252 candidate vulnerabilities and confirm 226 of them, which corresponds to precision of 88.5 ± 6.1% with 26 false positives in total across the three runs. We further performed root-cause analysis and identified that 226 confirmed

vulnerabilities are enabled by 33 distinct security flaws in the code discussed in §7. We further identified that 26 false positives have four distinct root causes, the false-positive breakdown is in §6.4.

Table 5 summarizes the per-platform vulnerabilities breakdown. We split all identified flaws into two categories: *specification violations* and *under-specified vulnerabilities*. This classification reflects specification strength, not exploitability: some under-specified flaws, such as identity collision via unverified email are easier to exploit in practice than some specification violations.

We also match each root cause vulnerability against the 60 CVEs: a finding is "matched" if any CVE's fix commit adds the same validation check that our finding shows is missing.

Specification violations and under-specified vulnerabilities. A flaw is a *specification violation* if the observed behavior contradicts a normative MUST requirement in the specification. For example, for temporal validity (I2), Authentik's JWT client-assertion endpoint accepts a token whose `exp` claim is ten minutes in the past (specification violation: the claim is present but ignored). Of the 33 root-cause vulnerabilities, 15 are specification violations. Examples include: SAML assertion replay due to absent anti-replay caches (I2, Keycloak, Casdoor, and Dex); SAML AudienceRestriction not validated (I5, Zitadel, Casdoor, and Authentik); OIDC issuer verification disabled (I1, Authentik); JWT client assertion expiration not enforced in the token endpoint (I2, Authentik); and OIDC nonce bypass via inverted check logic (I2, Logto).

Union of the three runs covers all 15 specification violations, 12 of the 15 are found by at least two runs, and 5 of the 15 are found by all three runs. The three flaws found by only one run are two Authentik OIDC temporal-validity flaws (I2) found only in the first run (where the LLM generated the needed expiration-bypass sequences), and one Casdoor SAML proof-integrity flaw (I1) found only in the second run.

Under-specified flaws are insecure in practice but backed by optional or ambiguous normative language in the specification.

For example, Zitadel's JWT handler skips expiration checking only when `exp` is absent (under-specified: RFC 7519 marks `exp` as optional).

Of 33 identified root-cause vulnerabilities, 18 are under-specified. Notable examples include: non-atomic configuration enforcement creating race windows (I3, Dex and Keycloak), cases where absent `<Conditions>` elements disable audience validation, a behavior the specification permits but does not recommend (I5, Keycloak and Dex), and identity collision through case-insensitive principal matching (I4, Vault).

Of the 18 under-specified flaws, all 18 are found by at least two runs and 9 of the 18 are found by all three runs, a higher rate than specification violations, reflecting that under-specified flaws are triggered by broader attack patterns that the LLM generates reliably across independent runs.

Matched and non-matched patterns. During CVE matching, we compare each identified code defect against the 60-CVE corpus from §4.2. An example of a "match" is a pair of CVE-2018-14637's patch which fixes Keycloak by adding `NotOnOrAfter` enforcement and our finding that Casdoor also lacks `NotOnOrAfter` enforcement. A finding is "non-matched" if no CVE fix in the dataset addresses the same problem. By this criterion, 15 of the 33 findings are non-matched and the other 18 are matched, although confirmed on platforms where they had not previously been reported (i.e., vulnerability "transferred" across the platforms). The non-matched count splits into 5 of the 15 specification violations and 10 of the 18 under-specified flaws. Non-matched findings concentrate in invariant I3 (six of seven) and I4 (two of four) violations, the two invariants with the sparsest CVE record (Table 1).

Table 5: Findings per platform. *Spec.*: Specification violation (normative MUST). *Under-spec.*: Security flaw is not covered by a specification. *Matched*: Root-cause pattern matches an existing CVE on some platform in the dataset of §4.2. *Non-matched*: Pattern does not match a CVE in the dataset.

Platform	Spec.	Under-spec.	Total	Matched	Non-matched
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
Total	**15**	**18**	**33**	**18**	**15**

Responsible disclosure. We began responsible disclosure to all affected platform maintainers on March 4, 2026. At the time of writing, 29 of the 33 disclosed flaws have been confirmed, 14 patched, and 19 assigned CVE identifiers. At least 10 of the assigned vulnerabilities have CVSS scores of 9.0 or higher, which is Critical severity.

6.4 False Positive Analysis

The false-positive count per run (Table 4) is stable (8.7 ± 1.5, $\sigma/\mu = 17\%$). We consolidate the 26 false positives into four distinct misclassification mechanisms. Specifically, we group FPs by invariant violation which flagged it and the cause of misfire.

Post-processing (10 FPs, I4, Step ❼). The I4 fallback (Step ❼, §5.4) compares the principal in the upstream artifact against the principal the broker granted. For 10 cases on Casdoor, the broker returned the same user under a different display format (e.g., the sent username `i2_logout_user` versus the granted display name `I2 Logout User`), causing the string comparison to trigger even though no identity confusion occurred.

Failed compliant sequence (8 FPs, multiple invariants, Step ❻). Sequence generation step (§5.2) produces a pair of attack and compliant sequences, both equipped with the oracle assertion checking the invariant. When the compliant sequence fails, i.e., returns a 4xx status (typically because the LLM used an incorrect endpoint or missing parameter), it may falsely trigger the oracle. For instance, a check whether the granted scope exceeds the requested one may falsely fire when the token request fails and no requested scope is captured.

Pre-processing (4 FPs, I1, Step ❺). Four FPs are HTTP 500 server errors on malformed inputs (NUL bytes in headers, empty subject claims) that the per-response check (Step ❺) conservatively flagged

as proof-integrity violations (I1). In each case the broker crashed during input parsing and never reached its authentication logic, so no invalid credential was accepted.

Test scaffold (4 FPs, Step ❻). The remaining four FPs arise from test infrastructure failures: Authentik returning an empty identity string and Vault rejecting a JWT due to a missing alias name. The oracle assertion (Step ❻) treats these infrastructure failures as invariant violations because the captured response values deviate from the expected.

6.5 Ablation Study

To measure how the design choices behind PORTA affect its results, we compare the full system against three configurations that turn off parts of the pipeline. These are configuration-level ablations rather than strict component-isolation experiments: each configuration changes a part of the pipeline, so the resulting deltas reflect what changing that part does to a whole system, not what a single part would do in isolation. Each configuration uses the same target platforms as the vulnerability discovery evaluation (§6.3), the same broker profile set, and three independent runs.

Table 6 reports the aggregate results for the three ablation variants of the system. By removing both the invariant definitions and the attack examples we obtain a Blind version of PORTA. It can use only the platform profile and a generic "find security bugs" prompt; the deterministic verdict pipeline remains intact. By keeping the invariant definitions in the generation prompt but removing the manually written examples, we obtain a No-example version. LLM-judge version keeps the full invariant-guided generation but replaces the deterministic verdict pipeline with a single LLM call that decides whether each captured sequence shows a violation. The LLM-judge not only outputs whether the sequence is vulnerable, but also a confidence level for its verdict. The three ablations isolate three independent dimensions: the invariant set determines what the system tests, the examples determine how precisely it tests, and the deterministic oracle determines whether the verdict can be trusted.

Blind. The Blind version produces zero true positives across all three runs, and every one of its 19 findings is a false positive. 74% of those are HTTP 500 server errors mislabeled as proof-integrity violations: with no target invariant, the LLM generates malformed inputs that crash request parsers rather than inputs that exercise authentication logic. This is not a sequence-count problem; Blind actually generates more sequences per run than the full system (636 vs. 501) but executes a smaller fraction of them successfully (56.5% vs. 72.6%), because the unguided sequences target the wrong endpoints with incorrect request formats.

No-examples. No-examples version identifies 26 of 33 findings (a subset of findings in §6.3) on a similar sequence budget, but its execution rate drops from 72.6% to 63.3%, per-run precision drops by 13 points (88.5% to 75.6%), and true positives per run drop by 43% (75.3 to 43.0). The seven findings (in §6.3) No-examples never finds all require multi-step setup or non-obvious parameters, including cross-organization token exchange, the JWT client assertion temporal bypass, and identity collision. Invariant definitions alone are too abstract to steer the LLM toward these patterns.

LLM-judge. Replacing the deterministic verdict pipeline with an LLM judge preserves the generation pipeline unchanged. LLM-judge runs on a comparable number of sequences at a comparable execution rate, so any difference in outcome is attributable to the verdict layer rather than to the inputs. Under these matched conditions, LLM-judge produces nearly three times as many findings per run as the full system (245.3 ± 22.7 vs. 84.0 ± 26.2), but only 20.3 ± 2.1 of those are true positives, for a precision of 8.4% compared to the full system's 88.5%. LLM-judge also misses 21 of the 33 included findings, including all of Casdoor's SAML audience-validation and assertion-replay findings and most of Authentik's I1 and I3 findings. All 12 LLM-judge's findings are a subset of PORTA's findings in §6.3.

Manual triage of false positives reveals that 63% of LLM findings are *fabricated*, meaning the LLM reports concrete attack descriptions (e.g., "intent-splicing bypass," "application-tag enforcement missing") that, on manual verification against the platform source, correspond to no actual security-relevant behavior. The deterministic assertion oracle cannot produce fabricated findings by construction: the assertion space is closed over typed-state boolean expressions written before execution, and every expression is evaluated mechanically against a concrete captured value.

Results in Table 6 aggregate findings across all tested LLM confidence levels. Using a high-confidence setting for the LLM produces the best results, as precision reaches 38.2%, still a 2.3× gap to the full system. In the artifact [1], we detail the results for different confidence configurations and whether adding I1–I5 invariant definitions to the judge prompt can improve the performance. All these configurations underperform compared to the one reported in Table 6.

7 Security Findings and Insights

7.1 Vulnerability Classification

This section studies whether real flaws reported in §6.3 are isolated implementation mistakes, or they follow similar patterns. Specifically, we analyze the 33 vulnerabilities along two axes: which security property is broken (failure families), and which coding assumption enabled the violation (assumption-mismatch patterns). These axes are orthogonal: a family describes *what* went wrong; a pattern describes *why* the code was written that way.

Cross-Platform Failure Families. Table 7 groups the 30 out of 33 findings by the broken security property and shows the seven most prevalent families, each affecting two to six independently developed platforms. Three of the 33 findings (Casdoor's cross-organization token exchange, Casdoor's revoked-token acceptance, and Logto's nonce predicate inversion) are single-instance, thus not aggregated into any family.

The family names describe the broken security property. Families violating invariants I1, I2, and I5 produce mostly specification violations (S) because those invariants have explicit MUST clauses in OIDC, SAML, and JWT specifications. Six of the seven I3 findings and all four I4 findings are under-specified flaws (U) that are not explicitly covered by specifications.

Typical code constructs related to the families (Table 7) are not instances of code reuse. The seven platforms are written in four different programming languages (Java, Go, Python, TypeScript) by

Table 6: Ablation results across three independent runs.

Config	Generation	Verdict	Sequences/Run	Exec. Rate	Findings/Run	TP/Run	Precision	# of findings
Blind	generic prompt	deterministic	636.0 ± 1.0	56.5 ± 3.1%	6.3 ± 3.5	0.0	0.0%	0 / 33
No-examples	invariant only	deterministic	492.0 ± 7.9	63.3 ± 1.7%	57.0 ± 3.6	43.0 ± 6.0	75.6 ± 11.5%	26 / 33
LLM-judge	invariant + examples	LLM	475.3 ± 7.0	71.5 ± 2.2%	245.3 ± 22.7	20.3 ± 2.1	8.4 ± 1.6%	12 / 33
Full	invariant + examples	deterministic	500.7 ± 12.5	72.6 ± 3.3%	84.0 ± 26.2	75.3 ± 27.2	88.5 ± 6.1%	33 / 33

Table 7: Cross-platform vulnerability families. *Inv.*: Violated invariant. *Platforms*: Number of independently developed platforms in the family. *S/U*: Specification violations / Under-specified flaws within the family. *Representative clause(s)*: Corresponding normative clause(s). *Typical code form*: Recurring code construct that produces the flaw.

Family	Inv.	Platforms	S / U	Representative Clause(s)	Typical Code Form
Audience not validated	I5	6	4 / 4	RFC 7519 §4.1.3; SAML §2.5.1.4	`verify_aud: False` or no audience check
Temporal validity	I2	4	3 / 2	SAML §2.5.1.2; RFC 7519 §4.1.4	`if field != nil { validate() }`
Flow / non-atomic config	I3	4	0 / 5	—	Config loaded at flow start, not re-checked
Assertion replay	I2	3	3 / 0	SAML §2.5.1.5	No persistent assertion-ID cache
Issuer not validated	I1	3	2 / 1	OIDC §3.1.3.7.4; SAML §2.2.3	`verify_iss: False` or cert from response
MFA bypass via federation	I3	2	1 / 1	NIST 800-63C §5.1 [12]	MFA check absent in federation code path
Identity collision	I4	3	0 / 4	—	Case-insensitive or email-based linking

independent development teams, and each family includes brokers built on top of unrelated third-party protocol libraries: Casdoor's SAML stack is built on `gosaml2`, Dex on `crewjam/saml`, Keycloak on its in-tree Java SAML core, Authentik on a custom Python SAML parser, Logto on `samlify`, Vault on the HashiCorp `cap/jwt` plugin, and Zitadel on its own SAML SP implementation.

Assumption-Mismatch Patterns. We observed that each newly identified vulnerability is enabled by an *assumption mismatch*: a coding pattern that is correct for a single-role RP but becomes incorrect when the same code runs in a broker. Consider the scenario in which a single-role RP receives credentials from one pre-configured IdP that the operator has explicitly trusted. In this scenario, several coding practices are correct: honoring an IdP's choice to omit an optional field, caching the IdP certificate once at startup, validating only what the active code path needs, and treating the in-flight protocol state as a single transaction. A broker code that applies these practices to upstream credentials treats attacker-supplied inputs as if they came from a pre-configured trusted source, which its threat model prohibits: the upstream IdP may be attacker-controlled, the assertion may have been tampered with, the configuration may have changed mid-flow, or two upstream identities may collide in the broker's local namespace.

We label the 33 bugs by the form of the assumption mismatch. To do this, two authors independently labeled each finding with one dominant assumption pattern by reading the relevant source file (per-finding assignments and source links are in the artifact [1]); disagreements in 3 cases were resolved by discussion (Cohen's $\kappa = 0.89$; all three disagreements were between silent skip and explicit disabling, the two most similar categories; details in the artifact [1]). The pattern set was not fixed in advance, and converged to the eight categories shown in Table 8. The two largest groups of findings by assumption-mismatch patterns—silent skip on absent field (9 findings) and explicit verification disabled (8 findings)—together account for slightly over half of all findings. Both arise directly from carrying RP-style "honor what is given" semantics into a context where what is given is attacker-controlled. The next largest are stale configuration snapshot (5 findings) and cross-namespace identity collapse (4 findings). The I3 findings concern broker behavior over a complete flow, while the cited specifications state only the local checks within that flow. The I4 findings concern cross-IdP namespace isolation, which no single-role specification defines. These two groups therefore produce mostly under-specified flaws (§7.2).

An assumption mismatch category is orthogonal to security family. A single family can mix multiple patterns (the audience family combines explicit disabling and silent skipping), and a single pattern can break multiple families (silent skipping breaks audience, temporal validity, and condition handling at once). The full per-finding mapping appears in the artifact [1].

7.2 Under-Specified Flaws

The specification-strength split across the 33 flaws reveals which invariant classes are least addressed by current specifications and practice. All four I4 flaws and six of seven I3 flaws are under-specified, meaning the violated requirement is either optional, undocumented, or debatable. The remaining I3 flaw violates a normative requirement clause. By contrast, I2 (temporal validity and replay) has seven corresponding specification violations out of ten, and I5 (audience and authorization scope) has five specification violations out of nine. This gradient reflects the degree of specification coverage: I1, I2, and I5 map to explicit MUST clauses, so violations are clear-cut. Contrastingly, the I3 invariant generalizes several normative clauses and is therefore not stated explicitly in the existing specifications. Consequently, most violations of this invariant are expected to be classified as under-specified. I4 corresponds to cross-IdP namespace isolation, which no specification addresses. These are the most structurally under-specified invariant classes in the identity broker ecosystem.

Table 8: *Assumption-mismatch Patterns* by which single-role RP assumptions leak into dual-role broker code. *# of Findings*: The number of bugs that share the pattern. *RP Assumption*: The defensive coding assumption that is correct for a single-role relying party. *Broker Impact*: The consequence of the mismatch.

Assumption-mismatch Pattern	# of Findings	RP Assumption	Broker Impact
Silent skip on absent field	9	Honors IdP omission of optional field	Attacker strips field to disable check
Explicit verification disabled	8	One trusted IdP makes the check redundant	Disabled check is the only namespace separator
Stale configuration snapshot	5	Single-step request has no in-flight state	Multi-step flow spans configuration changes
Cross-namespace identity collapse	4	Within one IdP, normalization is a convenience	Across IdPs, it merges distinct principals
No persistent anti-replay state	3	Browser session is itself non-replayable	Broker accepts the same proof across sessions
Branch coverage gap	2	Local-login path was hardened first	Federation path bypasses the local policy
Trust bootstrap reflexivity	1	Cert is cached from out-of-band metadata	Cert is read from the proof being verified
Non-state predicate	1	In trusted-IdP tests the two predicates agree	Attacker omits the field; predicate degenerates

7.3 Case Studies

We exemplify newly identified vulnerabilities via five case studies. The first four are specification violations. The fifth is an under-specified flaw presented as an end-to-end attack scenario, included to show that under-specified does not mean hard to exploit. For each case, we summarize the broker's behavior, the attack PORTA constructs, and the security impact.

Case A: JWT audience bypass (I5, Zitadel; assumption mismatch pattern: explicit verification disabled). Zitadel's Generic JWT IdP validates signature, issuer, and expiration of upstream JWTs, but never validates the audience claim against the configured downstream service. A JWT signed by the trusted IdP with `aud=service-B` authenticates successfully at a broker configured for `service-A`. PORTA constructs a control login with `aud=service-A` (the service the broker is configured to protect) and an attack login with `aud=service-B` (a different service); the broker erroneously accepts both.

Impact: cross-service lateral movement, i.e., a token issued for one downstream service authenticates at every other service behind the same broker.

This violates RFC 7519 §4.1.3, which requires that each recipient processing a JWT verify that it appears in the `aud` claim.

Case B: Issuer verification disabled (I1, Authentik; assumption mismatch pattern: explicit verification disabled). Authentik's OIDC source handler decodes upstream JWTs with issuer verification turned off. A token from an attacker-controlled IdP with a mismatched `iss` claim is accepted as if it came from the configured trust anchor. PORTA constructs a legitimate login from the trusted IdP and an attack login signed by an attacker IdP, both with the same identity claims; the broker accepts both.

Impact: complete trust anchor bypass, i.e., any issuer is treated as the configured trusted IdP.

This violates OIDC Core §3.1.3.7.4, which requires that the `iss` claim in the token exactly match the provider's configured Issuer Identifier.

Case C: Nonce predicate inversion (I2, Logto; assumption mismatch pattern: non-state predicate). Logto's OIDC SSO connector validates the nonce claim only when the upstream token contains one, rather than whenever the session sends one in the authorization request. An attacker, by omitting the nonce from the fabricated id_token, bypasses the check entirely. PORTA generates an authentication that includes a nonce in the request, then submits an id_token with no `nonce` claim to the callback; the broker accepts the token.

Impact: the nonce, which is intended to bind the id_token to the authorization request and prevent replay, is rendered ineffective.

This is a direct violation of OIDC Core §3.1.3.7.11.

Case D: MFA bypass via SAML federation (I3, Casdoor; assumption mismatch pattern: branch coverage gap). Casdoor enforces multi-factor authentication for password and OAuth login paths but does not invoke the MFA check on the SAML federation path. A user enrolled in MFA can therefore complete login through a SAML upstream IdP without ever presenting the second factor. PORTA constructs an MFA-enrolled user, performs a control login through the password path (which correctly demands MFA), then performs the same login through the SAML path, which returns a fully authenticated session in one step.

Impact: MFA becomes voluntary, i.e., an attacker who recovers a user's password and a federated identity at a permissive upstream IdP can choose the federation path to bypass the local second-factor requirement.

NIST 800-63C §5.1 requires that federation not be used to bypass authentication requirements.

Case E: Account takeover via unverified email auto-linking (I4, Logto; under-specified). We include this case to show that an under-specified flaw can correspond to a fully reproducible end-to-end attack. Logto's SSO handler links an incoming SSO identity to an existing local account whenever the email matches, with no requirement that the upstream IdP have set `email_verified`. We classify this as under-specified because no SSO specification normatively requires `email_verified` for account linking. The attack proceeds through an upstream IdP that the administrator has already configured as a trusted SSO connector on the Logto tenant. The attacker registers `victim@corp.com` on that IdP (which does not verify email ownership) and logs in through it; Logto silently links the attacker-controlled SSO identity to the victim's existing local account, giving the attacker full session access on the next login. PORTA captures this by performing a password login as the

victim, then submitting an SSO login from a mock IdP that asserts the victim's email without `email_verified`, and observing that the resulting session resolves to the same local user identifier.

Impact: account takeover for any local Logto account whose email is registrable on a permissive third-party IdP.

7.4 Recommendations for Broker Specifications

Based on the patterns reported in Table 8, we derive a set of high-level recommendations for broker specifications grouped into three categories: missing fields, configuration changes, and identity collisions.

Missing fields. If the active broker policy requires a time, audience, nonce, or assertion condition check, an input that omits the required field must be rejected. Omission must not result in a skipped check. This recommendation follows from the I2 and I5 findings where validation is skipped when a field is absent.

Configuration changes. If provider, trust, client, or MFA configuration changes before login completes, the broker must recheck the affected condition before issuing a session or reject the flow. This recommendation follows from the I3 findings where a login continues under an old configuration.

Identity collisions. Identities from different upstream IdPs must remain distinct unless an explicit linking rule declares them to be the same account. If email is used for linking, the rule must require a verified email claim. This requirement follows from the I4 identity collision findings and the Logto case study (§7.3).

8 Discussion and Limitations

Hypothesis coverage of the sequence generator.

The sequence generator samples attack strategies instead of enumerating every way to violate an invariant.

For example, in one sequence, where the mapper normalizes email casing inconsistently, the oracle may be triggered for I4 because two accounts are mapped to the same identity.

However, for the same broker, PORTA may not identify a scenario that triggers the same I4 invariant through a user identifier collision.

Recall experiment. As the experiment demonstrates the detection rate on a subset of known CVEs, it can serve only as a lower-bound estimate of the false-negative rate of PORTA. In addition, the current instantiation of PORTA has limitations due to implementation choices.

Specifically, CVE-2018-14637 requires an expired assertion, which is not currently supported because PORTA's builder always generates an assertion with a future expiration time.

Invariant scope. The five invariants were derived from the representative set of 60 CVEs. Another corpus may identify additional broker requirements or organize the same cases differently. PORTA tests the five invariants defined in this paper, but these invariants may not cover every possible broker security property.

Default-configuration scope. All findings are relative to pinned, minimally configured deployments. Enterprise installations routinely introduce custom attribute mappers, claim transformation scripts, external user stores, and group membership plugins. A platform that satisfies all five invariants under default configuration may fail them after customization. Our results characterize the translation logic each broker platform ships with, not the security of any specific production deployment.

9 Related Work

Protocol-level formal analysis. A substantial body of work establishes formal security guarantees for SSO protocol roles. Fett et al produced comprehensive models of OAuth 2.0 [8] and OpenID Connect [9] using the Web Infrastructure Model, later extended to the Financial-grade API [7] and FAPI 2.0 [16]. On the SAML side, Groß analyzed the Browser/Artifact profile [13], and Armando et al. formally analyzed and broke the SAML-based SSO flow for Google Apps [2]. These results define security properties for individual protocol roles (IdP, RP, authorization server, client) under adversary models that assume correct role behavior. They do not model what happens when a *broker* simultaneously occupies multiple roles, nor do they define the correctness obligations that arise at the verify–extract–map–issue broker translation boundaries. Our work targets exactly this gap: we ask what invariants a dual-role broker must preserve as it translates upstream proofs into downstream sessions, across protocol boundaries.

Empirical SSO security and implementation flaws. Traffic-guided studies by Wang et al. [36] and Sun and Beznosov [35] demonstrated that commercially deployed OAuth and OpenID implementations are rife with logic errors. Automated RP-side testing tools such as SSOScan [39] and SAAT [10] later confirmed that integration failures persist years after initial disclosure. Ghasemisharif et al. [11] took a complementary empirical approach, measuring SSO account hijacking and session-management failures across deployed websites. Somorovsky et al.'s XML Signature Wrapping work [34] is a direct technical ancestor of our I1 (Proof Integrity) invariant: it proved that the "verified object ≠ consumed object" failure mode is systemic across SAML implementations. Our work extends this line in two directions: we generalize beyond a single invariant and single protocol to five generic invariants, and we shift the analysis unit from individual RP or IdP implementations to the broker's translation pipeline.

IdP testing, compliance, and malicious-IdP attacks. Mainka, Mladenov, and Schwenk [25] introduced the malicious-IdP attacker model and used it to study SSO clients across OpenID and OpenID Connect deployments. Their subsequent SoK [26] systematized OIDC security properties and their evaluation. OAuch [29] tests OAuth authorization servers against standards and best-practice requirements, demonstrating the value of executable compliance oracles. Philippaerts et al. later extended this approach to evaluate open-source identity provider middleware [30]. Bisegna et al. [3] introduced Micro-Id-Gym for testing identity management implementations, and Yang et al. [37] fuzzed JWT library implementations for temporal and signature validation flaws. PORTA shares the compliance-oracle philosophy with OAuch, but targets the cross-protocol, cross-role translation pipeline rather than single-role protocol compliance. OAuch's checklist is built around an authorization server's obligations to a single relying party, so it has no notion of an upstream IdP feeding the system under test and cannot test multi-step flow atomicity or cross-IdP namespace isolation. The

11 of our 33 findings that violate I3 and I4 invariants are precisely the classes OAuch's compliance model has no representation for.

Brokered and intermediary SSO. The closest prior work targets the broker layer directly. Innocenti et al. [18] study brokered SSO on the public Web, quantifying broker deployment prevalence and identifying vulnerabilities in the *external* redirect-chain composition between brokers, upstream IdPs, and downstream sites. Luo et al. [24] introduce cross-app token attacks and the COVScan tester for OAuth 2.0 platforms; the intermediary layer is superficially similar, but the target is app-to-platform account linking rather than the intra-broker translation chain. Jannett et al.'s DISTINCT attack [19] exploits in-browser communication between dual SSO windows, targeting the browser-side composition rather than the server-side broker logic. Our work is complementary: where Innocenti et al. study how brokers are *composed* with their environment, we study whether the broker's *internal* translation logic preserves the five invariants we define. To our knowledge, no prior work provides a systematic, implementation-grounded specification of the correctness obligations that arise inside the broker's translation pipeline across multiple platforms and protocol bindings.

10 Conclusion and Future Work

Identity brokers implement a trust translation function between IdPs and downstream services.

In this work, we formalized five translation invariants (I1–I5) from 40 normative protocol requirements and 60 previous CVEs, and built PORTA to test whether deployed brokers satisfy them.

We consolidated PORTA's findings into 33 previously unreported broker security flaws. At the time of revision, 29 have been confirmed, 14 patched, and 19 assigned CVE identifiers. At least 10 of the assigned vulnerabilities have critical severity.

For future work, we plan to engage with standards bodies to lift the I3 and I4 invariants into specifications, as most of their violations identified by our tool are not currently covered. Further, we plan to extend the translation model from single brokers to chains of brokers so that invariant violations can be studied as they propagate across multiple translation hops.

Acknowledgements

Funding for this work was provided in part by NSERC Discovery Grant RGPIN-2026-07548 and NSERC-CSE Research Communities Grant ALLRP 588144-23. Credits for LLM access were generously provided by OpenAI. David Lie is supported by Tier 1 Canada Research Chair CRC-2019-00242 and the Schwartz Reisman Institute for Technology and Society. Researchers funded through the NSERC-CSE Research Communities Grants do not represent the Communications Security Establishment Canada or the Government of Canada. Any research, opinions or positions they produce as part of this initiative do not represent the official views of the Government of Canada.

References

[1] 2026. PORTA: Artifact. https://github.com/dlgroupuoft/PORTA.

[2] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. 2008. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering* (Alexandria, Virginia, USA) *(FMSE '08)*. Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/1456396.1456397

[3] Andrea Bisegna, Matteo Bitussi, Roberto Carbone, and Silvio Ranise. 2024. Enhancing Security Testing for Identity Management Implementations: Introducing Micro-Id-Gym Language and Micro-Id-Gym Testing Tool. *IEEE Security and Privacy* 22, 6 (Nov. 2024), 50–61. doi:10.1109/MSEC.2024.3450277

[4] Casbin. 2024. Casdoor: An Open-Source Identity and Access Management Platform. https://github.com/casdoor/casdoor. Accessed: 2026-04-08.

[5] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, XiaoFeng Wang, Xiaozhong Liu, Haixu Tang, and Baoxu Liu. 2023. Sherlock on specs: building LTE conformance tests through automated reasoning. In *Proceedings of the 32nd USENIX Conference on Security Symposium* (Anaheim, CA, USA) *(SEC '23)*. USENIX Association, USA, Article 198, 17 pages.

[6] Dex IdP. 2024. Dex: OpenID Connect (OIDC) Identity and OAuth 2.0 Provider. https://github.com/dexidp/dex. Accessed: 2026-04-08.

[7] Daniel Fett, Pedram Hosseyni, and Ralf Küsters. 2019. An Extensive Formal Security Analysis of the OpenID Financial-Grade API. In *2019 IEEE Symposium on Security and Privacy (SP)*. 453–471. doi:10.1109/SP.2019.00067

[8] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2016. A Comprehensive Formal Security Analysis of OAuth 2.0. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1204–1215. doi:10.1145/2976749.2978385

[9] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2017. The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. 189–202. doi:10.1109/CSF.2017.20

[10] Mohammad Ghasemisharif, Chris Kanich, and Jason Polakis. 2022. Towards Automated Auditing for Account and Session Management Flaws in Single Sign-On Deployments. In *2022 IEEE Symposium on Security and Privacy (SP)*. 1774–1790. doi:10.1109/SP46214.2022.9833753

[11] Mohammad Ghasemisharif, Amruta Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. 2018. O single sign-off, where art thou? an empirical analysis of single sign-on account hijacking and session management on the web. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, USA, 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, and Mary F. Theofanos. 2017. *Digital Identity Guidelines: Federation and Assertions*. NIST Special Publication 800-63C. National Institute of Standards and Technology. doi:10.6028/nist.sp.800-63c

[13] Thomas Groβ. 2003. Security Analysis of the SAML Single Sign-on Browser/Artifact Profile. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*. IEEE Computer Society, USA, 298.

[14] Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749. doi:10.17487/RFC6749

[15] HashiCorp. 2024. Vault: Identity-Based Secrets and Encryption Management System. https://github.com/hashicorp/vault. Accessed: 2026-04-08.

[16] Pedram Hosseyni, Ralf Küsters, and Tim Würtele. 2024. Formal Security Analysis of the OpenID FAPI 2.0 Family of Protocols: Accompanying a Standardization Process. *ACM Trans. Priv. Secur.* 28, 1, Article 4 (Nov. 2024), 36 pages. doi:10.1145/3699716

[17] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society. https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02A-3_Hussain_paper.pdf

[18] Tommaso Innocenti, Louis Jannett, Christian Mainka, Vladislav Mladenov, and Engin Kirda. 2025. "Only as Strong as the Weakest Link": On the Security of Brokered Single Sign-On on the Web. In *2025 IEEE Symposium on Security and Privacy (SP)*. 1009–1027. doi:10.1109/SP61157.2025.00024

[19] Louis Jannett, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. 2022. DISTINCT: Identity Theft using In-Browser Communications in Dual-Window Single Sign-On. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1553–1567. doi:10.1145/3548606.3560692

[20] Michael B. Jones, John Bradley, and Nat Sakimura. 2015. JSON Web Signature (JWS). RFC 7515. doi:10.17487/RFC7515

[21] Michael B. Jones, John Bradley, and Nat Sakimura. 2015. JSON Web Token (JWT). RFC 7519. doi:10.17487/rfc7519

[22] Michael B. Jones, Anthony Nadalin, Brian Campbell, John Bradley, and Chuck Mortimore. 2020. OAuth 2.0 Token Exchange. RFC 8693. doi:10.17487/rfc8693

[23] Jens Langhammer. 2024. authentik: The Authentication Glue You Need. https://github.com/goauthentik/authentik. Accessed: 2026-04-08.

[24] Kaixuan Luo, Xianbo Wang, Pui Ho Adonis Fung, Wing Cheong Lau, and Julien Lecomte. 2025. Universal cross-app attacks: exploiting and Securing OAuth 2.0 in integration platforms. In *Proceedings of the 34th USENIX Conference on Security*

Symposium (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 166, 18 pages.
[25] Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. 2016. Do Not Trust Me: Using Malicious IdPs for Analyzing and Attacking Single Sign-on. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. 321–336. doi:10.1109/EuroSP.2016.33
[26] Christian Mainka, Vladislav Mladenov, Jörg Schwenk, and Tobias Wich. 2017. SoK: Single Sign-On Security — An Evaluation of OpenID Connect. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. 251–266. doi:10.1109/EuroSP.2017.32
[27] OASIS Security Services Technical Committee. 2005. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.* OASIS Standard saml-core-2.0-os. OASIS. https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
[28] OASIS Security Services Technical Committee. 2005. *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0.* OASIS Standard 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, and Wouter Joosen. 2022. OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses* (Limassol, Cyprus) *(RAID '22)*. Association for Computing Machinery, New York, NY, USA, 460–481. doi:10.1145/3545948.3545955
[30] Pieter Philippaerts, Jan Vanhoof, Tom Van Cutsem, and Wouter Joosen. 2024. Is Your OAuth Middleware Vulnerable? Evaluating Open-Source Identity Providers' Security. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*. 3607–3612. doi:10.1109/GLOBECOM52923.2024.10901500
[31] Red Hat. 2024. Keycloak: Open Source Identity and Access Management. https://github.com/keycloak/keycloak. Accessed: 2026-04-08.
[32] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. 2023. OpenID Connect Core 1.0 Incorporating Errata Set 2. OpenID Foundation Specification. https://openid.net/specs/openid-connect-core-1_0.html
[33] Silverhand Inc. 2024. Logto: The Better Identity Infrastructure for Developers. https://github.com/logto-io/logto. Accessed: 2026-04-08.
[34] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. 2012. On breaking SAML: be whoever you want to be. In *Proceedings of the 21st USENIX Conference on Security Symposium* (Bellevue, WA) *(Security'12)*. USENIX Association, USA, 21.
[35] San-Tsai Sun and Konstantin Beznosov. 2012. The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 378–390. doi:10.1145/2382196.2382238
[36] Rui Wang, Shuo Chen, and XiaoFeng Wang. 2012. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer Society, USA, 365–379. doi:10.1109/SP.2012.30
[37] Jingcheng Yang, Enze Wang, Jianjun Chen, Qi Wang, Yuheng Zhang, Haixin Duan, Wei Xie, and Baosheng Wang. 2026. Token Time Bomb: Evaluating JWT Implementations for Vulnerability Discovery. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. USA. doi:10.14722/ndss.2026.24069
[38] Ronghai Yang, Guanchen Li, Wing Cheong Lau, Kehuan Zhang, and Pili Hu. 2016. Model-based Security Testing: An Empirical Study on OAuth 2.0 Implementations. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (Xi'an, China) *(ASIA CCS '16)*. Association for Computing Machinery, New York, NY, USA, 651–662. doi:10.1145/2897845.2897874
[39] Yuchen Zhou and David Evans. 2014. SSOScan: automated testing of web applications for single sign-on vulnerabilities. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 495–510.
[40] ZITADEL. 2024. ZITADEL: Identity Management Platform. https://github.com/zitadel/zitadel. Accessed: 2026-04-08.

Open Science

All artifacts needed to evaluate this paper's contributions are released [1].

Dependencies reviewers should be aware of. Running the full pipeline end-to-end requires an OpenAI API key for the profile-extraction and sequence-generation phases; the combined cost for the three runs reported in the paper was approximately $69.5 for the vulnerability discovery experiment (§6.3) and $53.53 for the recall experiment (§6.2). Reviewers who do not wish to obtain an API key can still reproduce the execution and verdict phases by replaying the saved `sequences.json` files against the live Docker containers, which requires no LLM access, we mention how to execute the `replay` mode of PORTA in `README.md`.

Ethical Considerations

We follow the ACM CCS ethics guidelines. All experiments target open-source software running in isolated Docker containers under our control. No production systems or user data were involved. We have followed responsible disclosure practices for all findings reported in this manuscript and the accompanying artifact, completing the disclosure process with the relevant parties.

Generative AI Usage

We disclose generative AI usage in this work in three categories.

LLM components within PORTA. The PORTA pipeline (§5) uses an LLM (`gpt-5_2-2025-12-1`, default temperature, top-p = 1.0) for three internal phases: platform profile generation (§5.1), attack sequence generation (§5.2), and oracle assertion generation. Every LLM output is validated by deterministic execution against a live broker instance: profiles are subjected to majority-vote merging across three independent runs and validated against the platform before use; sequences and oracle assertions are evaluated by the deterministic verdict pipeline (§5.4) over captured HTTP traces. No reported finding depends on the LLM's judgment alone.

AI-assisted source code lookup during triage. During the manual triage of the 252 candidate findings (§6.3), we used Claude agents as a semantic search tool over each platform's source code. Given a captured HTTP request/response trace from a PORTA run, the agent located the source file and function relevant to the observed behavior. All root-cause analysis, vulnerability classification (specification violation versus under-specified flaw), assignment to invariant and code-level pattern, and the resulting 33-finding catalog were performed manually by the authors.

Writing assistance. We used generative AI tools for grammar and style polishing of paper text. All technical content, claims, figures, tables, and arguments, that were improved or checked by AI tools, has been reviewed and verified by the authors before submission.

CVE numbers

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** The bold entries indicate CVSS ≥ 9.0 (Critical).