

When Trust Assumptions Fail: A Vulnerability Research Perspective

WHITE PAPER

Author: Samuel K. Thomas - Founder & CEO of QILAHK | Cybersecurity Trust Authority

Security leaders and architects
October 10, 2026

Table of contents

Abstract	3
Executive summary	4
Decisions for product and application security leaders	4
From unexpected behavior to a security finding	5
Standards and the basis for invariant testing	5
An assumption ledger for sensitive workflows	6
Stating security invariants	6
Context substitution as a research strategy	7
State transitions and race conditions	8
Differential testing and disagreement	8
Distinguishing defects from vulnerabilities	9
Reproducibility and evidence quality	10
Root cause and remediation completeness	10
Worked example of a reproducible finding	11
Integrating the assumption ledger	13
A 90 day adoption plan	13
Test selection and execution	14
Finding acceptance criteria	14
Remediation and closure	14
Reporting measures	15
Assessment scope and evidence handling	15
Action checklist	16
Recommended next step	16
References	17

Abstract

Unexpected authentication or authorization behavior becomes a defensible vulnerability finding when it can be connected to a supported security requirement and a reproducible change in authority. This white paper presents an assumption-centered assessment approach for authorized vulnerability research. An assumption ledger records the propositions behind sensitive workflows, their owners, supporting evidence, enforcement locations, and lifecycle conditions. Controlled context substitution, state-transition testing, and differential comparison then examine whether valid evidence remains applicable to the decision being made. A hypothetical cross-organization export workflow illustrates how a research lead can become a bounded counterexample, how alternative explanations can invalidate a finding, and how remediation must restore the invariant across related paths. Practical guidance covers evidence quality, independent reproduction, acceptance criteria, regression coverage, and closure, supported by a phased adoption plan and reporting measures. The approach is a proposed operating framework informed by standards and formal research; it is not presented as an empirically superior method or a report of newly discovered vulnerabilities.

Executive summary

Useful vulnerability findings connect unexpected behavior to a failed security requirement and a demonstrated change in authority. Authentication and authorization defects frequently arise when valid evidence is accepted in the wrong context, when a lifecycle change is not enforced, or when components disagree about who owns a check. Testing syntax and signatures alone cannot establish that these relationships are correct.

Security teams should use an assumption ledger to record the propositions behind sensitive decisions, then test those propositions through controlled substitutions and state transitions. Each finding should identify the invariant, actor, prerequisites, evidence, affected resource, and observed consequence. Independent reproduction and tests of adjacent paths should determine whether the issue and its remediation are understood completely.

This white paper turns that approach into an operating model for application security, product engineering, and authorized vulnerability assessment. It includes a worked counterexample, reporting criteria, and a phased adoption plan. Its examples are hypothetical and its method is not presented as empirically superior to every existing testing approach. The immediate decision is to make authority relationships and reproducible evidence part of finding acceptance and remediation closure.

Decisions for product and application security leaders

Select the workflows where assumption failures would matter most. Prioritize cross-tenant access, administrative operations, account recovery and linking, delegated jobs, and credential acceptance across services. Define the authorized assessment scope and provide controlled identities and resources. Researchers need enough requirement context to distinguish a failed boundary from deliberate sharing or another supported exception.

Adopt a common finding structure. Require a stated security invariant, a feasible actor, the necessary initial permissions, the tested path, the actual decision, and the observed authority consequence. The structure should support narrow findings as well as major ones. A report should not need dramatic impact language to receive attention when its evidence demonstrates a real violation of the approved relationship.

Define who resolves requirement ambiguity. Protocol behavior can be assessed against a selected specification, while business authorization requires an accountable product or resource owner. Researchers should record inferred assumptions and obtain clarification when those assumptions determine the finding. A disagreement over intended behavior should remain visible until an owner establishes the requirement; it should not be settled solely by the implementation's current output.

From unexpected behavior to a security finding

Unexpected behavior is not sufficient to establish a vulnerability. A researcher may observe a token accepted by an unfamiliar service, a session that persists after logout, or an account linked through a shared attribute. Each behavior could be intentional, mistaken, or exploitable under conditions not yet understood. The research task is to determine which security property applies, what the system actually does, and whether the difference permits an actor to exceed authorized authority.

Trust assumptions provide a productive starting point. Components assume that evidence comes from an approved source, that a principal belongs to the expected tenant, that a request represents the right actor, and that cached state remains relevant. These assumptions are often distributed across libraries, configuration, operational procedures, and business rules. A failure may therefore remain invisible to a test that examines only input syntax or individual functions.

Standards and the basis for invariant testing

Fett et al. (2016) formalize authentication, authorization, and session integrity for OAuth and identify attacks within their web model before proving properties of a corrected design. Fett et al. (2017) apply a related approach to OpenID Connect. The methodological significance is the separation between a desired property, a model of behavior, and a counterexample. A researcher can adopt that discipline without claiming to conduct a complete formal proof.

JWT guidance describes issuer and audience validation, explicit typing, and separate validation rules for different token uses (Sheffer et al., 2020). OAuth security guidance discusses practical threat conditions and defenses for modern deployments (Lodderstedt et al., 2025). These sources demonstrate why valid input can still be inappropriate evidence: acceptance has a purpose and context that syntax and signature checks alone do not establish.

NIST's Secure Software Development Framework recommends integrating security practices into software development and addressing vulnerabilities and their causes (Souppaya et al., 2022). This motivates connecting discovery with remediation and prevention. A report that identifies one rejected string after a patch may establish a local behavior change without showing that the underlying assumption is now enforced across all relevant paths.

The assessment method combines these lessons into an investigation workflow. It does not invent a new protocol or claim that an assumption ledger is a formal verification tool. It provides a practical representation of assumptions, evidence, test outcomes, and unresolved conditions. That representation can help researchers communicate findings and help maintainers decide whether a patch addresses the relevant invariant.

An assumption ledger for sensitive workflows

An assumption ledger is a structured record of propositions on which a security decision depends. Each entry names the proposition, the component relying on it, the authority that can establish it, the evidence supplied, the time interval in which it applies, and the consequence if it is false. The ledger should also identify where the proposition is checked and whether the check can be bypassed by an alternate workflow.

Examples include an assertion being intended for this client, an external subject being mapped to the correct local account, a group membership remaining current, and a service being permitted to act for the represented user. The researcher records these as precise statements rather than broad labels such as trusted token. Precision helps separate independent requirements and prevents one successful validation from concealing several unchecked assumptions.

The ledger distinguishes explicit assumptions from inferred ones. An explicit assumption may appear in protocol documentation, architecture requirements, or policy. An inferred assumption may be reconstructed from code or observed behavior. Inferred assumptions require confirmation before a researcher treats them as the organization's intended security boundary. Otherwise, the investigation can incorrectly label an approved delegation or deliberate shared audience as a defect.

Assumptions also have owners. A library can verify a signature while an application owns tenant mapping. A gateway can establish workload identity while a resource service owns permission checks. Recording ownership makes a failure report more actionable. It also reveals gaps where each team expects another component to perform the same missing check. Those gaps are valuable research leads, but they still require behavioral evidence.

The ledger should remain scoped. Attempting to enumerate every possible assumption across an enterprise can create an unwieldy inventory without useful tests. The assessment method starts from a selected operation, traces the authority needed to perform it, and expands only where the evidence identifies another relevant boundary. This produces a bounded assessment question with a clear success or failure condition.

Stating security invariants

An invariant states a property that should hold across the workflow's relevant states. For example, a principal belonging only to tenant A should not read a private resource owned by tenant B unless an explicit sharing rule permits it. The invariant describes the protected relationship, not the particular implementation. This allows the researcher to assess behavior even when the system uses an unexpected but valid mechanism.

Good invariants include their exceptions. A cross-tenant access rule that ignores approved sharing can generate false findings. A session-termination rule that ignores intentionally independent application sessions can do the same. The researcher should document what authorization permits, which transitions change that permission, and which conditions are relevant to the tested request. Exceptions must be supported by evidence rather than invented after an unexpected result.

An invariant should also identify its temporal meaning. After a membership is removed, access may be required to end immediately or within a documented interval. After a resource changes classification, a new access rule may need to apply before the content becomes available. These are different properties. A test that checks only eventual denial cannot establish correct ordering when the requirement concerns a transition.

The counterexample template is simple: under stated prerequisites, actor X supplied evidence E through path W, the system performed action A against resource R, and that action violated invariant V. The report then provides observations that support each part. This structure makes it easier to identify a missing prerequisite, an unsupported impact claim, or an ambiguity in the expected rule.

A finding need not use mathematical notation to follow this discipline. Clear prose can specify the same relationships. The essential requirement is that another researcher can determine what should have happened, reconstruct the relevant state, reproduce what did happen, and assess the authority difference. A screenshot of unexpected behavior may begin that process, but it rarely completes it.

Context substitution as a research strategy

Many tests corrupt evidence and verify that the system rejects it. Such tests are useful for parsing and integrity checks. Assumption-centered testing adds another class: keep evidence valid while changing the context in which it is presented. This directly examines whether the consumer checks the relationships that make the evidence applicable.

A recipient substitution presents a credential issued for one application to another. An issuer substitution retains a familiar subject identifier while changing the issuing namespace. A purpose substitution presents an identity assertion where an access credential is expected. A tenant substitution changes the resource context while retaining the authenticated principal. Each test needs controlled identities and resources and an explicit invariant defining expected rejection.

RFC 9068 provides a profile for JWT access tokens, while RFC 9207 defines issuer identification in authorization responses (Bertocci, 2021; Meyer zu Selhausen & Fett, 2022). These mechanisms illustrate different contextual dimensions. They should not be reduced to a checklist detached from the deployment. The researcher must identify which profile and issuer relationships the target actually uses before deciding which substitutions are meaningful.

A hypothetical API accepts a valid login assertion from a frontend because both use the same signing authority. The assessment question is whether that assertion meets the API's credential contract and whether acceptance permits an otherwise unauthorized operation. If an independent permission check denies every protected action, the observation may still indicate a validation defect but its demonstrated impact is narrower than full resource compromise. The report should preserve that distinction.

Substitution tests are especially informative at integration boundaries. A gateway may enforce one token profile while a backend route uses another. A newly added endpoint may bypass the common validator. An administrative application may share signing keys with ordinary clients but require

stricter audience checks. The research method follows actual acceptance paths and tests equivalent contexts across them rather than assuming one validator represents the entire system.

State transitions and race conditions

Static tests capture one moment in the workflow. State-transition tests examine what happens when identity, permissions, resource state, or session status changes. They are important because the same credential can be appropriate before a change and inappropriate afterward. The researcher must specify the transition, expected enforcement point, and relevant ordering to distinguish a race from an intentional consistency interval.

Examples include role removal, identity linking, credential rotation, session elevation, logout, and account recovery. A controlled test records the initial state, initiates the change, and observes decisions through each relevant path. The test should distinguish new credential issuance from use of existing credentials and distinguish a local browser session from downstream service authority. Otherwise, a successful denial at one layer can conceal continued authority at another.

Concurrency deserves dedicated tests. Two authentication transactions can run in different tabs, two administrative changes can overlap, and a background job can execute after the initiating session ends. A security invariant must survive the interleavings that the product supports. The researcher should vary timing in a controlled manner and preserve enough evidence to reconstruct the sequence. One isolated unexpected result is a lead, not sufficient evidence of a reproducible race condition.

Negative results require caution. Failure to reproduce a race under a small number of attempts does not establish its absence. A useful report states the tested interleavings, timing controls, and observation limits. If a suspected condition cannot be reliably reproduced, the researcher can still document the uncertainty and supporting code evidence, but should avoid presenting speculative impact as an observed compromise.

Remediation tests should repeat the transition rather than only replay a static request. A patch that rejects one stale token may leave a translated session or queued workflow active. The relevant question is whether the invariant now holds throughout the transition. This may require testing several components and their caches, not merely confirming that the component nearest the original finding returns a new error message.

Differential testing and disagreement

Differential testing compares decisions from components or configurations that should behave equivalently under a stated requirement. Two API routes may protect the same resource class, two gateways may consume the same token profile, or a policy simulator may predict the decision observed at enforcement. Disagreement can reveal a missing check, an undocumented exception, or an incomplete model.

The comparison requires a valid equivalence claim. Two services using the same identity provider can legitimately require different scopes or audiences. Two routes can intentionally expose different

resources. The researcher should not treat disagreement as a vulnerability until the expected relationship is established. Differential testing supplies a signal that directs investigation; the authority analysis determines whether the signal has security significance.

The assessment procedure creates a controlled request matrix across principal, resource, action, tenant, credential type, and lifecycle state. For each combination, the researcher records expected decisions and observes each relevant enforcement path. The matrix can be small and risk-focused. Its value lies in comparing equivalent conditions systematically and making uncovered conditions explicit.

Metamorphic tests offer a related approach when exact outputs are difficult to predict. A transformation that does not change the authorized relationship should preserve the access decision. A transformation that removes the only approved membership should remove the associated access after the required interval. The transformation itself needs justification; changing a resource label might legitimately alter permissions. The expected relation must come from the policy model.

Disagreement should lead to a traceable explanation. The researcher identifies which component used which evidence and policy version and whether it observed the relevant state change. This can reveal a deployment problem rather than a code defect. The report should assign the failure to the appropriate layer and avoid forcing every disagreement into a product-vulnerability category. Correct classification improves remediation and reduces misleading claims.

Distinguishing defects from vulnerabilities

A security defect violates a relevant security requirement. A demonstrated vulnerability additionally shows a feasible way for an actor to obtain unauthorized access, alter protected state, or undermine another protected property under specified conditions. Terminology varies across organizations, but the report should make its evidence and impact clear regardless of the label chosen.

A harmless display inconsistency may not cross any trust boundary. A token accepted by a parser may not reach a protected operation. A long-lived session may follow the documented policy. A filename containing the word bug may identify a regression-test artifact rather than describe the artifact itself as broken. These examples illustrate why context is essential: names and unexpected appearances do not establish failed authority.

Conversely, ordinary-looking behavior can be security-relevant. An account invitation, application consent, or support-assisted reset can change authority through intended interfaces. The assessment question is whether the actor initiating that change meets the required conditions. A vulnerability can therefore involve misuse of normal functions rather than malformed inputs or memory corruption. The failed assumption should be stated explicitly.

Impact claims should follow observed consequences. If the researcher demonstrates reading one controlled private record, the report can state unauthorized read access to the tested resource class under the tested conditions. It should not claim all enterprise data or arbitrary administrative control without additional evidence. Broader potential can be discussed as a hypothesis with clear prerequisites, separate from demonstrated impact.

Alternative explanations strengthen the research when they are tested. The investigator should ask whether a resource was intentionally shared, whether the tested account possessed an overlooked role, whether a cached decision remained within the documented interval, and whether the environment differed from the claimed baseline. Eliminating those explanations makes the finding more credible and helps maintainers reproduce it without debating assumptions that could have been resolved earlier.

Reproducibility and evidence quality

A reproducible report records the environment, versions, relevant configuration, identities, resources, and prerequisite permissions. It distinguishes the initial setup from the steps that trigger the failure. The recipient should be able to recreate the authority relationship without relying on an unexplained account that already has unusual privileges. A minimal setup often clarifies the root cause more effectively than a large production trace.

The evidence should connect requests with decisions and resulting state. Correlation identifiers, sanitized logs, policy snapshots, and timestamps can establish that connection. Screenshots may illustrate outcomes but should not substitute for the request and state evidence needed to reproduce them. Sensitive credentials should be omitted or replaced with controlled test artifacts so that the report itself does not create a reusable access path.

Positive and negative controls are essential. A positive control establishes that the legitimate workflow functions in the baseline. A negative control establishes that an adjacent unapproved relationship is normally denied. If the allegedly vulnerable operation succeeds for every account because the resource is public, the controls can expose that misunderstanding. If the baseline is broken, they can prevent an unrelated error from being misclassified as a security finding.

Independent reproduction provides stronger evidence than repetition by the original researcher alone. Another assessor may notice an unstated prerequisite or configuration difference. The report should record disagreements and resolve them through observable state rather than authority or confidence. When independent reproduction fails, the finding may need narrower scope, additional prerequisites, or reclassification.

Evidence quality also includes uncertainty. The researcher should identify which claims are directly observed, which derive from code analysis, and which remain hypothetical. This is not a weakness in the report; it allows readers to judge conclusions appropriately. A narrow, well-supported finding is more useful than a dramatic claim whose scope cannot be reconstructed from the supplied evidence.

Root cause and remediation completeness

The visible trigger is not always the root cause. A particular token might expose a consumer that lacks recipient validation. Blocking that token's identifier treats the example rather than the missing invariant. A route-specific patch might leave another route using the same unsafe interpretation. Root-cause analysis asks what class of inappropriate evidence or state transition the system failed to reject.

The remediation statement has three parts. It identifies the invariant to restore, the decision point responsible for enforcing it, and the set of paths that depend on that decision. This helps maintainers determine whether the fix belongs in a common validation library, application mapping logic, resource policy, or operational lifecycle process. It also creates a basis for regression testing beyond the original request.

Token exchange illustrates the need for policy-level repair. RFC 8693 defines exchange and actor-related semantics but does not prescribe every deployment's trust model (Jones et al., 2020). A service that receives a new downstream credential still needs a policy determining which authority may be exchanged and which subject it represents. A syntactically correct exchange can therefore require correction in authorization policy rather than protocol parsing.

Regression tests should include adjacent cases with the same failed assumption. If the finding involves one wrong recipient, the tests should exercise other consumers and token purposes that share the relevant acceptance path. If it involves membership removal, the tests should cover translated sessions and asynchronous work. Tests should remain tied to actual supported workflows; inventing unrelated edge cases can obscure the repair's scope.

Operational remediation may be necessary alongside code. Existing sessions might require invalidation, historical grants might require review, and affected logs might need analysis. These actions should follow the demonstrated authority path and incident context. This white paper does not prescribe universal incident steps for every validation defect. It argues that restoring future correctness and withdrawing already issued authority are distinct tasks that a complete assessment should consider.

Worked example of a reproducible finding

Consider a hypothetical service where organization A and organization B store private reports. The service supports explicit sharing, but the test baseline contains no sharing relationship. A controlled account belongs only to A. The researcher defines the invariant that this account cannot retrieve B's private report through any ordinary read route. This statement identifies the principal, resource relationship, action, and relevant exception before testing begins. It does not assume that every cross-organization interaction is forbidden.

The baseline contains two controlled reports and two accounts. The A account successfully reads A's report, while the B account successfully reads B's report. The A account is denied when requesting B's report through the documented route. These controls establish that the resources are private and that ordinary enforcement distinguishes the organizations. The researcher then examines an alternate export route that should preserve the same relationship. The equivalence requirement comes from the product's authorization design, not merely from the similarity of the endpoint names.

Suppose the alternate route produces B's report for the A account. The researcher records the authenticated session, target resource, request path, relevant policy state, and generated result. The report demonstrates access to the controlled B resource, rather than extrapolating to every organization or claiming administrative compromise. The researcher also verifies that no overlooked

group or sharing rule authorized the result. If that verification is incomplete, the finding retains an explicit unresolved prerequisite instead of presenting a definitive authority violation.

Root-cause analysis might reveal that the export route checks authentication but passes a requested organization identifier to a worker without validating membership. The worker's broad service identity can retrieve the report, but that capability does not establish the user's permission. The failed assumption is that a request reaching the worker already carries approved resource scope. The report assigns ownership to the component responsible for creating the work item and documents the worker's reliance on that decision.

A complete fix could require membership and resource checks when the export is submitted and a documented policy for later execution and retrieval. The exact policy depends on the service's requirements. Regression testing should cover the ordinary route, the export route, approved sharing, unshared resources, and membership removal during the job lifecycle. A patch that blocks only the demonstrated report identifier does not restore the invariant. A patch that denies every export preserves confidentiality at the cost of breaking the required feature.

Independent reproduction would recreate the baseline and confirm the same authority difference. After remediation, the evaluator would confirm both rejection of the unapproved operation and success of legitimate exports. The final report would include remaining coverage limits, such as resource classes or background paths not tested. This worked example illustrates how an assumption becomes a verified counterexample, how demonstrated impact stays bounded, and how remediation follows the failed relationship rather than the superficial appearance of the request.

The example also demonstrates how to handle a failed research lead. If independent inspection reveals that the B report was explicitly shared with A, the observed access follows the exception in the invariant. The researcher should withdraw or revise the finding and preserve the explanation. Doing so strengthens the method because it tests the requirement rather than pursuing a predetermined conclusion. A useful research process produces credible dismissals as well as credible vulnerabilities.

If only one deployment reproduces the result, the report should compare configuration and version differences before generalizing. An alternate worker permission, a feature flag, or an older routing path may explain the scope. The eventual finding might apply to an integration pattern rather than every installation of the product. That narrower conclusion can still be valuable when it clearly identifies the conditions under which the boundary fails and the components responsible for restoring it.

The researcher should retain a concise record of the accepted explanation after remediation. Future architectural changes can reintroduce the same assumption even if the original route disappears. A regression description framed around organization membership and approved sharing remains meaningful across those changes; a description framed only around one request string may not. This is the practical advantage of reporting the invariant: it gives maintainers a durable security requirement that can guide later implementation and review without preserving every incidental detail of the original demonstration.

Integrating the assumption ledger

Create the ledger from a selected sensitive operation. Trace the evidence and decisions needed to perform it. Record the expected producer, consumer, principal namespace, recipient, tenant, credential purpose, freshness rule, and permitted delegation where relevant. For each proposition, identify the evidence that establishes it and the component responsible for checking it. Keep the ledger focused enough to generate useful tests.

Connect ledger entries to existing engineering work. An acceptance requirement can become a validation interface, an account-mapping rule, or a resource-policy check. A lifecycle requirement can become an invalidation test. A delegation requirement can become a worker submission and retrieval contract. The ledger should help teams identify gaps in ordinary design and testing rather than create a separate documentation system with no implementation owner.

Update entries when the workflow changes. A new client, route, provider, tenant model, or background service can change the relevant assumptions. Preserve the approved exception and its basis. If an assumption remains unknown, assign an investigation rather than recording a confident expected result. The most useful ledger distinguishes known requirements, implementation observations, and unresolved questions.

A 90 day adoption plan

Days 1 through 30 should establish a small pilot and reporting baseline. Select two or three sensitive workflows and document their invariants with engineering owners. Build controlled accounts, private test resources, and ordinary successful operations. Identify expected denials and approved exceptions. Review a small sample of prior findings to see whether their prerequisites and authority consequences were independently reproducible.

Days 31 through 60 should perform assumption-centered testing. Present valid evidence in adjacent recipient, purpose, issuer, and tenant contexts where the selected workflows make those dimensions relevant. Exercise membership removal, identity linking, session elevation, and delegated execution. Compare equivalent routes and policy predictions with actual decisions. Preserve alternate explanations and distinguish a research lead from a verified finding.

Days 61 through 90 should validate remediation and integrate the method into review. Reproduce accepted findings independently. Confirm that patches restore the invariant across related paths and preserve legitimate operation. Add the relevant cases to regression testing. Use the pilot to simplify ledger fields that do not improve decisions and retain those that clarify responsibility, scope, or reproduction.

Expansion should follow demonstrated usefulness, not the number of ledger entries created. A completed pilot should have clearer requirements, better evidence, and verified fixes. Record assessor effort and false findings as well as accepted defects. These observations help adapt the operating model without claiming a measured improvement that the pilot did not establish.

Test selection and execution

Begin with positive and negative controls. Confirm that the intended user can perform the operation and an adjacent unapproved user is denied. Verify that the test resource is private and that the account lacks overlooked grants. Only then vary the contextual relationship. This avoids reporting a public resource or pre-existing privilege as a newly discovered boundary failure.

Context substitution should preserve the properties not being tested. A correctly issued credential for another recipient is more informative about recipient validation than a randomly corrupted credential. A subject string from another issuer can test namespace handling. An authenticated account requesting another tenant's private resource can test the resource relationship. The intended denial must come from the workflow's requirement, including its supported exceptions.

Transition testing should state the event and required enforcement interval. Record the initial permission state, perform the controlled change, and observe all relevant consumers. Include existing sessions, queued work, and downstream credentials where the workflow uses them. If concurrency matters, vary supported interleavings and retain timing evidence. A result that cannot be reproduced should remain a qualified lead or a narrower code-based concern.

Finding acceptance criteria

A finding is ready for engineering review when its initial state can be recreated, its invariant is supported, and its observed consequence exceeds the actor's approved authority. The report should explain which assumption failed and where it should have been enforced. It should state the demonstrated resource class and operation rather than extrapolating to arbitrary compromise without evidence.

Evidence should connect the request, decision, and resulting state. Include versions, relevant configuration, policy snapshots, correlation identifiers, and sanitized traces as needed. Screenshots can illustrate the result, but the report should not depend on an unexplained screenshot alone. Avoid including reusable secrets or unrelated user information when controlled artifacts can establish the same relationship.

Alternative explanations should be investigated before acceptance. Check approved sharing, overlooked membership, deliberate session independence, allowed convergence intervals, and configuration differences. If an explanation remains unresolved, state it directly. A narrow report with clear uncertainty is more actionable than an expansive claim whose prerequisites cannot be reconstructed.

Remediation and closure

Require a remediation statement describing the invariant restored, the responsible decision point, and the paths covered. A block on one demonstrated identifier or request string may stop the example without correcting the assumption. A shared validator may need repair, but some failures belong in account mapping, resource policy, job submission, or lifecycle management. Assign the correction according to the actual authority path.

Regression testing should include adjacent cases that rely on the same relationship. Recipient confusion calls for relevant neighboring consumers and credential purposes. Cross-tenant export calls for submission, execution, and retrieval checks under the approved job policy. Membership withdrawal calls for previously translated authority. Keep tests tied to supported workflows rather than accumulating unrelated edge cases that obscure the repair.

Close the finding only after both rejection and legitimate functionality are verified. Independent reproduction should confirm the original outcome where feasible and the corrected behavior afterward. If the issue affected already issued authority, assess whether sessions, credentials, or grants need separate withdrawal. Future correctness and existing authority are distinct concerns and should be resolved according to the incident and product context.

Reporting measures

Track findings that meet the invariant and evidence criteria, time to independent reproduction, unresolved requirement questions, false findings, and remediation coverage. Record whether demonstrated impact stayed within the supplied evidence. Track legitimate-flow regressions introduced by fixes. These measures support quality and operational reliability rather than rewarding the largest raw finding count.

Report coverage by sensitive workflow and assumption class. A strong result for recipient validation does not establish correct identity linking or session withdrawal. Describe untested routes, resource classes, and lifecycle paths. Keep assumptions inferred from limited documentation separate from those confirmed by owners. Coverage reporting should help choose the next investigation, not disguise uncertainty behind a single score.

Evaluate the method's burden during adoption. Detailed documentation is useful when it clarifies a complex boundary, but a small workflow may need only a short invariant and reproducible case. Assessors should spend effort where it improves evidence or remediation. Refinement should preserve the independent requirement and observed consequence even when the record becomes shorter.

Assessment scope and evidence handling

Define research authorization separately from the permissions supplied by a test account. The account's technical reach does not determine the permitted assessment scope. Use controlled resources and harmless operations whenever they can demonstrate the relationship. The goal is sufficient evidence of the failure, not maximum access or disruption.

Protect assessment records as sensitive engineering information. They may describe authority paths, internal resource identifiers, and configuration weaknesses. Retain enough evidence for reproduction and regression without unnecessarily keeping secrets or unrelated personal data. Agree on handling, recipients, and retention as part of the assessment process.

Action checklist

- State the protected relationship and its supported exceptions.
- Record assumption ownership and the evidence each consumer requires.
- Establish positive and negative controls before testing a lead.
- Test valid evidence in inappropriate contexts and supported state transitions.
- Bound impact to demonstrated authority and document unresolved conditions.
- Reproduce findings independently and test adjacent paths after remediation.
- Verify that fixes preserve legitimate workflows and address existing authority where needed.

Recommended next step

Pilot the method on a cross-tenant read workflow and a delegated background workflow. Write their invariants with product owners, build controlled test resources, and assess the relevant acceptance and lifecycle boundaries. Use one verified finding or one well-supported dismissal to test the reporting process. The first milestone is a reproducible decision about a concrete authority relationship, followed by a complete correction when a violation is established.

References

- Bertocci, V. (2021). JSON Web Token profile for OAuth 2.0 access tokens (RFC 9068). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc9068.html>
- Fett, D., Küsters, R., & Schmitz, G. (2016). A comprehensive formal security analysis of OAuth 2.0. arXiv. <https://arxiv.org/abs/1601.01229>
- Fett, D., Küsters, R., & Schmitz, G. (2017). The Web SSO standard OpenID Connect: In-depth formal security analysis and security guidelines. arXiv. <https://arxiv.org/abs/1704.08539>
- Jones, M., Nadalin, A., Campbell, B., Bradley, J., & Mortimore, C. (2020). OAuth 2.0 token exchange (RFC 8693). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc8693.html>
- Lodderstedt, T., Bradley, J., Labunets, A., & Fett, D. (2025). Best current practice for OAuth 2.0 security (RFC 9700). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc9700.html>
- Meyer zu Selhausen, K., & Fett, D. (2022). OAuth 2.0 authorization server issuer identification (RFC 9207). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc9207.html>
- Sheffer, Y., Hardt, D., & Jones, M. (2020). JSON Web Token best current practices (RFC 8725). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc8725.html>
- Souppaya, M., Scarfone, K., & Dodson, D. (2022). Secure Software Development Framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST SP 800-218). National Institute of Standards and Technology. <https://csrc.nist.gov/pubs/sp/800/218/final>