

# Trust Boundary Failures in Modern Authentication Systems

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 security and platform leaders     | 4  |
| The authentication handoff problem              | 5  |
| Standards and architectural foundation          | 5  |
| A boundary contract for authentication evidence | 6  |
| Origin and issuer boundary failures             | 6  |
| Recipient and token interpretation failures     | 7  |
| Transaction continuity and session integrity    | 8  |
| Identity mapping and delegation boundaries      | 9  |
| Freshness and lifecycle boundaries              | 9  |
| Design choices and supported exceptions         | 10 |
| Worked example of a federated document service  | 11 |
| Minimum acceptance requirements                 | 13 |
| A 90 day implementation plan                    | 13 |
| Verification that establishes readiness         | 14 |
| Measures and reporting                          | 14 |
| Implementation tradeoffs and exceptions         | 15 |
| Action checklist                                | 15 |
| Recommended next step                           | 15 |
| References                                      | 16 |

## Abstract

Modern authentication systems distribute identity evidence across providers, browsers, applications, gateways, and resource services. Each handoff creates an opportunity to interpret a valid credential beyond the authority it establishes. This white paper examines failures involving issuer selection, recipient binding, credential purpose, account mapping, tenant context, session lifecycle, and delegated authority. Drawing on published standards and security research, it proposes a boundary contract that records the producer, consumer, accepted evidence, applicable context, resulting authority, and withdrawal conditions. An illustrative federated workflow demonstrates how separate checks can preserve assurance across components. Practical guidance translates the analysis into accountable ownership, a 90-day implementation plan, controlled verification, and leadership measures. The central recommendation is to assess the full path from authentication evidence to resource authorization rather than infer broad trust from successful login or signature validation. Examples are analytical and hypothetical; no new experiments or vulnerabilities are claimed.

## Executive summary

Authentication assurance must survive every handoff between the identity provider, browser, application, gateway, and resource service. A valid signature or successful login establishes a bounded fact. It does not establish that the credential belongs to the receiving application, represents the correct tenant, or remains sufficient after a lifecycle change. Security leaders should require explicit acceptance rules at each handoff and independent authorization where the resource is accessed.

The recommended approach is a boundary contract: a short, owned specification of the evidence a component accepts, the context it verifies, the authority it produces, and the events that end that authority. Contracts make integration assumptions visible and provide a direct basis for negative tests, incident response, and change review. Start with administrative applications, cross-tenant workflows, account linking, and high-value APIs, where an interpretation error can have consequential effects.

This white paper explains the failure mechanisms, applies them to a federated document service, and provides an implementation plan. Its guidance combines published standards with architectural analysis. The examples illustrate conditions to investigate; they are not reports of newly discovered vulnerabilities. The immediate leadership decision is to assign boundary ownership and fund verification of the full authentication path, including recovery, translated sessions, and downstream delegation.

## Decisions for security and platform leaders

Approve a common definition of authenticated context before selecting additional controls. An application identity should identify the trusted issuer, subject namespace, account mapping, and relevant assurance context. Resource services should receive the context needed for their own decisions. A generic flag called authenticated is insufficient when different consumers require different evidence. The standard should permit several credential formats while specifying the meaning each format must preserve.

Assign an accountable owner for every acceptance decision. Identity engineers own issuer relationships and issuance configuration. Application teams own account mapping and browser-session interpretation. API teams own credential profiles and gateway-to-service context. Resource owners define permissions and sensitive-operation requirements. Platform operators maintain the reliability of invalidation and configuration delivery. One security owner should coordinate the complete path so that a requirement cannot disappear between team boundaries.

Prioritize corrective work by the authority an error could expose. A wrong-audience credential accepted by a low-impact route and a credential accepted by an administrative API require different urgency. Examine independent containment before assigning impact. The objective is to close consequential gaps while maintaining legitimate operation, not to replace every component simply because its current contract is undocumented.

## The authentication handoff problem

Authentication systems answer questions about identity using evidence that other components can understand. A password verifier may establish knowledge of a secret, an authenticator may sign a challenge, and an identity provider may issue an assertion to an application. Applications then translate those results into sessions and decisions about resources. Each translation introduces an opportunity to interpret the evidence more broadly than its issuer intended. This white paper examines that interpretation problem: how can a system accept authentic evidence and still reach an incorrect security decision?

The question matters because architectural diagrams often compress authentication into a single arrow labeled login. That arrow hides exchanges with different purposes and adversarial conditions. A browser redirect carries a response through an environment partly controlled by the user and potentially influenced by other websites. A token endpoint exchanges a transient code for credentials. An application converts an external identifier into an internal account. A backend interprets a session or token as authority to act. None of these transitions automatically inherits every guarantee established at the previous transition.

## Standards and architectural foundation

OpenID Connect separates authentication information conveyed to a client from an access token used with a resource. Its validation rules include matching the expected issuer and checking the intended audience (OpenID Foundation, 2023, §3.1.3.7). The relevant lesson is that acceptance depends on context, not solely on whether a token can be parsed. A consumer needs an expectation established before processing the response.

JWT guidance likewise requires application-specific validation, including issuer and audience checks and mutually exclusive rules for different token uses (Sheffer et al., 2020, §§3.8-3.12). OAuth guidance addresses redirection, code injection, and replay, while issuer identification provides a mechanism for distinguishing authorization servers in a response (Lodderstedt et al., 2025; Meyer zu Selhausen & Fett, 2022). These specifications supply examples of boundaries that require more than transport encryption.

Fett et al. (2016) analyze OAuth authentication, authorization, and session integrity in an expressive web model. Their work identified attacks and proved properties for a corrected model under specified assumptions. Their subsequent OpenID Connect analysis formalizes related properties and implementation guidance (Fett et al., 2017). This white paper adopts the methodological lesson that the property being defended must be stated independently of the mechanism used to defend it.

WebAuthn binds authentication to relying-party and origin context through public-key credentials and ceremony validation (World Wide Web Consortium [W3C], 2021). That binding addresses a particular portion of the path. It does not itself define every application session transition or business-resource decision. The framework extends the analytical viewpoint to those surrounding transitions, where evidence is often translated into a representation with fewer contextual constraints.

## A boundary contract for authentication evidence

For this analysis, authentication evidence is a statement supported by a mechanism: a signed assertion, an authenticated response, or a server-maintained session record. Its meaning can be represented by six questions. Who produced it? Which subject does it describe? Which recipient may use it? Which transaction or ceremony produced it? During what interval is it relevant? What level and kind of assurance does it actually establish? A receiving component should have answers before promoting the evidence into application authority.

Let  $E$  denote received evidence,  $C$  the receiving context, and  $P$  the local policy. The acceptance model is  $\text{Accept}(E, C, P)$  only if provenance, integrity, intended use, transaction continuity, freshness, and local account binding all satisfy their relevant predicates. This expression is an organizational device, not a formal proof. The predicates depend on the selected protocol and deployment. An opaque server session and a JWT do not require identical parsing rules, but both require an account and lifecycle interpretation.

The distinction between authenticity and applicability is central. Authenticity concerns whether evidence originates from the claimed source without unauthorized alteration. Applicability concerns whether that source may make the relevant statement for this recipient and purpose. A valid signature from an approved partner can be authentic while remaining inapplicable to an administrative application. Conflating these questions allows a successful low-level check to conceal a missing higher-level decision.

Authority can also change during translation. A provider establishes an external identity, but an application decides which local record corresponds to it. A gateway verifies a token, but an internal service decides which operations the resulting request may perform. A session manager records a login, but a sensitive operation may require more recent or stronger evidence. The model therefore treats each promotion of evidence as a new decision with a new owner.

Boundary contracts make those decisions reviewable. Each contract names the producer, consumer, expected evidence type, trusted configuration, rejection conditions, and permitted output. It also identifies who can modify the configuration and how changes propagate. The contract must distinguish what is established from what remains unresolved. For example, a successful external login can establish an issuer-subject pair while leaving organization membership and document ownership to independent policy checks.

## Origin and issuer boundary failures

An origin boundary separates the party that presents a response from the party authorized to produce it. Transport security authenticates a connection endpoint, but it cannot by itself establish that an arbitrary endpoint is the intended identity authority. The consumer must obtain its expectation through a trusted configuration or another constrained bootstrap process. Otherwise, it risks selecting the authority using the same untrusted message that needs validation.

Consider an illustrative application supporting several partner organizations. It begins authentication with one provider but processes the callback using another provider selected from request data. Even

if both providers are legitimate and both communication channels are protected, the application can lose continuity between its original choice and the response it accepts. The relevant invariant is that authority selection survives the transaction. The scenario is analytical; it does not describe a newly discovered flaw in a named implementation.

Issuer confusion becomes especially difficult when identifiers are assumed to be globally unique. The string representing a subject may be unique only within an issuer's namespace. If an application discards the issuer and keeps only the subject, it can collapse distinct identities into one account. This is not a failure of the issuer to identify its own user. It is a failure of the consumer to retain the namespace in which the identifier has meaning.

The practical review question is therefore not merely whether a provider is trusted. It is whether the provider is trusted to assert this class of identity in this application, and whether that trust is fixed before processing the assertion. A partner authorized to authenticate its employees should not automatically determine internal administrators or tenants belonging to another partner. Configuration should express these distinctions directly rather than expecting developers to remember them in downstream code.

A defensible contract records the allowable issuer set and its relationship to tenant context. It also records how metadata and verification keys are obtained and refreshed. Key rotation is an operational necessity, but accepting newly discovered keys without preserving issuer ownership can expand the trust relationship unintentionally. Tests should therefore examine both ordinary rotation and disagreement between configuration, metadata, and presented evidence.

## **Recipient and token interpretation failures**

Recipient binding asks whether evidence was created for the component that receives it. The question applies even when an organization uses one identity provider for many applications. A provider's broad organizational authority does not erase distinctions between clients and resources. When a consumer treats every provider-issued token as interchangeable, it grants authority based on shared infrastructure rather than the evidence's intended use.

An illustrative failure occurs when an API accepts an authentication assertion intended for a frontend client as though it were an API credential. The signature can remain valid throughout. The mistake is the promotion of a statement about login into permission to invoke a resource. A separate resource policy might still deny the operation, but the authentication layer has already accepted evidence under the wrong contract. This explains why independent authorization remains valuable even with rigorous authentication.

RFC 9068 defines a JWT profile for OAuth access tokens, including validation expectations that distinguish those tokens from other JWT applications (Bertocci, 2021). The point for system design is not that every access token must use that format. It is that token purpose needs an explicit acceptance profile. A general-purpose verification function should not return an undifferentiated authenticated identity to consumers with different trust requirements.

The recommended design separates token interpretation into dedicated interfaces. An interface for validating an external login result returns an external identity and ceremony context. An interface for validating an API credential returns an applicable principal, recipient, and permitted credential context. These results should be represented by distinct types or data structures where possible. Such separation does not prove correctness, but it makes inappropriate substitution visible in code review and testing.

Recipient tests should include plausible but inappropriate evidence, not only malformed strings. Wrong-recipient tokens, tokens from adjacent clients, expired assertions, and correctly signed credentials of another purpose are stronger tests of the contract than random corruption alone. Each should fail at the boundary that owns the relevant decision. Logging should explain the failed category without exposing reusable credentials or allowing an external user to enumerate sensitive account details.

## **Transaction continuity and session integrity**

Authentication is a sequence, not an isolated response. The consumer must connect the initiating browser session, selected provider, returned response, and eventual application session. Without that continuity, an attacker may influence which identity is attached to a victim's interaction even if the individual identity is legitimate. The intended property is that the application session corresponds to the authentication transaction initiated for that user agent.

PKCE associates code redemption with a verifier from the initiating client transaction (Sakimura et al., 2015). Its scope should remain precise. Protecting code redemption does not automatically establish correct tenant mapping, resource permissions, or session cleanup. A review should identify the property supplied by each mechanism and inspect the remaining properties independently. This prevents the presence of one recognized control from ending the assessment prematurely.

The browser creates concurrency that diagrams frequently omit. Users can authenticate in several tabs, switch organizations, press back, or resume a suspended workflow. A design relying on a single mutable global variable for the expected transaction can associate one tab's response with another tab's context. The correct response is not to prohibit ordinary browser behavior reflexively, but to maintain transaction-specific state with explicit expiry and consumption rules.

Session integrity also involves post-login transitions. Linking an additional identity, elevating a session, changing a recovery method, and transferring ownership are distinct operations. They may require evidence about the currently authenticated account and the new identity being attached. Treating any successful login as sufficient for every account transition can create an unintended bridge between separate accounts. The boundary contract should identify which account must demonstrate control at each step.

An application can further confuse session identity with a user-supplied identifier. A route parameter or form field selecting an account does not inherit authenticity merely because the requester has a valid session. At the point where a session is associated with a target object, the system must establish the relationship independently. This is an authorization question adjacent to authentication, and preserving the distinction improves both error reporting and remediation ownership.

## Identity mapping and delegation boundaries

External authentication becomes useful only after the application maps it into its own account model. That mapping can depend on issuer-subject pairs, invited membership, organization identifiers, or administrative provisioning. An email address may be useful for communication while remaining unsuitable as the sole enduring identity key. The analytical concern is whether a mutable or reassigned attribute can silently redirect authority to a different local account.

Consider a hypothetical service that automatically joins users to an organization whenever their asserted email domain matches the organization's domain. The rule may be intentionally convenient, but it assumes that the identity source verifies the domain relationship and that the organization approves automatic admission. Those assumptions belong to different authorities. A trusted provider's ability to authenticate a user is insufficient to resolve the organization's membership policy unless that policy explicitly delegates the decision.

Account linking needs similarly explicit ownership. If the application merges records based only on a shared visible attribute, it can confuse correspondence with demonstrated control. A safer contract requires a defined basis for linking, records the identities participating in the change, and supplies a reversible administrative process for correcting mistakes. Whether that basis is dual authentication, managed provisioning, or verified invitation depends on the product and its risk model.

Delegation introduces another translation. RFC 8693 distinguishes delegation and impersonation and describes token exchange, while leaving deployment trust policy to implementations (Jones et al., 2020). The operating question is who retains responsibility for the original user's authority when a service acts downstream. A service credential can prove the calling workload's identity without proving that the initiating user may perform the requested action.

The contract therefore records both the immediate actor and the represented subject when the workflow needs them. A downstream service must know whether it is authorizing the workload itself, a user represented by that workload, or an explicitly approved combination. If an architecture discards that distinction, audits may attribute an operation to a user who never possessed the relevant permission or to a service too broad to reveal the initiating cause.

## Freshness and lifecycle boundaries

Evidence is valid within a lifecycle. An assertion may remain cryptographically verifiable after an account is disabled, a membership changes, or an administrator detects compromise. Whether it should continue to authorize a session depends on the system's documented semantics. A signature is evidence of issuance; it is not a live statement that every associated organizational fact remains unchanged.

OAuth token introspection offers a way for a protected resource to obtain information about a token, including whether it is active (Richer, 2015). Revocation provides a protocol for invalidating tokens, with implementation and propagation considerations (Lodderstedt et al., 2013). These mechanisms address lifecycle needs, but application sessions may have independent lifetimes. Revoking a credential at one layer does not justify assuming that all translated sessions have ended.

The important architectural question is where state has been copied. An identity provider can issue an assertion that an application exchanges for a local cookie. The application can then use that cookie to obtain service credentials or initiate asynchronous work. Each copy of authority may have a different expiry and invalidation path. Lifecycle review should follow those copies rather than stopping at the original credential's lifetime.

A lifecycle contract specifies what happens after logout, account disablement, credential reset, policy change, and suspected key compromise. The requirements can differ by event. Ordinary logout may terminate one browser session; compromise response may require broader invalidation. Conflating them encourages either excessive disruption or insufficient containment. The contract should state the scope of each event and identify which components must acknowledge it.

Freshness also requires a failure policy. If current status cannot be obtained, a consumer must decide whether cached information remains acceptable. That decision should depend on the operation and the age of the evidence, not on an undocumented fallback. Low-risk reads may tolerate a bounded interval; administrative changes may require current verification. The framework treats this as a product and security decision that must be observable and testable.

## **Design choices and supported exceptions**

The analysis supports organizing authentication review around boundary ownership. Teams should identify who owns issuer configuration, token validation profiles, account mapping, session invalidation, and downstream identity propagation. When ownership is diffuse, each component can assume another has already enforced the missing property. A contract assigns that property to a specific decision point and makes the assumption available to reviewers.

Typed evidence can improve this arrangement. If a library returns a generic dictionary after signature verification, developers may use its contents before establishing applicability. A narrower result type can require successful profile validation before producing an authenticated principal. This proposal should not be interpreted as a claim that static typing prevents all security defects. It reduces one class of accidental promotion while leaving configuration and runtime state to additional controls.

Defense in depth remains important because no contract eliminates endpoint compromise or operational error. DPoP binds use of certain OAuth tokens to a proof key and request context (Fett et al., 2023). It can reduce replay opportunities under its assumptions, but cannot establish the correctness of an application's account mapping. The broader design lesson is to select protections for specific failure mechanisms and avoid treating any single mechanism as complete system assurance.

Several alternative explanations deserve attention. A failure that resembles issuer confusion may actually originate in an incorrect tenant provisioning rule. A session surviving account disablement may reflect an intentional offline-access policy rather than an implementation defect. A token accepted by several services may represent a deliberately shared audience. An assessment must compare behavior with explicit requirements before assigning a vulnerability category.

The framework is most useful when requirements are initially incomplete. It forces a team to state whether those behaviors are intended and who approved them. Once requirements are explicit,

ordinary engineering techniques can test them. The framework itself does not replace protocol conformance, risk assessment, or application-specific authorization design. Its contribution is a way to connect those activities through the evidence that passes between them.

## **Worked example of a federated document service**

Consider a hypothetical document service used by two organizations. Each organization authenticates through an approved provider, and the frontend establishes a local browser session. An API gateway accepts access credentials, while a document service applies ownership and sharing policy. A background worker produces exports requested by users. This architecture contains several distinct promotions of evidence even though a product diagram might describe it as a single sign-on system. The purpose of the example is to demonstrate the contract analysis without claiming an observed vulnerability.

The first contract concerns provider selection. Before redirecting the browser, the frontend records the intended organization, provider, and transaction. The returned response must remain associated with that selection. A provider authenticated for organization A cannot determine the organization merely by supplying an attribute that says B. The frontend's output is an external identity with a verified namespace and transaction context. It is not yet a statement that the identity belongs to every organization represented in the product.

The second contract maps the external identity into an application account. An invitation or provisioning rule establishes organization membership. The mapping must not silently merge two accounts because they share a mutable display attribute. If a legitimate user changes a communication address, the service should preserve the identity relationship according to its account model. If an address is reassigned, the service should have a defined process for preventing the reassignment from transferring access to historical documents unintentionally.

The third contract applies at the gateway. Its credential acceptance profile identifies the expected issuer, resource audience, permitted token purpose, and validity conditions. A frontend login result cannot become an API credential simply because the gateway recognizes its signature. After acceptance, the gateway supplies authenticated context to the document service through a protected and clearly owned channel. Client-provided identity headers must not be treated as equivalent to gateway-established identity context. The service's contract must explain how it distinguishes those sources.

The fourth contract belongs to the document service. It checks whether the authenticated principal may perform the requested action on the selected document in the relevant organization. The principal's presence in one organization does not establish ownership of a document in another. An approved sharing relationship can permit access, but that relationship must be evaluated as policy. This step demonstrates why a correct authentication path cannot eliminate the need for object-level authorization or why an authorization failure should not automatically be blamed on the identity provider.

The export worker introduces a temporal boundary. The service might approve an export at submission time, or require authorization again when the worker begins and when the result is

downloaded. Either policy can be described coherently, but the choice matters after membership removal. The architecture must identify whose authority the worker exercises, how the initiating subject is retained, and who may retrieve the result. A broad worker credential should not erase the approved scope of the individual export request.

An assessment would create controlled documents for both organizations and test each contract independently. It would interleave two login transactions, substitute adjacent credential purposes, change a mutable account attribute, remove membership before export execution, and attempt retrieval under a different session. Expected decisions would follow the documented policy rather than assumptions about what most services do. The evidence would identify the rejection point and resulting resource state. These tests would evaluate assurance continuity without requiring malicious content or access to unrelated users' data.

The worked example reveals a useful diagnostic distinction. If the gateway accepts the wrong credential but the document service denies access, the authentication contract has failed while resource containment has held. If the document service permits an unshared cross-organization read with a correctly accepted credential, the authorization contract has failed. If an export continues after membership removal under a documented submission-time approval policy, the observation alone is insufficient to establish a defect. Precise contracts make these outcomes interpretable and direct remediation to the component responsible for the violated requirement.

The service also needs a contract for configuration changes. An administrator who adds a provider or changes account-mapping rules can alter the meaning of future authentication evidence. Review should therefore identify the administrative authority behind those changes and determine whether existing sessions retain their earlier interpretation. Configuration history can explain why identical-looking evidence receives different treatment at different times. Without that history, an assessor may misclassify a deliberate migration as inconsistent enforcement or overlook an unauthorized expansion of the issuer's authority.

A practical acceptance record would include the contract version used for the decision and a non-secret correlation identifier connecting the browser transaction to the resulting session. The record should retain enough context to diagnose disagreement without storing the full reusable credential. During an incident, investigators could determine whether the system accepted the wrong issuer, mapped the identity incorrectly, or applied an outdated resource rule. This observability proposal supports investigation; it does not establish that logging prevents the failure itself.

The contracts should also be reviewed after product changes. Adding a mobile client, background worker, or alternate login path can introduce a new evidence transformation. Reusing an existing validation helper is appropriate only if its contract matches the new consumer's purpose. The architecture owner can assess that match by comparing accepted evidence, expected context, and permitted output. This keeps the worked analysis useful as the service evolves rather than preserving a correct description of a superseded authentication path.

## Minimum acceptance requirements

Each contract should be brief enough to maintain and detailed enough to test. Record the producer, consumer, credential purpose, trusted configuration, subject mapping, transaction linkage, validity conditions, and permitted output. State which additional checks belong to the next consumer.

Document the supported exceptions, including deliberately shared audiences or offline session behavior. Exceptions need an owner and a reason; they should not be inferred from the fact that a request currently succeeds.

Implementation should separate integrity verification from profile acceptance. The component should expose an authenticated principal only after all required applicability checks pass. Where practical, use distinct interfaces for login assertions, API credentials, and delegated service context. A shared validation library can support these interfaces, but it should not make every token interchangeable merely because verification uses the same key source.

Operational records should identify the contract version and decision outcome without retaining reusable credentials. Record enough context to determine whether rejection involved issuer, recipient, purpose, account mapping, freshness, or resource policy. Protect these records because they describe valuable access relationships. Logging supports diagnosis and accountability; enforcement must remain at the actual decision point.

## A 90 day implementation plan

Days 1 through 30 should establish scope and a reliable baseline. Select a small set of sensitive workflows and map the evidence from initiation to resource access. Include alternate login routes, account linking, recovery, background jobs, and administrative configuration. Identify the owners and current acceptance rules. Capture representative successful operations and expected denials using controlled accounts and resources. Report unknown relationships as gaps requiring investigation, rather than treating an incomplete map as proof of safety.

Days 31 through 60 should address the highest-impact interpretation gaps. Introduce or tighten credential profiles, preserve issuer-subject namespaces, constrain provider selection, and review local account linking. Confirm that gateway-established identity cannot be replaced with client-supplied context. Add tests using valid but inappropriate credentials, because random invalid input does not establish recipient or purpose isolation. Deploy changes in stages and verify that supported clients and concurrent browser interactions continue to work.

Days 61 through 90 should verify lifecycle behavior and institutionalize review. Exercise account disablement, membership removal, logout, credential reset, and key rotation. Follow authority into local sessions and downstream credentials. Measure the interval until each relevant consumer denies subsequent operations under its documented requirements. Review exceptions that permit continued operation. Add contract review to client onboarding, API introduction, federation changes, and identity migration. Close the rollout only when both the new enforcement and the required business workflows are verified.

The schedule is a planning template, not a promised completion time. A large estate may need several waves. Each wave should finish a coherent workflow rather than documenting many applications superficially. Dependencies on partner providers, application redesign, or legacy sessions may require separate milestones. Keep those dependencies visible in the delivery plan and define interim containment for high-impact gaps.

## **Verification that establishes readiness**

Testing should compare actual decisions with requirements written independently of the implementation. Create adjacent clients and tenants with controlled identities. Confirm ordinary successful login and legitimate resource access before introducing substitutions. Then vary the issuer namespace, credential recipient, token purpose, transaction association, or target organization while keeping the other conditions valid. Every case should identify the component responsible for rejection and the protected outcome.

Lifecycle verification should include multiple consumers and previously issued authority. Disable a test account and observe the browser application, API, worker, and download path. A central denial does not establish that a translated session has ended. Conversely, continued background execution may be intentional under a documented submission-time approval rule. The test must use the policy for that workflow rather than a universal assumption that every activity ends simultaneously.

Readiness requires evidence for both protection and functionality. The required workflow should succeed, inappropriate context should be rejected, configuration ownership should be recorded, and lifecycle delays should remain within approved bounds. Any untested path should be listed explicitly. A successful review of one gateway route should not be reported as assurance for alternate application routes or administrative interfaces that were outside the test scope.

## **Measures and reporting**

Use a small set of measures tied to decisions. Track the proportion of in-scope workflows with an owned contract, the proportion with tested negative cases, and the proportion with verified withdrawal behavior. Report accepted inappropriate-context requests by consequence and responsible boundary. Measure withdrawal delays separately for application sessions and downstream credentials. Include legitimate-flow failures so that the program cannot appear successful simply by denying every request.

Coverage denominators matter. Ten tested workflows out of twelve in scope provide a different assurance claim from ten workflows selected from an unknown estate. Establish the inventory before presenting a percentage. Record the period and contract versions covered by the tests. An architectural change can invalidate earlier evidence, so readiness should be renewed when the acceptance path or trust relationship changes materially.

Leadership reporting should lead with unresolved consequential gaps and the next decision needed to close them. Avoid presenting a long count of validated signatures as evidence of complete assurance. Summarize the authority affected, independent containment, accountable owner, planned

correction, and verification status. Technical appendices can retain the request traces and detailed cases needed by engineers.

## **Implementation tradeoffs and exceptions**

Strict context validation can expose clients that relied on broad or inconsistent acceptance. Treat compatibility as an engineering requirement, not a reason to remove the boundary silently. Identify the intended credential contract for each client, plan migration, and monitor legitimate failures during rollout. Where a temporary exception is needed, document its scope and termination condition and verify the compensating resource restrictions.

Freshness requirements should reflect the operation. An administrative change may require current assurance while a low-impact read tolerates a bounded session age. Universal reauthentication can impose unnecessary friction, while indefinite acceptance can retain inappropriate authority. Resource owners should approve the relevant interval and explain the failure behavior when current status is unavailable. The contract turns that tradeoff into a reviewable decision.

## **Action checklist**

- Name the owner of every sensitive authentication handoff.
- Separate signature verification from issuer, recipient, purpose, and transaction acceptance.
- Preserve identity namespaces and use an explicit account-linking rule.
- Apply independent resource and tenant authorization.
- Follow lifecycle changes into local sessions, workers, and delegated credentials.
- Test valid evidence in the wrong context and verify legitimate workflows after correction.
- Review contracts whenever a new consumer or identity relationship is introduced.

## **Recommended next step**

Begin with one administrative workflow and one cross-tenant API. Complete their contracts, controlled tests, and withdrawal exercise before expanding the program. Use the results to refine the common acceptance interfaces and identify shared libraries that need correction. This creates a concrete operating standard grounded in the actual estate. Scale through successive workflows while retaining the ownership, evidence, and exception discipline that made the first wave reviewable.

## 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>
- Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., & Waite, D. (2023). OAuth 2.0 demonstrating proof of possession (DPoP) (RFC 9449). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc9449.html>
- 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., Dronia, S., & Scurtescu, M. (2013). OAuth 2.0 token revocation (RFC 7009). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc7009.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>
- OpenID Foundation. (2023). OpenID Connect Core 1.0 incorporating errata set 2. [https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html)
- Richer, J. (2015). OAuth 2.0 token introspection (RFC 7662). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc7662.html>
- Sakimura, N., Bradley, J., & Agarwal, N. (2015). Proof key for code exchange by OAuth public clients (RFC 7636). Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc7636.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>
- World Wide Web Consortium. (2021). Web authentication: An API for accessing public key credentials - Level 2. <https://www.w3.org/TR/webauthn-2/>