WG Working Group
Internet-Draft
Intended status: Informational
Expires: 25 July 2026

A. RAUT
Amazon

21 January 2026

OAuth Transaction Tokens Best Current Practice
draft-oauth-transactiontokens-bcp-01

Abstract

This document provides best current practices for implementing and deploying OAuth 2.0 Transaction Tokens as specified in draft-ietf-oauth-transaction-tokens. Transaction Tokens (Txn-Tokens) enable workloads in a trusted domain to preserve and propagate user identity and authorization context across service boundaries during the processing of external programmatic requests. This BCP addresses practical deployment considerations including token service architecture, size management, propagation patterns, validation strategies, and operational monitoring that are essential for secure and effective implementation in production environments.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-oauth-transactiontokens-bcp/.

Source for this draft and an issue tracker can be found at https://github.com/ashayraut/oauth-transactiontokens-best-current-practice.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

1.1. Background

Modern distributed systems built on microservice architectures face a fundamental challenge: maintaining security context as requests traverse multiple service boundaries. When an external actor initiates an API request, the user identity and authorization context must be preserved and made available to all downstream internal microservices involved in processing that request. Without a standardized mechanism, organizations resort to ad-hoc solutions that introduce security vulnerabilities, operational complexity, and interoperability challenges.

The OAuth 2.0 Transaction Tokens specification (draft-ietf-oauth-transaction-tokens) addresses this challenge by defining a token format and exchange protocol that enables secure context propagation across internal microservices within trusted domains. However, the specification focuses on protocol mechanics rather than deployment practices. Real-world implementations face additional challenges including latency constraints, token size limitations, schema evolution, propagation reliability, and operational monitoring.

1.2. Purpose of the BCP

This Best Current Practice document provides implementers with guidance derived from production deployments of Txn-Token systems. It addresses practical considerations that fall outside the scope of the protocol specification but are critical for successful deployment. The recommendations in this document are based on operational experience with large-scale microservice environments where hundreds of internal microservices must coordinate security context propagation across complex call chains.

This BCP is intended for: - Organizations implementing Transaction Token Services - internal microservice developers integrating Txn-Token support - Security architects designing authorization systems - Operations teams monitoring token propagation

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. This document uses terminology from draft-ietf-oauth-transaction-tokens including "Transaction Token" (Txn-Token), "Transaction Token Service", "trusted domain", and "authorization context".

3. Best Current Practices

3.1. Transaction Token Service Implementation

3.1.1. Service Architecture

Organizations SHOULD setup Transaction Token Service (TTS) which hosts functionality to issue token, replace token and other Txn-Token related functionality. Organizations SHOULD prefer architectures where the authorization service that authenticates and authorizes external actors, invokes the TTS for getting Transaction Token as part of authentication and authorization requests and pass Txn-token as well with it. This way, it avoids an explicit calls from external endpoint to TTS and lesser code changes in external services. Additionally, all internal microservices that want to replace tokens SHOULD connect directly to TTS. This architecture strikes balance between the options to either have authorization service host all TTS functionality or external services needing to connect TTS.

3.1.2. Context Selection

Transaction Token Services MUST include all mandatory claims defined in draft-ietf-oauth-transaction-tokens. However, services SHOULD NOT include all optional contexts by default. Optional contexts such as transaction context (tctx) or custom claims MUST be added only when explicitly requested.

Organizations SHOULD provide client libraries that offer interfaces for requesting Txn-Tokens with specific optional contexts. This approach prevents token bloat while ensuring that services can obtain the context they require. When an optional context cannot be added due to parsing errors, format violations, or unavailability, the Transaction Token Service MUST NOT fail the token issuance. Instead, it SHOULD issue the token without that specific context and MAY log the condition for operational monitoring.

3.2. Token Size Management

3.2.1. Size Limits

Transaction Token Services MUST NOT issue tokens larger than 4KB. While HTTP specifications do not mandate maximum header sizes, common web server implementations impose limits to prevent Denial of Service attacks. Apache defaults to 8KB maximum header size, but organizations must account for other headers in the same request. A 4KB limit for Txn-Tokens provides reasonable headroom while preventing operational issues.

Organizations SHOULD implement monitoring on token size to detect trends toward the limit. Services that consistently approach size limits indicate either excessive context inclusion or the need for context relocation strategies.

3.2.2. Context Relocation

When authorization context exceeds 4KB, Transaction Token Services SHOULD implement a relocation endpoint. The service stores oversized contexts in a separate data store using the Txn-Token identifier as the primary key. The Txn-Token itself contains only a reference to the relocated context.

Client libraries for token validation SHOULD transparently handle context relocation. When an internal microservice requests a context that has been relocated, the library fetches it from the relocation endpoint. This pattern mirrors Policy Information Points in Attribute-Based Access Control (ABAC) systems where additional attributes are retrieved at runtime.

Context relocation introduces additional latency and failure modes. Organizations SHOULD treat relocation as an exception rather than the normal case. Monitoring SHOULD track relocation frequency to identify services that consistently require excessive context.

3.3. Token Lifetime and Expiration

Transaction tokens SHOULD have a time-to-live of less than 5 minutes. Organizations SHOULD determine appropriate lifetimes by working backward from latency Service Level Agreements (SLAs) defined for external endpoints.

Short token lifetimes reduce the window for token compromise and limit the impact of token mix-up scenarios. However, lifetimes must accommodate the longest expected call chains in the SOA. Organizations SHOULD measure actual request processing times and set token lifetimes to exceed the 99th percentile by a reasonable margin.

When tokens expire during request processing, services MUST NOT automatically request new tokens. Expired tokens indicate either excessively long call chains or performance problems that require investigation. Services SHOULD fail requests with expired tokens and emit telemetry for operational monitoring.

3.4. Schema Governance

3.4.1. Context Visibility

Organizations MUST govern the contexts added to Txn-Tokens. Once a context appears in a Txn-Token, it becomes visible to all services in the call chain. Services may develop dependencies on these contexts in ways not anticipated by the context provider. This phenomenon follows Hyrum's Law: with sufficient consumers, all observable behaviors of a system will be depended upon by somebody.

Before adding a new context to Txn-Tokens, organizations MUST consider the implications of making that context universally visible. If a context represents an identifier or concept known only to services early in the call chain, adding it to the Txn-Token exposes it to all downstream services. Those services may develop business logic dependencies on the context, not just security dependencies.

When the format or semantics of a widely-visible context must change, the organization faces a painful migration process. All services that depend on the context must be identified, updated, and deployed. Organizations SHOULD prefer adding new contexts with different names rather than changing existing contexts when semantic changes are required.

3.4.2. Backward Compatibility

Organizations MUST implement backward compatibility tests for Txn-Token contexts. Automated tests SHOULD verify that changes to context format or structure do not break existing consumers. These tests SHOULD run as part of the continuous integration pipeline for the Transaction Token Service.

Backward compatibility testing becomes increasingly important as the number of services consuming Txn-Tokens grows. Without automated verification, format changes risk cascading failures across the SOA.

3.5. Token Propagation

3.5.1. Propagation Control

Organizations MUST prevent Txn-Tokens from propagating outside the trusted domain. While tokens contain encrypted sensitive data, organizations SHOULD implement explicit controls to block external propagation. Propagation libraries MUST detect when an internal microservice attempts to include a Txn-Token in a request to an external endpoint and MUST remove the token from that request.

This defense-in-depth approach protects against misconfiguration and implementation errors. Even if token encryption remains secure, preventing external propagation eliminates entire classes of potential vulnerabilities.

3.5.2. Propagation Libraries

Organizations SHOULD provide standardized propagation libraries that handle token lifecycle within an internal microservice workload processing. These libraries MUST extract the Txn-Token from the incoming HTTP header, store it in request-scoped memory, add the token to outgoing request headers, and clear it from memory when request processing completes.

Standardized libraries provide several benefits. First, they enforce propagation controls including external blocking to avoid the token flowing outside your trust boundary. Second, they can be used to consistently emit telemetry about token initiation, propagation, and validation. Third, they provide a centralized point for implementing fallback behaviors when tokens are missing.

3.5.3. Propagation Reliability

Organizations SHOULD monitor propagation success rates across the SOA. Unless propagation success reaches 100% for a given call chain, services cannot reliably enforce authorization policies based on Txn-Token contents. Services MUST implement reasonable fallback behaviors when tokens are absent.

Propagation libraries MAY implement automatic token initiation when an incoming request lacks a Txn-Token. The library requests a placeholder token from the Transaction Token Service indicating that no context was received at the current service. This placeholder enables downstream services to identify where propagation broke in the call chain, facilitating operational debugging.

3.5.4. Token Mix-up Prevention

Token mix-up represents the most severe propagation risk. Mix-up occurs when a token intended for one request is incorrectly attached to a different request. If token T1 is meant for request R1 and token T2 for request R2, but T2 is sent with R1 due to a propagation bug, actors may access data they are not authorized to see.

Token mix-up scenarios are difficult to detect because they may not cause obvious failures. The request succeeds but with incorrect authorization context. Organizations MUST implement request-scoped token storage in propagation libraries to prevent mix-up. Tokens MUST be associated with specific request contexts and MUST NOT be stored in shared or global state.

Organizations SHOULD implement testing strategies that deliberately attempt to cause token mix-up under concurrent load. These tests verify that propagation libraries correctly isolate tokens across concurrent requests.

3.5.5. Trust Boundary Handling

When requests cross trust boundaries within the organization, propagation libraries MUST either block token propagation or replace token contents with appropriately scoped contexts. Organizations SHOULD define trust boundaries explicitly and configure propagation libraries with boundary detection logic.

At trust boundaries, services MAY request new Txn-Tokens from the Transaction Token Service with contexts appropriate for the target trust domain. This approach maintains context propagation while respecting security boundaries.

3.5.6. Cache Considerations

The introduction of Txn-token provides more information now to the entire microservice architecture graph. There are Services in the graph that cache data to avoid calling dependent services multiple times. Now, they SHOULD consider Txn-Token contexts to be included in the cache keys. If not included, there is a risk that incorrect data is vended out or cache hit is impacted because the dependent services might be using the Txn-Token contexts for computing the results which might get cached.

Organizations SHOULD provide guidance to workload developers on cache key construction when Txn-Tokens are involved. Cache invalidation strategies MUST account for context changes that affect cached data.

3.6. Token Validation

3.6.1. Validation Libraries

Organizations SHOULD provide standardized validation libraries that handle signature verification, decryption, and token parsing. These libraries SHOULD synchronize cryptographic keys in the background, ensuring that services always have current keys for verification and decryption.

Validation libraries SHOULD decode Txn-Tokens into strongly-typed objects appropriate for the implementation language. This approach prevents parsing errors and provides compile-time verification of context access patterns.

3.6.2. Error Handling

Validation libraries SHOULD emit standardized error codes for common failure conditions including expired tokens, malformed tokens, and signature verification failures. These error codes enable consistent operational monitoring across the SOA.

3.6.3. Fallback Policies

Services SHOULD NOT automatically fail requests when Txn-Tokens are missing or invalid. Organizations MUST define fallback policies that balance security with user experience. Fallback policies MAY include serving redacted data, limiting functionality, or requesting step-up authentication.

The appropriate fallback depends on the sensitivity of the requested operation. Services accessing highly sensitive data MAY require valid Txn-Tokens and fail requests when tokens are absent. Services providing less sensitive functionality SHOULD implement graceful degradation.

3.7. Telemetry and Monitoring

3.7.1. Adoption Monitoring

Transaction token adoption in large SOA environments takes time. Organizations SHOULD implement comprehensive telemetry to monitor adoption progress, propagation reliability, and validation patterns.

When organizations provide standardized libraries for token initiation, propagation, and validation, telemetry logic SHOULD be embedded in those libraries. This approach ensures consistent telemetry across all services without requiring individual workload implementations.

3.7.2. Telemetry Aggregation

Services SHOULD aggregate telemetry locally before transmitting to centralized monitoring systems. Local aggregation reduces network overhead and enables higher-frequency sampling without overwhelming monitoring infrastructure.

Centralized monitoring systems SHOULD store telemetry in data warehouses that support analytical queries. Organizations SHOULD implement automated monitors that alert on significant changes in propagation rates, validation failures, or token expiration rates.

3.7.3. Key Metrics

Organizations SHOULD monitor the following key metrics: - Token initiation rate by service - Propagation success rate by call chain - Token expiration rate during request processing - Validation failure rate by error type - Token size distribution - Context relocation frequency

These metrics provide visibility into Txn-Token health across the SOA and enable rapid identification of deployment issues.

3.8. Key Management

Organizations MUST implement secure key management practices for Txn-Token cryptographic operations. Key management SHOULD follow the guidelines in RFC 4107 "Guidelines for Cryptographic Key Management".

Transaction Token Services MUST support key rotation without service disruption. Validation libraries MUST support multiple concurrent keys to enable zero-downtime rotation. Organizations SHOULD automate key rotation on a regular schedule.

3.9. Batch Processing pattern

OAuth Transaction Tokens are designed to propagate security context through a call chain within a trust domain. To maintain a high security posture without the overhead of a global revocation infrastructure, these tokens are short-lived (typically minutes). In many modern architectures, a transaction may be asynchronous. For example, a request may be placed on a message queue (e.g., Kafka, RabbitMQ) and processed by a worker service hours or days later. By the time the worker resumes the transaction, the original Transaction Token has expired.

Batch Token (Voucher): A long-lived, opaque, or encrypted token representing the transaction context during a period of rest. Initiator: The internal microservice that receives a Transaction Token and requests a Batch Token before an asynchronous pause. Rehydrator: The internal microservice that takes a Batch Token and exchanges it for a fresh, short-lived Transaction Token to resume processing.

3.9.1. Initiation (Pausing the Transaction)

When a internal microservice determines that a transaction will exceed the TTL of the current Transaction Token (TraT), it SHOULD request a Batch Token from the Transaction Token Service (TTS). The request to the TTS SHOULD include: * The current valid TraT. * The intended "use case ID" or "namespace" to constrain the token.

The TTS returns a Batch Token with a TTL suitable for the asynchronous delay (e.g., 24 hours to 7 days).

3.9.2. Rehydration (Resuming the Transaction)

When a worker service (the Rehydrator) picks up the task, it MUST NOT use the Batch Token directly to call downstream services. Instead, it MUST exchange the Batch Token at the TTS for a fresh TraT. The TTS SHALL: 1. Verify the Batch Token’s signature and expiration. 2. Validate that the Rehydrator is authorized for the specific "use case ID" or "namespace" embedded in the Batch Token. 3. Issue a new, short-lived TraT containing the original claims (e.g., subject, original requester IP).

4. Security Considerations

4.1. Token Mix-up Prevention

Token mix-up represents a critical security risk. Organizations MUST implement request-scoped token storage and MUST test for mix-up scenarios under concurrent load. Token mix-up can result in unauthorized data access without obvious system failures, making it particularly dangerous.

4.2. Trust Boundary Controls

Organizations MUST define trust boundaries explicitly and MUST implement controls that prevent inappropriate token propagation across those boundaries. Failure to control propagation at trust boundaries can expose sensitive contexts to unauthorized services.

4.3. Cache Security

Services that cache data based on Txn-Token contexts face security risks if cache keys do not incorporate all relevant contexts. Organizations MUST provide guidance on secure cache key construction and MUST audit caching services for correct context handling.

4.4. Fallback Mechanisms

While fallback mechanisms improve availability, they can introduce security vulnerabilities if not carefully designed. Organizations MUST ensure that fallback policies do not inadvertently grant excessive access when Txn-Tokens are absent. Fallback policies SHOULD be explicitly documented and reviewed by security teams.

4.5. Token Lifetime

Short token lifetimes reduce the window for token compromise but may cause operational issues if set too aggressively. Organizations MUST balance security considerations with operational requirements when setting token lifetimes.

4.6. External Propagation

Preventing Txn-Tokens from leaving the trusted domain is critical. Organizations MUST implement multiple layers of defense including library-level controls, network-level filtering, and monitoring for external propagation attempts.

4.7. Batch processing security consideration

4.7.1. Token Constraining

Batch Tokens MUST be sender-constrained or scoped to specific namespaces. This prevents a compromised service from "stealing" a Batch Token from a queue and successfully minting a Transaction Token for an unrelated flow.

4.7.2. Data Mutability and Consent

Asynchronous delays increase the risk that the underlying authorization context has changed (e.g., a user has revoked consent). The TTS SHOULD perform a "freshness check" during rehydration for claims marked as mutable or sensitive.

4.7.3. Infinite Exchange Prevention

To prevent a transaction from living indefinitely through repeated rehydrations, the TTS SHOULD implement a maximum chain depth or total transaction lifetime counter within the token metadata.

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

RFC2119 (https://datatracker.ietf.org/doc/html/rfc2119) Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

TXNTOKENS (https://www.ietf.org/archive/id/draft-ietf-oauth-transaction-tokens-06.html) Tulshibagwale, A., Hardt, D., and G. Fletcher, "OAuth 2.0 Transaction Tokens", draft-ietf-oauth-transaction-tokens-06 (work in progress).

6.2. Informative References

RFC4107 (https://datatracker.ietf.org/doc/html/rfc4107) Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005.

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Acknowledgments

TODO acknowledge.

Author’s Address

ASHAY RAUT
Amazon

Email: asharaut@amazon.com