Network Working Group
Internet-Draft
Intended status: Informational
Expires: 23 November 2026

A. RAUT

22 May 2026

Transaction Tokens For Agents
draft-araut-oauth-transaction-tokens-for-agents-02

Abstract

This document specifies an extension to the to support agent context propagation within Transaction Tokens for agent-based workloads. The extension defines the use of the act field to identify the agent performing the action, and leverages the existing sub field (as defined in the base Transaction Tokens specification) to represent the principal. The sub field is populated according to the rules specified in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html), based on the 'subject_token' provided in the token request. For autonomous agents operating independently, the sub field represents the agent itself. These mechanisms enable services within the call graph to make more granular access control decisions, thereby enhancing security. This document specifies an extension to OAuth Transaction Tokens OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html) to support agent context propagation within Transaction Tokens for agent-based workloads. The extension defines the agentic_ctx claim, a structured JWT claim that carries chain-level metadata required for multi-agent flow integrity and MAY contain additional deployment-specific agent context. The extension addresses two deployment patterns: single-agent flows where one agent acts on behalf of a user within a transaction, and multi-agent flows where multiple agents collaborate across a call chain. In multi-agent scenarios, the Transaction Token is replaced at each agent transition by the Transaction Token Service, updating the agentic_ctx claim while preserving the immutable identity context (sub and act claims) established at transaction initiation. This specification leverages the existing act claim as defined in RFC8693 (https://datatracker.ietf.org/doc/html/rfc8693) and the sub claim as defined in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html). It does not define new token types or grant types; it operates entirely within the existing Transaction Token issuance and replacement mechanisms defined by the base specification.

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://ashayraut.github.io/oauth-transactiontokens-for-agents/draft-oauth-transaction-tokens-for-agents.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-araut-oauth-transaction-tokens-for-agents/.

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

Status of This Memo

This Internet-Draft will expire on 23 November 2026.

Copyright Notice

Table of Contents

1. Introduction

Traditional zero trust authorization systems face new challenges when applied to AI agent workloads. Unlike conventional web services, AI agents possess capabilities for autonomous operation, behavioral adaptation, and dynamic integration with various data sources. These characteristics may lead to decisions that extend beyond their initial operational boundaries.

Transaction Tokens (Txn-Tokens) are short-lived, signed JSON Web Tokens [RFC7519] that convey identity and authorization context. However, the current Txn-Token format lacks sufficient context for services within the call chain to implement fine-grained access control policies for agent-based workflows. Specifically, it does not provide adequate information about the AI agent's chain-level context, limiting transaction traceability and authorization granularity in multi-agent deployments.

This document defines the agentic_ctx claim, a structured extension within a Transaction Token that carries chain-level metadata for multi-agent flow integrity. It also specifies how the existing act claim (defined in [RFC8693]) and sub claim (defined in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html)) are used in agent-based transaction flows.

The extension introduces a two-layer model for agent transaction context:

* Identity Layer: The sub claim (representing the subject of the transaction) and the act claim (representing the agent that obtained the original access token) are immutable throughout the transaction lifetime. These claims are populated per the rules defined in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html) and RFC8693 (https://datatracker.ietf.org/doc/html/rfc8693) respectively.

* Context Layer: The agentic_ctx claim carries chain-level metadata required for multi-agent flow integrity (hop tracking, current and originating agent identifiers) and MAY contain additional deployment-specific agent context. The internal structure beyond the normative fields defined in this specification is not prescribed; deployments MAY include additional context such as operational posture, provenance, assurance levels, or identity information as required by their trust domain policies.

To support richer context population, the TTS SHOULD have access to an Agent Registry or equivalent mechanism that enables it to distinguish agent callers from non-agent workload callers and to source agent metadata within the trust domain. When the TTS identifies a caller as an agent, it applies agent-specific token issuance rules as defined in this specification. When the caller is not identified as an agent, the TTS follows the base specification rules without modification. Deployments without an Agent Registry MAY still populate agentic_ctx using information derivable from the token exchange flow itself (e.g., current_actor from client_id, hop_count from replacement count).

This extension leverages the existing Txn-Token infrastructure to enable secure propagation of AI agent context throughout the service graph.

1.1. Conventions and Terminology

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 (https://datatracker.ietf.org/doc/html/rfc2119) RFC8174 (https://datatracker.ietf.org/doc/html/rfc8174) when, and only when, they appear in all capitals, as shown here.

2. Terminology

Agentic-AI: AI Agentic applications are software applications that utilize Large Language Models (LLM)s and plans, reasons,and takes actions independently to achieve complex, multi-step goals with minimal human oversight.

Workload: An independent computational unit that can autonomously receive and process invocations, and can generate invocations of other workloads. Examples of workloads include containerized microservices, monolithic services and infrastructure services such as managed databases.

Trust Domain: A collection of systems, applications, or workloads that share a common security policy. In practice this may include a virtually or physically separated network, which contains two or more workloads. The workloads within a Trust Domain may be invoked only through published interfaces.

Call Chain: A sequence of synchronous invocations that results from the invocation of an external endpoint.

External Endpoint: A published interface to a Trust Domain that results in the invocation of a workload within the Trust Domain. This is the first service in the call chain where request starts.

Transaction Token (Txn-Token): A signed JWT with a short lifetime, providing immutable information about the user or workload, certain parameters of the call, and specific contextual attributes of the call. The Txn-Token is used to authorize subsequent calls in the call chain.

Transaction Token Service (Txn-Token Service): A special service within the Trust Domain that issues Txn-Tokens to requesting workloads. Each Trust Domain using Txn-Tokens MUST have exactly one logical Txn-Token Service.

Agent Registry: A service or data store accessible to the Transaction Token Service that maintains a registry of known agent identities. The TTS MAY use the Agent Registry to determine whether a requesting workload is an agent and to source metadata for agentic_ctx population. The presence of an Agent Registry is RECOMMENDED but not required.

3. Protocol overview

3.1. Transaction Flow

This section describes the process by which an agent application obtains a Transaction Token, either acting autonomously or on behalf of a principal. The external endpoint requests a Txn-Token following the procedures defined in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html), augmented with additional context for agent identity and, when applicable, principal identity.

3.2. Agent Application Transaction Flows

The Transaction Token creation process varies depending on the presence of a principal.

3.2.1. Subject-Initiated Flow

When a human subject initiates the workflow, the following steps occur:

1. The subject invokes the agent application to perform a task.

2. The agent application calls an external endpoint. The external endpoint returns an OAuth challenge.

3. The agent application authenticates using an OAuth 2.0 Authorization Code flow RFC6749 (https://tools.ietf.org/html/rfc6749) access token. The access token contains sub and client_id claims as per [RFC9068]. If the access token represents a delegated flow (human delegating to agent), it MAY contain an act claim per [RFC8693].

4. The external endpoint submits the received access token along with its subject token to the Txn-Token Service. Subject token requirements are specified in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html).

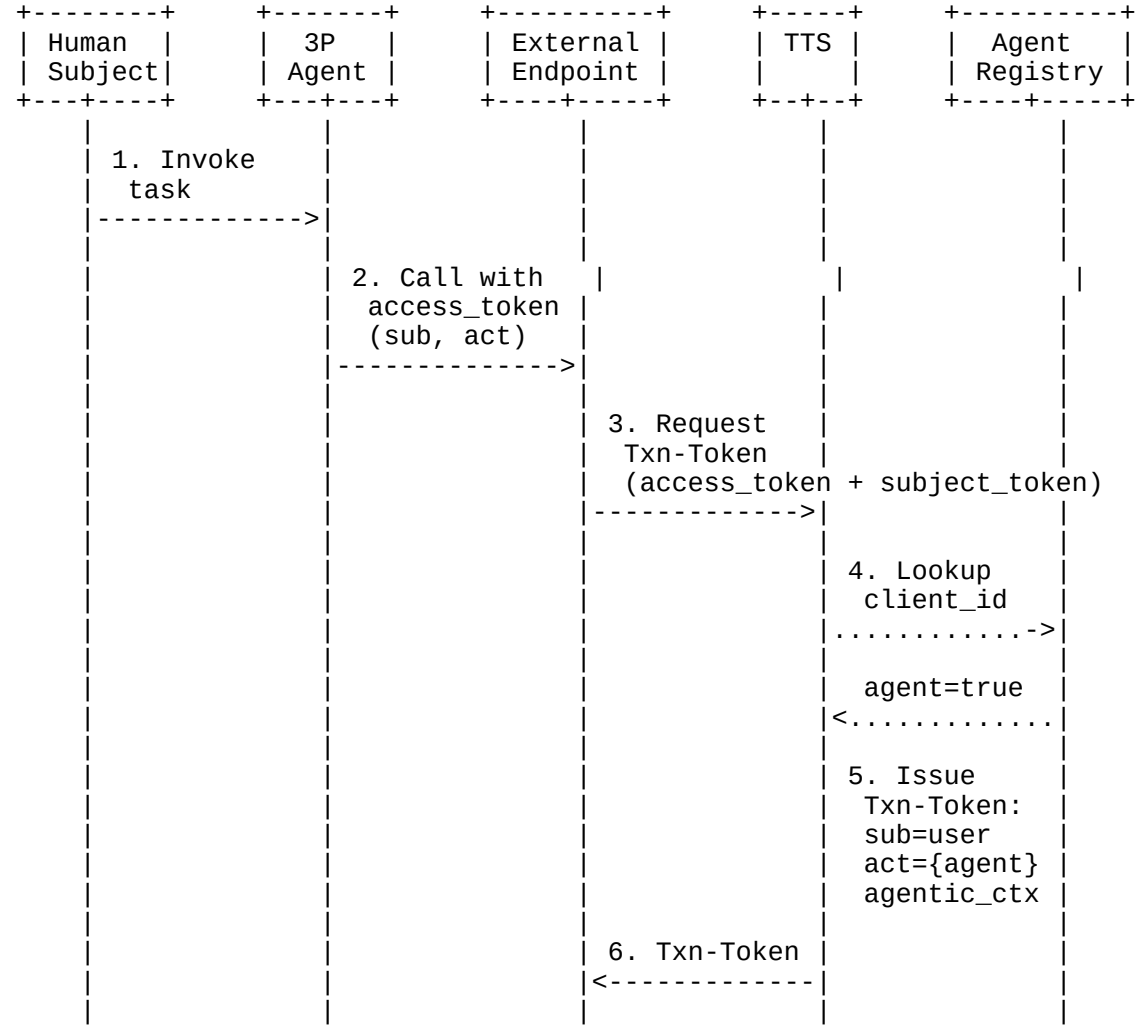
5. The Txn-Token Service validates the access token.

6. The Txn-Token Service populates the Txn-Token’s sub claim following the rules specified in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html). The sub claim is determined based on the subject_token provided in the request.

7. If the access token contains an act claim, the Txn-Token Service copies the act claim value to the Txn-Token’s act field. The value is copied as-is without modification.

8. If the TTS has access to an Agent Registry or equivalent mechanism, it performs a lookup using the client_id from the access token. If the client_id resolves to a registered agent, the TTS populates the agentic_ctx claim. This is the case when agent is within trust domain and user start request using agent within trust domain. If no Agent Registry is available, the TTS MAY still populate agentic_ctx using information derivable from the token exchange (e.g., client_id as current_actor).

9. The Txn-Token Service issues the Txn-Token to the requesting workload.



Legend:
- '-->' Solid: Primary request/response flow
- '...>' Dotted: Registry lookup (optional, per 則3.7)

3.2.2. Autonomous Flow

When the agent application operates autonomously, the following steps occur:

1. The agent application initiates a task based on an event or scheduled assignment.

2. The agent application calls an external endpoint. The OAuth challenge flow starts.

3. The agent application authenticates using an OAuth 2.0 Client Credentials Grant RFC6749 (https://tools.ietf.org/html/rfc6749).

4. The agent application uses the access token to call the external endpoint.

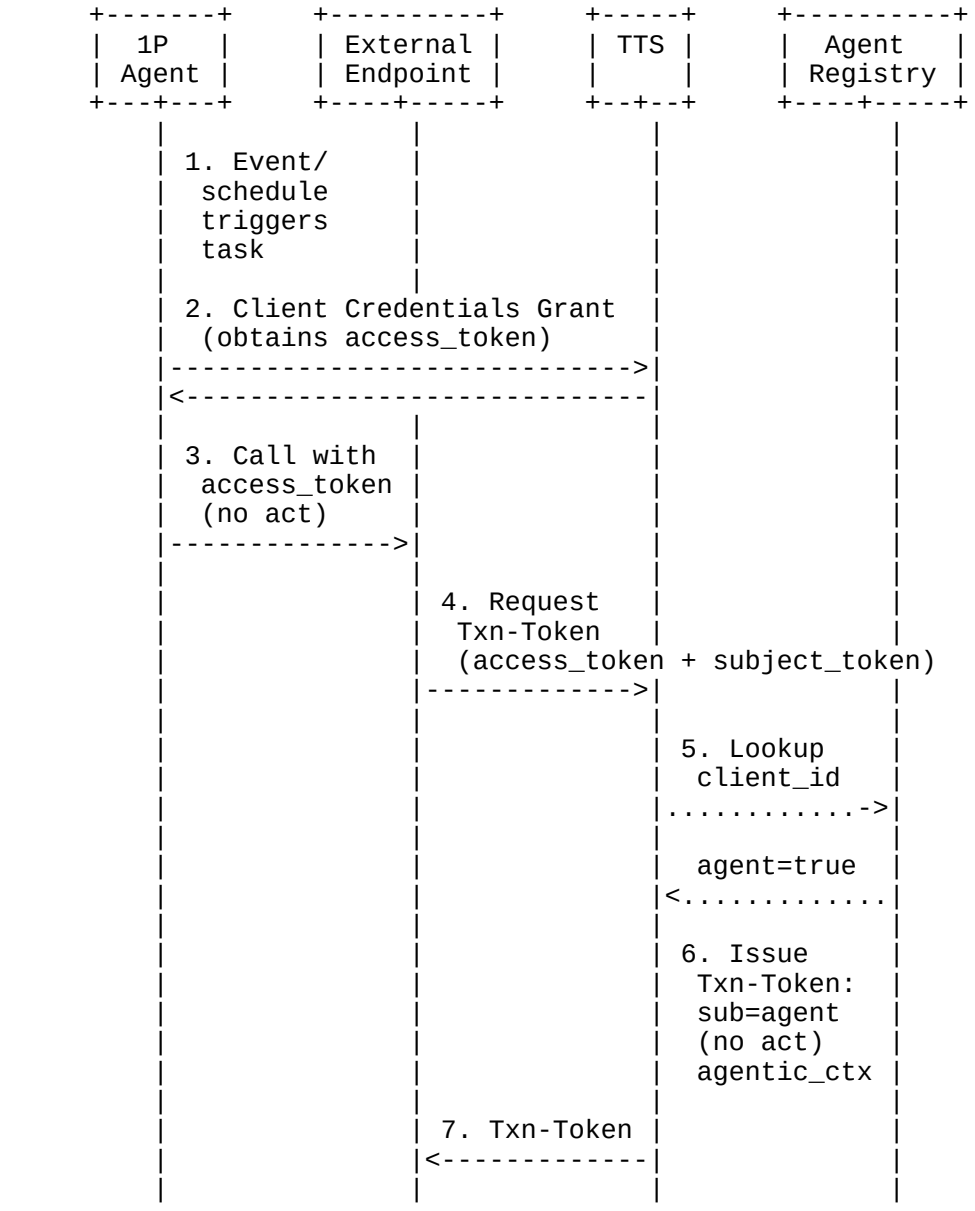
5. The external endpoint submits the received access token along with its subject token to the Txn-Token Service. Subject token requirements are specified in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html).

6. The Txn-Token Service validates the access token.

7. The Txn-Token Service populates the Txn-Token’s sub claim following the rules specified in OAUTH-TXN-TOKENS (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html). For autonomous agents, the sub claim is determined based on the subject_token provided in the request, which typically represents the agent’s own identity.

8. If the access token contains an act claim, the Txn-Token Service copies it to the Txn-Token. For Client Credentials Grant flows, the access token typically does not contain an act claim, and therefore the Txn-Token MUST NOT contain an act claim.

9. If the TTS has access to an Agent Registry, it performs a lookup and populates agentic_ctx if the caller is identified as an agent. If no Agent Registry is available, the TTS MAY still populate agentic_ctx using information derivable from the token exchange.



Note: In autonomous flows, act is absent (no human delegation). The originator in agentic_ctx is set from client_id.

3.3. Replacement tokens

Txn-Token Service provides capability to get a replacement Txn-Token as defined in the OAUTH-TXN-TOKENS.replacement flow (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html#name-creating-replacement-txn-to). This specification extends the replacement flow with agent-specific behavior.

When a workload identified as an agent receives a Transaction Token and intends to invoke a downstream workload, it SHOULD request a replacement Txn-Token from the TTS before making the downstream call. This ensures that agentic_ctx accurately reflects the current executing agent and that chain integrity metadata is maintained.

If a deployment enforces chain integrity policies based on hop_count or min_assurance_level, then agents MUST request replacement before downstream invocation. Failure to replace results in stale chain metadata that does not reflect the actual execution path.

3.3.1. Replacement When Caller is Identified as an Agent

When the workload requesting a replacement Txn-Token is identified as an agent (per Section 3.7), the following rules apply in addition to the base specification replacement rules:

* The sub, txn, and aud claims MUST NOT be modified (per base specification).

* The act claim, if present, MUST NOT be modified. The act claim is immutable throughout the entire transaction lifetime.

* The scope claim can only be attenuated (per base specification).

* The req_wl claim MUST be updated by appending the current requesting workload’s identifier, using the comma (,) character as separator per the base specification.

* The agentic_ctx claim MUST be updated as follows:

  - The current_actor field MUST be set to the identifier of the new requesting agent.

  - The chain_metadata.hop_count MUST be incremented by one.

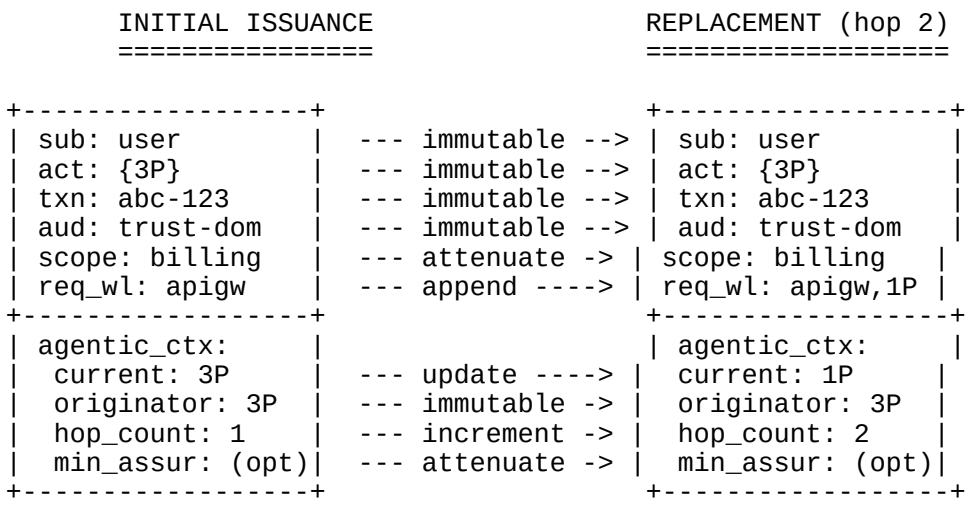
- If chain_metadata.min_assurance_level is present, it MUST be recalculated. The value MUST be the minimum of the existing min_assurance_level and the new agent's assurance level (monotonic attenuation).

- Any deployment-specific fields within agentic_ctx MAY be updated according to the trust domain's policies.

3.3.2. Replacement When Caller is Not Identified as an Agent

When the workload requesting a replacement Txn-Token is NOT identified as an agent, the TTS MUST follow the base specification [OAUTH-TXN-TOKENS] replacement rules without modification. The agentic_ctx claim, if present in the incoming Txn-Token, MUST be carried forward without modification. The act claim, if present, MUST NOT be modified. This means, even if agents are anywhere in the call chain, the internal workloads receiving a Txn-Token now gets claims in Txn-Token that inform if one or more agents were involved in the call chain and they can do additional authorization checks.

3.3.3. 3.4.3. Token State Transitions



3.4. Txn-Token Format

No changes to the JWT header from the base specification: typ MUST be txntoken+jwt, with a signing key identifier such as kid.

The Txn-Token body augments the base claim set with the act field (when present in the input access token) and the agentic_ctx claim for agent context. Existing claims like txn, sub, aud, iss, iat, exp, scope, tctx, and req_wl retain identical semantics, population rules, and immutability guarantees as defined in [OAUTH-TXN-TOKENS].

In this example, the agent is a 3rd-party agent not part of the trust domain. It hits the API Gateway in the trust domain, and the API Gateway requests a Txn-Token from the Txn-Token Service using the access token received from the 3P agent and its own subject token (to authenticate with the Txn-Token Service). The requesting workload is the API Gateway. The agent is identified by the act claim copied from the access token issued to the 3P agent to act on behalf of the user.

```
{
  "iat": 1697059200,
  "aud": "trust-domain.example",
  "exp": 1697059500,
  "txn": "c2dc3992-2d65-483a-93b5-2dd9f02c276e",
  "sub": "user:alice@example.com",
  "scope": "trade.stocks",
  "req_wl": "apigateway.trust-domain.example",
  "act": {
    "sub": "agent-identity-1"
  },
  "tctx": {
    "action": "BUY",
    "ticker": "MSFT",
    "quantity": "100"
  },
  "agentic_ctx": {
    "current_actor": "agent-identity-1",
    "originator": "agent-identity-1",
    "chain_metadata": {
      "hop_count": 1
    }
  }
}
```

3.5. Agentic Context

The Txn-Token MAY contain an agentic_ctx claim. Txn-Tokens are increasingly used in environments where transactions are executed by or with the assistance of autonomous or semi-autonomous agents (for example, Large Language Model (LLM)-based agents, workflow orchestrators, and policy-driven automation components). In such deployments, relying exclusively on subject identity and generic

transaction parameters is insufficient to make robust authorization decisions. Additional information about the agent chain and its operational context is often required.

The agentic_ctx claim provides a container for agent-specific context within the Transaction Token. This specification defines a minimal set of normative fields required for multi-agent flow integrity. Deployments MAY include additional fields within agentic_ctx as required by their trust domain policies.

The depth of context available within agentic_ctx differs between external and internal agents. For external agents entering the trust domain, the TTS populates the normative fields based on information available from the token exchange (e.g., act.sub or client_id) and assigns context commensurate with the trust domain's policy for unverified external actors. For internal agents within the trust domain, the TTS MAY additionally populate deployment-specific fields from verified internal sources (Agent Registry, hardware attestation, transport-layer identity). This asymmetry reflects the reality that the trust domain does not own or control external agents and therefore cannot verify their operational posture to the same degree.

3.5.1. Normative Fields

The following fields within agentic_ctx are defined by this specification:

current_actor: REQUIRED. A string identifying the agent currently executing the request. The value is the agent's identifier as determined by the TTS (e.g., from Agent Registry lookup, client_id, or act.sub). This field is updated during replacement flows when the new caller is identified as an agent.

originator: REQUIRED. A string identifying the agent that initiated the transaction chain. This value is set when the initial Txn-Token is issued and MUST NOT be modified during replacement flows. For single-agent flows, originator and current_actor have the same value. The originator field exists to support authorization policy evaluation in scenarios where the act claim is absent (e.g., autonomous agent flows using Client Credentials Grant where no delegation occurs). In flows where act is present, the originator value will typically match act.sub. This field ensures that the chain origin is always available within agentic_ctx regardless of whether the transaction involves human delegation.

chain_metadata: REQUIRED. A JSON object containing chain-level integrity metadata. The following sub-fields are defined:

hop_count: REQUIRED. An integer representing the number of agent transitions in the current chain. Set to 1 at initial issuance. Incremented by 1 at each replacement where the caller is identified as an agent.

min_assurance_level: OPTIONAL. A string representing the minimum assurance level across all agents in the chain. When present, the TTS MUST set this to the minimum of the existing value and the new agent's assurance level during each replacement flow (monotonic attenuation). This field is only meaningful in deployments that define an ordered set of assurance levels and maintain an Agent Registry or equivalent source from which agent assurance levels can be determined. This specification does not define a fixed enumeration of assurance level values. Deployments MUST define their own ordered set and comparison function. Non- normative examples include: "unverified", "low", "medium", "high".

3.5.2. Deployment-Specific Fields

Beyond the normative fields defined in Section 3.6.1, deployments MAY include additional fields within agentic_ctx to carry context specific to their trust domain. This specification does not prescribe the structure or semantics of these additional fields.

Examples of deployment-specific context that MAY be included:

* Operational posture: Hardware attestation status (e.g., TEE type), runtime integrity measurements, security tier classification.

* Provenance: Cryptographic hashes of the agent's behavioral configuration (system prompt, model parameters), software version identifiers, source code references.

* Identity metadata: Workload identity details (e.g., SPIFFE SVID), execution node information, deployment environment identifiers.

* Risk signals: Real-time risk scores, anomaly detection flags, behavioral drift indicators.

The TTS is responsible for populating deployment-specific fields using verified sources appropriate to the trust domain's security requirements. The TTS MUST NOT rely on self-reported data from the agent for any field that influences authorization decisions.

In deployments without an Agent Registry, the TTS MUST NOT populate fields that require external verification (such as min_assurance_level or deployment-specific posture fields) without a verified source.

3.5.2.1. Example with Deployment-Specific Context

The following is a non-normative example showing agentic_ctx with deployment-specific posture and provenance fields in addition to the normative fields:

```
{
  "agentic_ctx": {
    "current_actor": "spiffe://prod.acme.com/billing-agent",
    "originator": "3p-assistant-ext-99",
    "chain_metadata": {
      "hop_count": 2,
      "min_assurance_level": "low"
    },
    "posture": {
      "tee": "aws-nitro-enclave",
      "assurance": "high",
      "boot_gold": true
    },
    "prov": {
      "manifest_hash": "sha256:e3b0c44298fc1c149afbf4c8996fb924...",
      "model_id": "llama-3.1-70b-v1",
      "version": "2.4.1"
    },
    "identity": {
      "workload_id": "spiffe://prod.acme.com/billing-agent",
      "origin_node": "node-77-east-1"
    }
  }
}
```

Note: The posture, prov, and identity fields shown above are deployment-specific examples. Their structure and semantics are determined by the trust domain’s policies and are not defined by this specification.

3.5.3. Implementation Guidance

The following guidance applies to implementations using deployment-specific fields within agentic_ctx:

* Resource Servers receiving a Txn-Token with agentic_ctx SHOULD evaluate the normative fields (current_actor, originator, chain_metadata) for chain integrity decisions.

* Resource Servers MAY evaluate deployment-specific fields against local policy to make fine-grained authorization decisions.

* If a Resource Server does not recognize a deployment-specific field, it MUST ignore that field and MUST NOT reject the token solely on that basis.

* The TTS SHOULD populate deployment-specific fields through verified sources such as: hardware attestation documents verified against manufacturer roots of trust, out-of-band registry lookups using the agent's authenticated client_id, and transport-layer identity (e.g., mTLS certificates or SPIFFE SVIDs).

3.6. Agent Identification

The Transaction Token Service needs to determine whether a requesting workload is an agent in order to apply the appropriate token issuance and replacement rules defined in this specification.

3.6.1. Identification Mechanisms

The TTS SHOULD have access to an Agent Registry or equivalent mechanism to identify agent callers. When an Agent Registry is available, the TTS uses it to determine whether agent-specific token issuance rules apply and to source metadata for agentic_ctx population.

When the TTS receives a Txn-Token request or replacement request and has access to an Agent Registry, it SHOULD perform the following determination:

1. Extract the client_id from the presented access token.

2. Query the Agent Registry to determine if the client_id corresponds to a registered agent.

3. If the client_id resolves to a registered agent:

    * The TTS SHOULD apply the agent-specific token issuance rules defined in this specification.

    * The TTS SHOULD populate the agentic_ctx claim with at minimum the normative fields defined in Section 3.6.1.

* If the access token contains an act claim, the TTS MUST copy it to the Txn-Token without modification.

4. If the client_id does NOT resolve to a registered agent:

* The TTS SHOULD follow the base specification [OAUTH-TXN-TOKENS] rules without modification.

* The TTS SHOULD NOT populate an agentic_ctx claim.

In deployments without an Agent Registry, the TTS MAY use alternative mechanisms to identify agents, such as:

* The presence of an act claim in the access token (indicating delegation to an agent).

* Convention-based client_id patterns.

* Out-of-band configuration or policy rules.

The specific mechanism for agent identification is deployment-defined. This specification defines the behavior once the determination is made, not the mechanism itself.

3.6.2. Registry Contents

When an Agent Registry is used, it SHOULD contain, at minimum, the following information for each registered agent:

* client_id: The OAuth 2.0 client identifier for the agent.

* agent_name: A human-readable name for the agent.

If the deployment uses min_assurance_level, the registry SHOULD also contain:

* assurance_level: The assurance level assigned to this agent, used for min_assurance_level calculation during replacement flows.

Additional registry contents are deployment-specific and MAY include fields such as expected posture, manifest hashes, or behavioral configuration references.

The implementation of the Agent Registry is out of scope for this specification. Deployments MAY use existing service registries, dedicated agent catalogs, or policy engines to fulfill this role.

4. Multi-Agent Flows

In complex agentic workflows, a transaction often originates from a 3rd-party (3P) agent and propagates through one or more 1st-party (1P) agents within the local trust domain. To maintain Zero Trust integrity, this specification uses the agentic_ctx claim to track the chain of agents involved in the transaction. This ensures that downstream Resource Servers can evaluate the chain integrity and overall assurance level, rather than relying solely on the identity of the immediate caller.

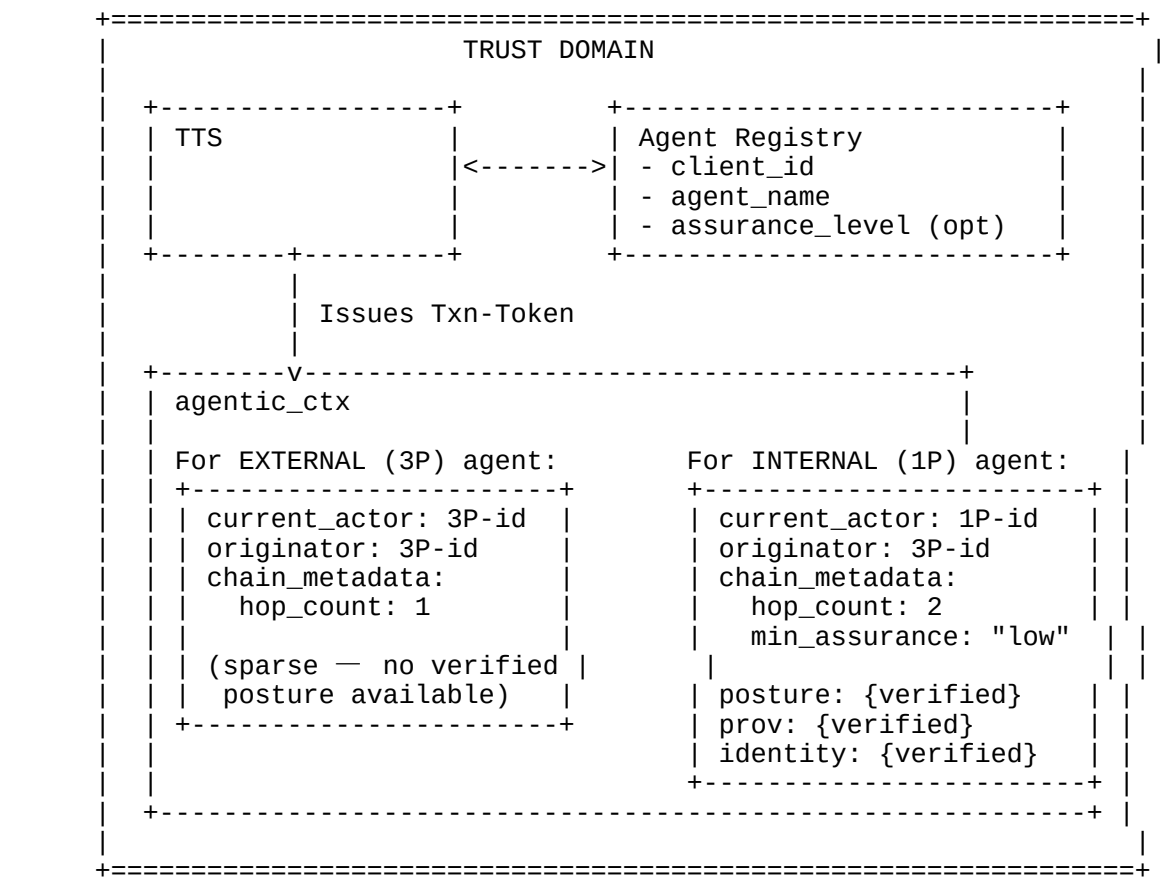
4.1. External vs. Internal Agents

The agentic_ctx differentiates between two categories of agents through the normative current_actor and originator fields:

External Agents (3P): Agents that are not owned or controlled by the trust domain. They enter the trust domain through an external endpoint. Because their hardware and software are outside local control, the TTS has limited ability to verify their operational posture. Their identity is typically captured via the act claim (from the access token) and reflected in the originator field. Deployment- specific fields for external agents are typically sparse or marked as "unverified".

Internal Agents (1P): Agents that operate within the trust domain and are owned or controlled by the trust domain operator. The TTS has full ability to verify their operational posture through internal mechanisms (Agent Registry, hardware attestation, transport-layer identity). Their identity is reflected in the current_actor field during replacement flows. Deployment-specific fields for internal agents can be richly populated from verified sources.

This distinction is not a protocol-level differentiation — both external and internal agents are identified through the same mechanisms (Section 3.7). The distinction affects the _depth_ of context the TTS can populate within agentic_ctx.



Note: External agents get normative fields only (TTS cannot verify their posture). Internal agents get normative + deployment-specific fields populated from verified sources.

4.2. Monotonic Attenuation of Trust

A chain's security posture is only as strong as its weakest link. When min_assurance_level is used, the TTS MUST calculate it during every token replacement flow. If a "high" assurance internal agent is triggered by a "low" assurance 3P originator, the transaction's overall assurance level remains "low". This prevents Identity Laundering, where unverified external agents bypass security guardrails by proxying requests through internal services.

The Transaction Token Service (TTS) determines the min_assurance_level by comparing the existing value with the assurance level of the new requesting agent (as recorded in the Agent Registry) and selecting the minimum.

4.2.1. Delegation via Replacement Flow

When an internal agent (the "Delegatee") requires a Transaction Token to continue a chain initiated by another actor (the "Delegator"), it SHOULD follow the replacement flow procedures defined in [OAUTH-TXN-TOKENS] with the following modifications:

* Subject Immutability: The txn, sub, and aud claims MUST be copied from the subject_token to the new Transaction Token without modification.

* Actor Immutability: The act claim, if present, MUST be copied without modification. The act claim represents the original agent delegation and MUST NOT change throughout the transaction lifetime.

* Current Actor Update: The current_actor field MUST be set to the Delegatee’s identifier.

* Chain Metadata Update: The TTS MUST increment hop_count and, if min_assurance_level is present, recalculate it.

* Workload Tracking: The TTS MUST append the Delegatee’s workload identifier to the req_wl claim.

* Originator Preservation: The originator field MUST NOT be modified.

If a deployment enforces chain integrity policies, internal agents MUST request replacement before downstream invocation to ensure accurate chain metadata.

4.2.2. Multi-Agent Example

The following example illustrates a multi-agent flow where a 3rd-party agent ("3p-assistant-ext-99") initiates a transaction on behalf of a user, and the request is subsequently handled by an internal 1st-party agent ("1p-billing-svc-v2") within the trust domain.

Flow description:

1. The 3P agent ("3p-assistant-ext-99") obtains an access token via OAuth 2.0 Authorization Code flow on behalf of the user. The access token contains an act claim identifying the agent: {"sub": "3p-assistant-ext-99"}.

2. The 3P agent calls the trust domain’s API Gateway with this access token.

3. The API Gateway submits the access token and its own subject token to the TTS, requesting a Txn-Token.

4. The TTS validates the access token, identifies the caller as an agent (via Agent Registry lookup or alternative mechanism), and issues the initial Txn-Token with:

   * sub populated from the subject_token (the user)

   * act copied from the access token: {"sub": "3p-assistant-ext-99"}

   * agentic_ctx.current_actor set to "3p-assistant-ext-99"

   * agentic_ctx.originator set to "3p-assistant-ext-99"

   * agentic_ctx.chain_metadata.hop_count set to 1

   * req_wl set to the API Gateway’s identifier

5. The API Gateway forwards the Txn-Token to an internal agent ("1p-billing-svc-v2").

6. The internal agent needs to call a downstream Resource Server. It requests a replacement Txn-Token from the TTS, presenting the existing Txn-Token as the subject_token.

7. The TTS identifies "1p-billing-svc-v2" as an agent. It issues a replacement Txn-Token with:

   * sub, txn, aud unchanged

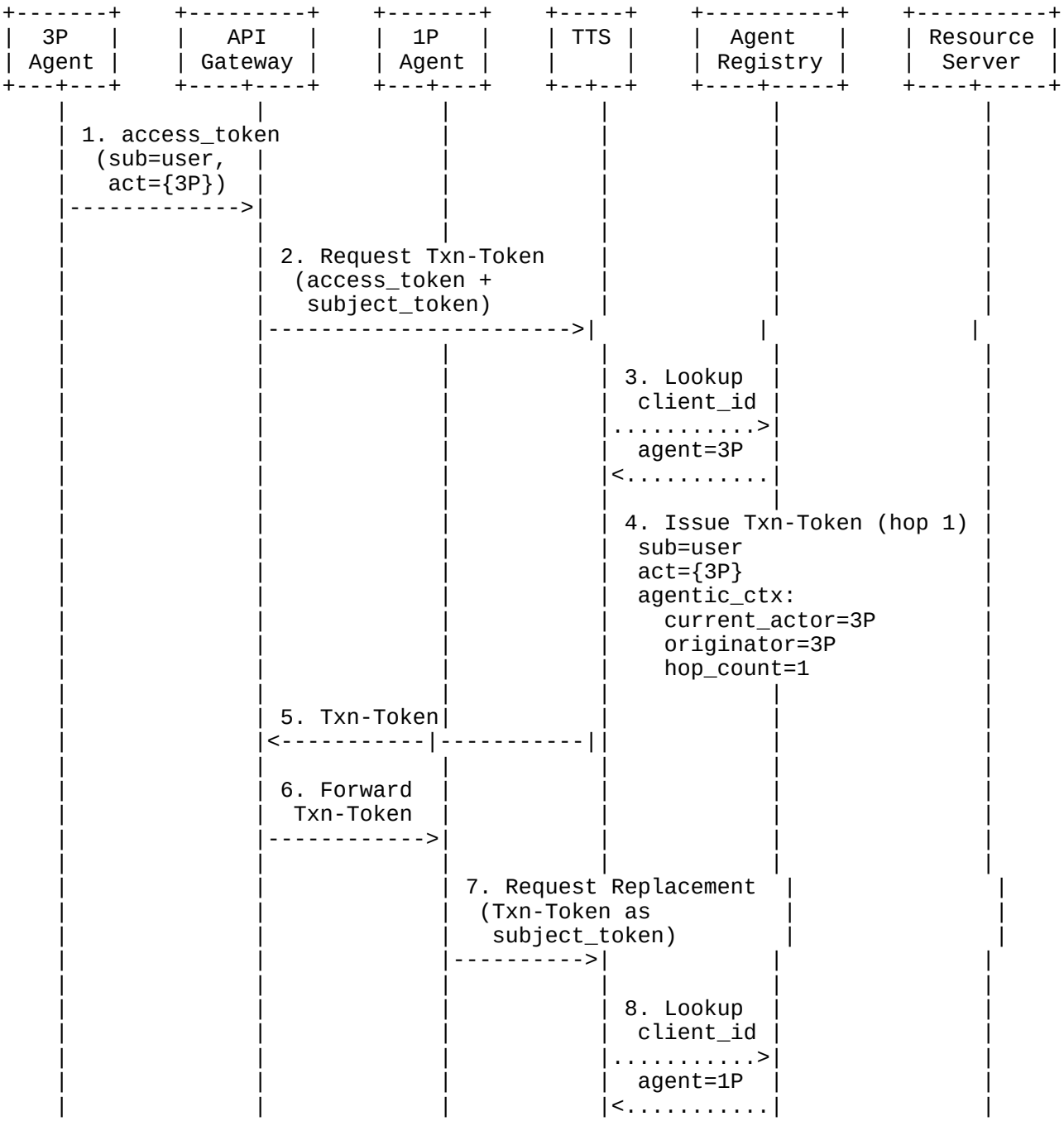
   * act unchanged: {"sub": "3p-assistant-ext-99"}

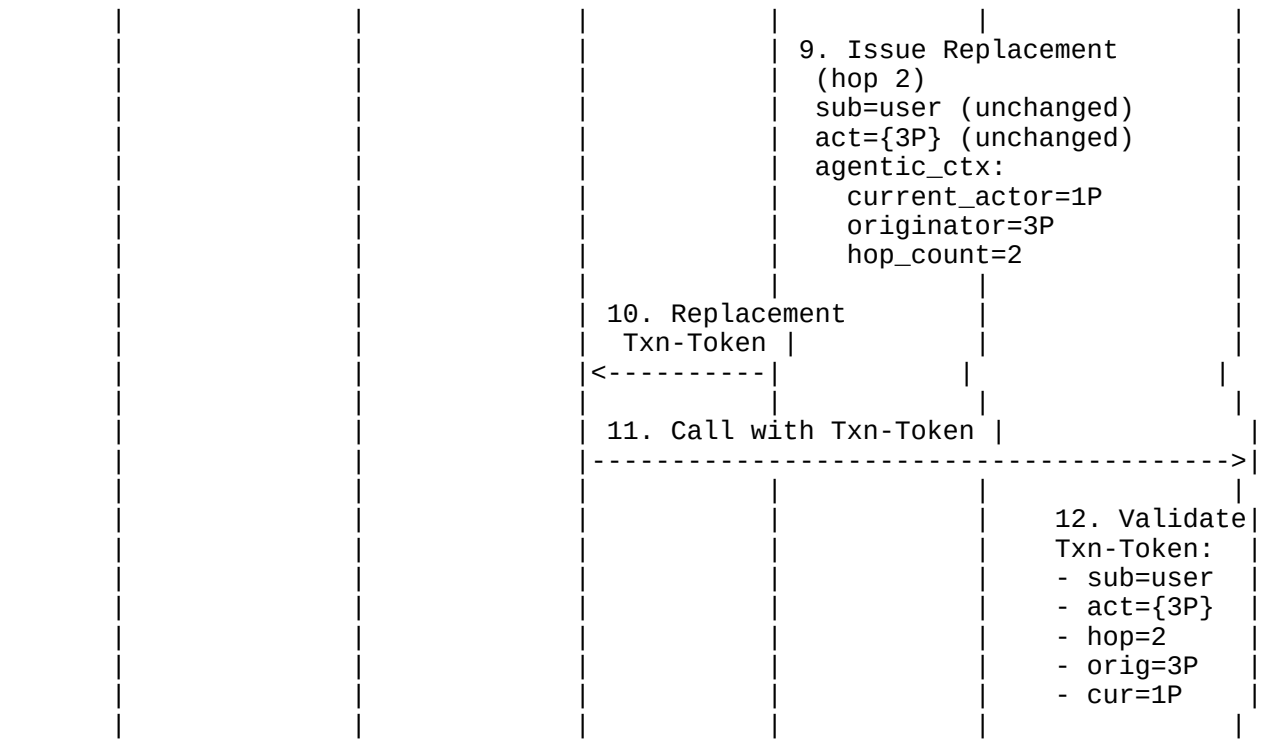
   * agentic_ctx.current_actor updated to "1p-billing-svc-v2"

   * agentic_ctx.originator unchanged: "3p-assistant-ext-99"

   * agentic_ctx.chain_metadata.hop_count incremented to 2

* req_wl appended with the internal agent's identifier





Legend: - --> Solid: Primary request/response flow - ...> Dotted: Agent Registry lookup (optional, per § 3.7) - The following is the Txn-Token at the final hop (after step 7), showing only normative fields:

```
{
  "iat": 1712850000,
  "aud": "trust-domain.example",
  "exp": 1712850300,
  "txn": "abc-123-xyz",
  "sub": "user_8821@example.com",
  "iss": "https://txn-svc.trust-domain.example",
  "scope": "billing.process",
  "req_wl": "apigateway.trust-domain.example,1p-billing-svc-v2.trust-domain.example",
  "act": {
    "sub": "3p-assistant-ext-99"
  },
  "agentic_ctx": {
    "current_actor": "1p-billing-svc-v2",
    "originator": "3p-assistant-ext-99",
    "chain_metadata": {
      "hop_count": 2
    }
  }
}
```

Note: The act claim value {"sub": "3p-assistant-ext-99"} is identical to what was set at hop 1. It represents the original agent delegation and is never modified during replacement flows. The originator field within agentic_ctx mirrors this value for convenience in authorization policy evaluation.

The following is a non-normative example of the same token in a deployment that uses min_assurance_level and deployment-specific posture fields:

```
{
  "iat": 1712850000,
  "aud": "trust-domain.example",
  "exp": 1712850300,
  "txn": "abc-123-xyz",
  "sub": "user_8821@example.com",
  "iss": "https://txn-svc.trust-domain.example",
  "scope": "billing.process",
  "req_wl": "apigateway.trust-domain.example,1p-billing-svc-v2.trust-domain.example",
  "act": {
    "sub": "3p-assistant-ext-99"
  },
  "agentic_ctx": {
    "current_actor": "1p-billing-svc-v2",
    "originator": "3p-assistant-ext-99",
    "chain_metadata": {
      "hop_count": 2,
      "min_assurance_level": "low"
    },
    "posture": {
      "tee": "aws-nitro-enclave",
      "assurance": "high"
    },
    "prov": {
      "manifest_hash": "sha256:4455aabb..."
    }
  }
}
```


4.2.3. Loop Prevention

To prevent infinite recursion in autonomous agentic loops, the TTS MUST track the hop_count within chain_metadata and enforce a maximum depth. If the hop_count exceeds a deployment-defined threshold, the replacement request MUST be rejected.

Implementations MAY include additional chain tracking fields within agentic_ctx (such as a path of agent identifiers) but such fields are not defined by this specification.

5. Security Considerations

All the security considerations mentioned in [OAUTH-TXN-TOKENS] apply.

5.1. Token Replay Protection

Implementations MUST enforce strict token lifetime validation. The short-lived nature of Transaction Tokens helps mitigate replay attacks, but implementations SHOULD also consider:

* Implementing token tracking mechanisms within trust domains

* Validating token usage context

5.1.1. Actor Identity Security

* Implementations MUST validate act claims in tokens.

* The Txn-Token Service MUST verify the authenticity of actor context before token issuance.

* During replacement flow, the Txn-Token Service MUST NOT modify the act field in the incoming Txn-Token. The immutability of act ensures that the original agent delegation is always visible and cannot be overwritten, preventing identity laundering attacks.

5.1.2. Subject Context Protection

* Systems MUST prevent unauthorized modifications to the sub claim during token propagation. Txn-Tokens are cryptographically signed to ensure integrity.

* During replacement flow, the Txn-Token Service MUST NOT modify the sub claim in the incoming Txn-Token.

* The Txn-Token Service MUST follow the subject population rules defined in [OAUTH-TXN-TOKENS] to ensure proper subject representation.

5.1.3. Transaction Chain Integrity

* Implementations MUST maintain cryptographic integrity of the token chain.

* Services MUST validate tokens at trust domain boundaries.

* Systems MUST implement protection against token tampering during service-to-service communication.

5.1.4. AI Agent Specific Controls

* Implementations MUST enforce scope boundaries for AI agent operations.

* Systems SHOULD implement behavioral monitoring for AI agent activities by logging act, sub, and agentic_ctx claims in audit logs.

* Systems MUST maintain audit trails of AI agent activities.

5.1.5. Token Transformation Security

* The Txn-Token Service MUST validate all claims during access token to Txn-Token conversion.

* Implementations MUST verify signatures and formats of all tokens.

* Systems MUST prevent unauthorized manipulation during token transformation.

* The Txn-Token Service MUST ensure that the act field accurately represents the actor identity from the access token.

5.1.6. Replacement Token Considerations

* Systems MUST verify the authenticity and validity of original tokens before replacement.

* Systems MUST implement controls to prevent unauthorized replacement requests.

* The immutability of act, sub, and originator during replacement ensures consistent identity context throughout the transaction lifecycle.

5.1.7. Infrastructure Security

* All component communications MUST use secure channels.

* Implementations MUST enforce strong authentication of the Authorization Server.

* Systems MUST implement regular rotation of cryptographic keys.

* Trust domain boundaries MUST be clearly defined and enforced.

5.1.8. Prevention of Identity Laundering

* When min_assurance_level is used, implementations MUST enforce Monotonic Attenuation.

* The TTS MUST NOT allow a replacement token to have a higher min_assurance_level than the incoming subject token, even if the current actor has a higher assurance level.

* This prevents a low-trust 3rd-party originator from "laundering" its identity through a high-trust internal agent to bypass security guardrails at the Resource Server.

* The immutability of the act claim and originator field provides an additional safeguard: the original delegating agent identity is always preserved and visible to downstream Resource Servers regardless of how many replacement hops occur.

5.1.9. Integrity of the Agent Registry

* When an Agent Registry is used, the security of the agent identification mechanism (Section 3.7) relies on the integrity of the registry.

* If an attacker can add entries to the registry, they can cause the TTS to treat arbitrary workloads as agents, triggering agent-specific token issuance rules inappropriately.

* If an attacker can modify entries, they can alter the assurance level assigned to an agent, undermining the monotonic attenuation guarantee.

* Access to the Agent Registry MUST be restricted to authorized deployment pipelines and protected with strong integrity controls.

* The Agent Registry SHOULD support audit logging of all modifications.

* Implementations SHOULD use cryptographic signatures or content-addressable storage to detect unauthorized modifications.

5.1.10. Agent Identification Mechanism Security

* When the TTS relies on client_id lookup in the Agent Registry to determine whether a caller is an agent, a compromised client_id could lead to incorrect agent classification.

* Mitigations include:

- Binding client_id to transport-layer identity (e.g., mTLS certificate or SPIFFE SVID).

- Requiring additional verification before accepting agent classification.

- Implementing rate limiting and anomaly detection on registry lookups.

5.1.11. Deployment-Specific Field Security

* When deployments include additional fields within agentic_ctx (such as posture or provenance), the TTS MUST NOT rely on self-reported data from the agent for any field that influences authorization decisions.

* Hardware-backed fields (e.g., TEE attestation) SHOULD be derived from cryptographic attestation documents verified against the hardware manufacturer’s root of trust.

* Software-related fields (e.g., manifest hashes) SHOULD be retrieved via out-of-band registry lookups rather than agent self-assertion.

5.1.12. Loop Detection and Recursion Limits

* The hop_count within chain_metadata is REQUIRED to prevent resource exhaustion in autonomous agentic workflows.

* The TTS SHOULD enforce a maximum hop_count to prevent resource exhaustion attacks. If the count exceeds a defined threshold, the replacement request MUST be rejected.

5.1.13. Data Leakage in Context Propagation

* The agentic_ctx may contain deployment-specific fields with sensitive internal information.

* When a Txn-Token egresses the trust domain (e.g., a 1st-party agent calling an external 3rd-party service), the TTS MUST strip deployment-specific internal fields from the token to prevent infrastructure leakage.

* Only the normative fields and the minimum necessary context should be egressed from the trust domain.

5.1.14. Stale Chain Metadata

* If agents propagate incoming Txn-Tokens without requesting replacement, the agentic_ctx will contain stale metadata that does not reflect the actual execution path.

* Deployments that rely on hop_count or min_assurance_level for authorization decisions MUST enforce mandatory replacement for agents (see Section 3.4).

* Resource Servers SHOULD be aware that in deployments without mandatory replacement, hop_count represents a lower bound on the actual number of agent transitions.

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, https://www.rfc-editor.org/rfc/rfc2119 (https://www.rfc-editor.org/rfc/rfc2119).

RFC8174 (https://datatracker.ietf.org/doc/html/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

RFC6749 (https://tools.ietf.org/html/rfc6749) Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, https://www.rfc-editor.org/rfc/rfc6749 (https://www.rfc-editor.org/rfc/rfc6749).

RFC7519 (https://tools.ietf.org/html/rfc7519) Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, https://www.rfc-editor.org/rfc/rfc7519 (https://www.rfc-editor.org/rfc/rfc7519).

RFC7515 (https://tools.ietf.org/html/rfc7515) Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, https://www.rfc-editor.org/rfc/rfc7515 (https://www.rfc-editor.org/rfc/rfc7515).

RFC8693 (https://tools.ietf.org/html/rfc8693) Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, https://www.rfc-editor.org/rfc/rfc8693 (https://www.rfc-editor.org/rfc/rfc8693).

RFC9068 (https://tools.ietf.org/html/rfc9068) Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, https://www.rfc-editor.org/rfc/rfc9068 (https://www.rfc-editor.org/rfc/rfc9068).

RFC9396 (https://datatracker.ietf.org/doc/html/rfc9396) T. Lodderstedt, J. Richer, B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, https://www.rfc-editor.org/rfc/rfc9396 (https://www.rfc-editor.org/rfc/rfc9396).

OAUTH-TXN-TOKENS (https://datatracker.ietf.org/doc/draft-tulshibagwale-oauth-transaction-tokens) Atul Tulshibagwale, George Fletcher, Pieter Kasselman, "OAuth Transaction Tokens", https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html (https://drafts.oauth.net/oauth-transaction-tokens/draft-ietf-oauth-transaction-tokens.html)

Appendix A. Acknowledgments

name: Dr. Chunchi (Peter) Liu email: Liuchunchi(Peter) <liuchunchi=40huawei.com@dmarc.ietf.org>

Appendix B. Contributors

name: Atul Tulshibagwale org: SGNL email: atul@sgnl.ai

Author’s Address

ASHAY RAUT
Email: asharaut@amazon.com