Delegation Chain Splicing in OAuth 2.0 Token Exchange (RFC 8693)

Summary

A structural weakness in how RFC 8693 represents delegation chains via nested `act` claims allows an attacker with access to two or more legitimately-issued delegation tokens to recombine their `act` claim steps into a fabricated delegation chain that never actually occurred. I call this **chain splicing**.

The attack exploits the fact that nested `act` claims are plain JSON objects with no per-step integrity protection. Unless implementations independently sign each delegation step, there is no cryptographic mechanism to verify that a given chain of `act` claims represents an actual sequence of delegation events.

Affected specifications: - RFC 8693 (OAuth 2.0 Token Exchange) — `act` claim (§4.1) - OIDC-A 1.0 — `delegation_chain` claim - draft-oauth-ai-agents-on-behalf-of-user — tokens containing `act` claims

Severity: Critical for systems where the delegation chain is used for authorization decisions, audit trail integrity, or any form of policy enforcement. Given the practical need for such checks in multi-agent systems, the likelihood of an implementation being vulnerable is high, despite the spec's guidance that the chain should be "informational only."

Background: How `act` Claims Work

RFC 8693 §4.1 defines the `act` (actor) claim as a recursive JSON structure representing who is currently acting on behalf of the token subject:

```
{
  "sub": "user-alice",
  "act": {
    "sub": "agent-B",
    "act": {
      "sub": "agent-A"
    }
  }
}
```

This token asserts: "Agent A delegated to Agent B, who is currently acting on behalf of Alice."

The spec states (§4.1): "for the purpose of applying access control policy, the consumer of a token MUST only consider the token's top-level claims and the party identified as the current actor by the act claim. Prior actors identified by any nested act claims are informational only."

The delegation history is carried as nested JSON — **but the spec does not require that individual steps be signed or independently verifiable.**

The Attack

Prerequisites

The attacker is a compromised or malicious intermediary agent that participates in at least one legitimate delegation chain and can obtain a valid actor credential. Obtaining the actor credential is straightforward:

- **Self-authentication (lowest barrier):** The intermediary authenticates itself to any trusted STS using its own client credentials and obtains a token where it is the subject. This is a standard OAuth 2.0 flow — no observation or interception of other chains is required.
- **Participation in multiple flows:** The intermediary receives tokens as part of its normal function in different delegation flows and retains them.
- **Shared infrastructure:** Tokens are logged, cached, or observable through shared agent deployment infrastructure.

Mechanism

The attack exploits the token exchange endpoint (RFC 8693 §2), which accepts two inputs: a `subject_token` (the principal being acted on behalf of) and an `actor_token` (the identity of the party requesting to act). The STS validates each token **independently** — both carry valid signatures from their respective issuers. It then issues a **new**, properly-signed token combining the two.

The splice occurs when a compromised or malicious intermediary presents tokens from **different delegation contexts** to the STS:

Chain 1 (legitimate): Alice -> Agent-X -> Agent-Y

```
Token exchange request:
  subject_token = Alice's token
  actor_token   = Agent-X's credential
Result: {sub: "user-alice", act: {sub: "agent-X"}}

Then Agent-X calls exchange again:
  subject_token = above token (Alice, acted on by Agent-X)
  actor_token   = Agent-Y's credential
Result: {sub: "user-alice", act: {sub: "agent-Y", act: {sub: "agent-X"}}}
```

Chain 2 (legitimate): Bob -> Agent-M -> Agent-N

```
Similar flow producing:
  {sub: "user-bob", act: {sub: "agent-N", act: {sub: "agent-M"}}}
```

The splice: Agent-X is a compromised intermediary that participates in Chain 1. To perform the splice, Agent-X needs a valid credential identifying Agent-N as the actor. It can obtain this by:

1. **Observing Agent-N's credential** from Chain 2 (through logging, shared infrastructure, or direct compromise), or
2. **Having Agent-N authenticate itself** — if Agent-X controls or collaborates with Agent-N, Agent-N can simply authenticate to any trusted STS using its own client credentials and hand the resulting token to Agent-X. No interception is required.

Agent-X now calls the token exchange endpoint with:

```
subject_token = Alice's token (from Chain 1)
actor_token   = Agent-N's credential (obtained via either method above)
```

The STS validates Alice's token — **valid signature, legitimate token.** Validates Agent-N's actor credential — **valid signature, legitimate credential.** The STS has no mechanism to verify that these two tokens belong to the same delegation flow. It issues:

```
{
  "sub": "user-alice",
  "scope": "documents:read documents:write",
  "act": {
    "sub": "agent-N",
    "act": {
      "sub": "agent-X"
    }
  }
}
```

This token is **properly signed by the STS** — the JWS signature is cryptographically valid. But the delegation chain it asserts (Alice -> Agent-X -> Agent-N) **never actually occurred.** Agent-N was only ever authorized to act in Bob's chain. Agent-X never legitimately delegated to Agent-N on Alice's behalf.

The resource server receives a validly-signed token from a trusted STS. It has no basis to reject it.

Scenario 1: Healthcare Records System

A hospital network uses AI agents to assist physicians across departments.

Legitimate chains: - Dr. Patel (oncology) -> Clinical Notes Agent -> Lab Results Agent *Purpose: Agent retrieves lab results to summarize in clinical notes*

- Dr. Kim (radiology) -> Imaging Agent -> Report Generator Agent *Purpose: Agent generates radiology reports from imaging data*

Spliced chain: Dr. Patel -> Clinical Notes Agent -> Report Generator Agent

The Report Generator Agent was never authorized to access Dr. Patel's oncology patient data. It was designed to work with radiology imaging data only. But the spliced chain claims it was delegated through Dr. Patel's Clinical Notes Agent, potentially granting it access to oncology records, chemotherapy plans, and sensitive patient notes it should never see.

Impact: The resource server sees a valid-looking chain: a known physician delegated to a known agent, which sub-delegated to another known agent. All identities are real. All agents are registered. The chain just never happened.

Scenario 2: Multi-Tenant SaaS Platform

A SaaS platform allows enterprise customers to deploy AI agents that access shared infrastructure APIs.

Legitimate chains: - Acme Corp (tenant-acme) -> Acme's Data Pipeline Agent -> Shared Storage Agent *Scope: `storage:read` on tenant-acme's partition*

- Globex Inc (tenant-globex) -> Globex's Analytics Agent -> Shared Storage Agent *Scope: `storage:read storage:write` on tenant-globex's partition*

Spliced chain: Acme Corp -> Acme's Data Pipeline Agent -> Globex's Analytics Agent

Now Globex's Analytics Agent appears to be operating under Acme Corp's delegation chain. If the resource server uses the top-level `sub` (tenant-acme) to determine partition access, Globex's agent gains read access to Acme's data.

The danger is amplified because the Shared Storage Agent is a legitimate, commonly-used agent that appears in many chains. An attacker who can observe its delegation tokens across tenants has a library of `act` claim fragments to splice from.

Impact: Cross-tenant data access through fabricated delegation provenance.

Scenario 3: CI/CD and Infrastructure Automation

A DevOps platform uses AI agents for deployment orchestration across environments.

Legitimate chains: - Release Manager -> Build Agent -> Staging Deploy Agent *Scope: `deploy:staging`*

- SRE On-Call -> Incident Agent -> Production Rollback Agent *Scope: `deploy:production rollback:execute`*

Spliced chain: Release Manager -> Build Agent -> Production Rollback Agent

The Production Rollback Agent was only authorized under the SRE On-Call's incident response chain — a high-trust flow with additional approval gates. The spliced chain makes it appear that a routine release process delegated to the production rollback agent, bypassing incident response approval requirements.

Impact: Production deployment actions executed under fabricated delegation authority, bypassing approval gates.

Why This Happens

The root cause is that **the token exchange endpoint validates its inputs independently but has no mechanism to verify they belong to the same delegation context.**

RFC 8693 §2 defines the exchange as accepting a `subject_token` and an optional `actor_token`. The STS verifies each token's signature and validity. It then constructs a new token with an `act` claim representing the delegation. But the spec does not define:

1. **No cross-validation of inputs.** The STS validates the `subject_token` signature and the `actor_token` signature independently. There is no requirement that the STS verify these tokens originate from the same delegation flow, authorization session, or trust context.
2. **No binding between delegation steps.** There is no `aud`/`sub` chaining requirement within nested `act` claims. Unlike OIDC-A's `delegation_chain` (which specifies that the `aud` of step N must match the `sub` of step N+1), RFC 8693's `act` claim structure provides no mechanism to verify that consecutive delegation steps are actually connected.
3. **The STS faithfully propagates unverifiable history.** When the STS receives a `subject_token` that already contains nested `act` claims from prior delegation steps, it nests this history into the new token. The STS has no mechanism to verify the accuracy of this historical chain — it trusts whatever the incoming token asserts. If the incoming token was legitimately signed by a trusted upstream STS, the history is accepted without question.
4. **Intermediaries have access to multiple contexts.** In multi-agent systems, intermediary agents routinely participate in multiple delegation chains. A compromised intermediary that serves as an actor in both Alice's chain and Bob's chain has access to the token material needed to present mismatched inputs to the STS.

Anticipated Counterarguments

I expect three objections from specification authors. I address each below.

"The spec says nested `act` claims are informational only — this is an implementation bug, not a protocol flaw."

RFC 8693 §4.1 does state that prior actors are "informational only" for access control. However, the term "informational" is dangerously ambiguous in a security context. There is no such thing as truly informational identity data in practice:

- If it can be logged, it will be used in audits. If audited, its integrity matters.
- In regulated industries (healthcare under HIPAA, infrastructure under SOC 2), log integrity is a compliance requirement. A falsified delegation chain that assigns blame to the wrong party is a security failure regardless of whether it was used for "access control."
- Policy engines increasingly use delegation context for decisions that are functionally equivalent to access control: "was there a human in the loop?", "how many hops from the original principal?", "does this match a known-good delegation pattern?"

The specification creates what I term a **"spec-induced footgun"** — a feature that invites insecure usage by providing delegation chain data without integrity protection, while acknowledging that implementations will inevitably rely on that data.

"The security model assumes you trust the STS. If it's compromised, you have bigger problems."

This attack does not require a compromised STS. The STS behaves correctly — it validates both input tokens, confirms their signatures, and issues a properly-signed output token. The problem is that the STS is presented with **mismatched inputs** by a compromised intermediary (an agent, not

the STS itself). In multi-agent systems, intermediary agents that participate in multiple delegation chains are numerous and vary widely in trust level. Compromising one agent gives the attacker the ability to present cross-chain token pairs to a perfectly honest STS.

Furthermore, in distributed systems, delegation chains routinely span multiple STSs from different trust domains. No individual STS has a global view to validate the entire chain's history. Each STS trusts the signed token it receives from upstream — but the upstream token may already carry a spliced chain that was accepted by a different STS.

CVE-2025-55241 (Microsoft Entra ID) is instructive: the vulnerability existed because a backend service accepted actor tokens without validating their tenant origin. The "trust the STS" assumption failed at scale.

"The lack of per-step integrity is a feature — it allows flexibility, such as chain redaction for privacy."

If redaction is a required feature, it must be designed securely. Removing per-step integrity protection to enable redaction is trading integrity for convenience. A secure redaction mechanism would replace a step with a signed attestation of redaction, not silently create a discontinuous chain. The current design makes legitimate redaction and malicious splicing indistinguishable.

Conditions for Exploitability

The attack is exploitable when **all** of the following hold:

1. The system uses nested `act` claims to represent multi-hop delegation (not just single-level actor identification)
2. At least one intermediary participates in multiple delegation chains and is compromised, malicious, or coerced (realistic in multi-agent systems where agents serve multiple principals)
3. The intermediary can present mismatched `subject_token` / `actor_token` pairs to the token exchange endpoint — using a subject token from one chain and an actor credential obtained from another context (via observation, shared infrastructure, or direct self-authentication to a trusted STS)
4. The STS does not cross-validate that the `subject_token` and `actor_token` belong to the same delegation context (RFC 8693 does not require this)
5. The resource server or downstream services use the delegation chain history for any purpose beyond pure informational logging — including audit attribution, scope determination, or trust decisions

Condition 4 deserves emphasis: RFC 8693 says nested `act` claims are "informational only" for access control. But in practice, systems use delegation chain history for: - Audit trails ("who authorized this action?") - Compliance reporting ("was there a human in the loop?") - Trust scoring ("how many hops from the original principal?") - Anomaly detection ("is this a normal delegation pattern?")

If chain history is used for any of these purposes, chain splicing undermines it.

Mitigations

For Specification Authors

1. **Require cross-validation of token exchange inputs.** The STS should verify that the `subject_token` and `actor_token` belong to the same delegation context — for example, by checking that the `actor_token`'s subject was the intended next delegate of the `subject_token`'s current actor. This could leverage the `may_act` claim (RFC 8693 §4.4) as a binding mechanism: the `subject_token` declares who may act on its behalf, and the STS rejects exchanges where the `actor_token` identity is not listed.
2. **Define `aud`/`sub` chaining within `act` claims.** Require that nested `act` claims include `aud` and `sub` fields that form a verifiable chain: the `aud` of step N must match the `sub` of step N+1 (as OIDC-A specifies for `delegation_chain`, but applied to `act`). This would allow downstream relying parties to detect discontinuities introduced by splicing.
3. **Consider per-step signing or delegation receipts.** Each STS that performs a token exchange could include a signed "delegation receipt" — a compact JWS asserting "Agent-X delegated to Agent-Y at time T with scope S, authorized by STS-Z." This provides independently-verifiable provenance for each step rather than relying on the STS to faithfully propagate unverifiable history.

For Implementers

1. **Use `may_act` to restrict who can exchange a token.** Populate the `may_act` claim (RFC 8693 §4.4) in subject tokens to explicitly enumerate which actors are authorized to perform exchange. The STS should reject exchanges where the actor is not listed. Note: `may_act` validates the actor's *identity* but not that the `actor_token` was *acquired within the same delegation context*. It raises the bar but does not fully prevent splicing if the authorized actor is itself compromised.
2. **Validate delegation chains against an authoritative record.** Don't rely solely on the `act` claim in the token. Maintain a server-side ledger of delegation events and cross-reference incoming chains against it.
3. **Enforce maximum chain depth.** Limit nested `act` claims to a reasonable depth (e.g., 3-5 levels) to reduce the splicing surface.
4. **Use scope reduction as a cross-check.** Each delegation step should demonstrably reduce scope. A spliced chain that shows scope expansion through the chain is a red flag.
5. **Implement chain fingerprinting.** Hash the complete chain and register it at each STS. Downstream services can verify the chain fingerprint against the issuing STS.

Verification Guidance

A secure implementation should satisfy at least one of the following:

- **Not use nested `act` claims** for access control, audit attribution, or trust decisions (per RFC 8693 §4.1 guidance — though this is rarely practical)

- **Independently verify each delegation step** against an authoritative server-side record of delegation events
- **Reject chains that fail `aud`/`sub` continuity checks** (verifying that consecutive delegation steps are actually connected)
- **Enforce `may_act`** (§4.4) to restrict which actors can exchange a given subject token

---

Disclosure

This finding was identified during security research with the assistance of AI tools (Claude by Anthropic and Gemini by Google), which were used for collaborative analysis, adversarial review, and independent validation from the RFC 8693 specification text. I am sharing it with specification authors and the IETF OAuth Working Group for independent validation and consideration.

Date: 2026-02-26 **Author:** [Your name] **AI Assistance:** Claude (Anthropic) and Gemini (Google) were used for research, analysis, and independent validation

---

References

- RFC 8693 — OAuth 2.0 Token Exchange — §4.1 (act claim), §1.1 (delegation vs impersonation)
- CVE-2025-55241 — Microsoft Entra ID Actor Token Vulnerability — Real-world validation of act claim misuse
- OIDC-A 1.0 — OpenID Connect for Agents — delegation_chain with aud/sub chaining
- draft-oauth-ai-agents-on-behalf-of-user-02 — OBO flow producing act claims
- OWASP Top 10 for Agentic Applications 2026 — ASI03 (Identity and Privilege Abuse)