Memory Is Not Authority

CoreBond Research Series — CBRS-009
August 2026
Todd Kirton
DOI: 10.5281/zenodo.22242402

1. The Decision That Did Not End

A user signs in. Authentication succeeds. The system creates a session, issues a session secret, or allows an application to act with a token. Minutes later, another action is permitted without repeating the original authentication ceremony. Nothing is wrong with that sequence. It is one of the reasons sessions exist.

NIST describes a session secret as a secret issued by the verifier at authentication and used to establish continuity of authenticated sessions. Its session guidance then places boundaries on that continuity through session termination, timeout, and reauthentication requirements (National Institute of Standards and Technology [NIST], 2025).

The later action does not begin from zero. It relies on retained state produced by an earlier event. If the later decision changes because that state exists, is absent, has a particular value, or records a particular history, the state has become trust-relevant.

This is not a new theory of lifecycle management. Security systems already expire sessions, revoke tokens, preserve versions, retain logs, and destroy sensitive material. The narrower question is what proposition or security property a later decision is carrying forward. Storage lifetime tells us how long an artifact exists. It does not, by itself, tell us what that artifact is still entitled to establish.

Paper 7 used authority to ask who is entitled to establish or change a trust-relevant condition. The title here uses the word more narrowly: persistence alone does not grant retained state continuing decision weight. The question is why a later decision is still permitted to rely on what the system remembers.

2. Forgetting Can Be the Vulnerability

The obvious reaction is to minimize persistence. Less retained state can mean fewer stale artifacts and fewer things to protect. That intuition is useful until it becomes a general rule. Then it fails.

OAuth refresh-token rotation provides a clean counterexample. Current OAuth security guidance requires public clients to use sender-constrained refresh tokens or refresh-token rotation. Under rotation, the authorization server issues a new refresh token, invalidates the previous one, and retains information about their relationship. Reappearance of an invalidated token can therefore reveal replay and possible compromise (Lodderstedt et al., 2025). The security property depends partly on remembering that the old token existed and is no longer acceptable.

Software-update security reaches the same conclusion from another direction. The Update Framework (TUF) requires clients to persist trusted metadata and compare incoming versions against already trusted versions to detect rollback. It also uses expiration checks to detect potential freeze attacks (The Update Framework, 2026). A client that forgets newer trusted state can lose part of the basis for rejecting older state.

Certificate Transparency pushes the point further. Its append-only logs are not temporary clutter waiting for cleanup. RFC 9162 describes Certificate Transparency as using append-only logs so interested parties can detect certificate misissuance (Laurie et al., 2021). In that architecture, historical continuity is part of the security property.

Trustworthy systems therefore cannot treat statelessness as a universal ideal. Sometimes forgetting is the vulnerability.

3. Valid Is Not the Same as Applicable

If forgetting can weaken security, the opposite mistake is to assume retained state is safe to reuse as long as it remains technically valid. That fails because validity and applicability answer different questions.

NIST makes the distinction unusually clear for OAuth access tokens. It states that an access token allows an application to access services on a subscriber's behalf after authentication, but the relying party must not interpret the token's presence as evidence that the subscriber is currently present without other signals. NIST also notes that access and refresh tokens can remain valid long after the authentication session ends and the subscriber leaves the application (NIST, 2025).

Two propositions are involved. The token may still establish delegated authority to access a protected resource. It does not necessarily establish current human presence. The token has not failed. The attempted inference has.

TUF exposes the same distinction without a human user. Older metadata may remain correctly signed, yet a client that has already trusted a newer version can reject older metadata as a rollback attempt. Signature verification and present acceptability are related but distinct questions (The Update Framework, 2026).

This paper uses applicability for that second question: what fact, relationship, version, permission, or security property is the retained state being relied upon to carry forward, and has anything occurred that ends permission to rely on it for that purpose? Freshness may be part of the answer, but applicability is broader than a clock check.

The phrase inherited premise is shorthand for this situation, not a proposed standards term. A later decision may rely on retained state without recreating the event that produced it. If that state still passes its local validity rule but no longer supports the proposition being inferred from it, the system can be mechanically consistent and semantically wrong.

4. Time Is Only One Way State Becomes Wrong for Now

Expiration is attractive because it is simple. Give state a lifetime, reject it after the deadline, and the problem appears bounded. Many systems correctly use timeouts and expiration. The evidence does not support time as a universal invalidation rule.

State can be superseded before it expires. TUF can reject an older version because a newer trusted version is already known. A refresh token can be invalidated through rotation. A session can end through logout. A policy or authorization change can alter what a later decision is permitted to carry forward.

Distributed authorization makes the limitation of simple age especially visible. Google's Zanzibar system was designed so authorization decisions respect causal ordering of user actions amid changes to access-control lists and object contents (Pang et al., 2019). Its 'new enemy' problem shows why this matters: a causally prior removal can need to be visible before a later authorization decision. Recent state can still be wrong for the decision if it omits that prior change.

Time can therefore end applicability, but so can supersession, revocation, detected reuse, a causally prior policy change, termination of a relationship, a change in the proposition being inferred, or compromise of the basis on which the state was trusted. These are examples, not a universal taxonomy.

The operational question is: what is this state being relied upon to carry forward, and has anything happened that ends that reliance?

5. State Has Different Jobs

A universal retention rule also fails because retained state performs different security jobs.

Some state carries an earlier decision forward. Sessions are the obvious example. Other state remembers a negative event. A revoked credential, invalidated refresh token, replay record, or distrust decision may need to remain remembered specifically so old authority does not silently return.

Some state establishes ordering. Versions, sequence values, counters, and comparable records can tell a system that it has already moved past an earlier state. Forgetting them may reopen rollback or reuse. Historical state has another job entirely. Certificate Transparency retains history not because an old log entry continues to authorize current behavior, but because the record helps make prior issuance observable and auditable (Laurie et al., 2021).

Relationship state carries another meaning: a session, federation relationship, token family, ownership relationship, or pairing can exist over time and later end or change. Sensitive retained state creates a different pressure because its continued existence can increase compromise value even while it remains useful.

These are not proposed as a formal taxonomy. They show why 'keep it' and 'delete it' are both inadequate architectural rules. Historical memory may need to be durable. Replay memory must survive long enough to prevent reuse. A session should not silently become permanent. Sensitive material may need protection while useful and deliberate destruction when continued retention becomes the larger risk.

The architecture therefore needs more than a storage decision. It needs to know what security job the state is doing.

6. Moving State Changes the Problem

The word stateless can make this discussion slippery. An architecture can reduce server-side state by carrying claims in a signed token. That can genuinely solve particular storage, replication, or availability problems. It does not automatically eliminate every question about current state, revocation, or changed relationships.

OAuth token introspection illustrates the tradeoff. RFC 7662 defines an endpoint through which a protected resource can ask whether a token is currently active. It explicitly allows introspection responses to be cached for performance, but warns that caching reduces the liveness of authorization information and can create a window in which a revoked token is still accepted from stale cached state (Richer, 2015).

Self-contained validation and current-state lookup therefore solve different problems. A signed token can carry integrity-protected claims and permit local validation. Revocation, logout, relationship changes, or other dynamic conditions may still require some way to make the changed world visible.

Refresh-token rotation makes the same point from another direction: the old token is invalidated while relationship information is retained to detect replay (Lodderstedt et al., 2025). Externalization can remove real local state and improve an architecture. The narrower point is that moving information changes which component carries the premise and how changes become visible; it does not by itself prove that later decisions have no state problem left to reason about.

7. When Remembering Becomes Exposure

Necessary memory is not a license to keep everything.

IETF privacy guidance identifies correlation as a threat and notes that persistent or infrequently replaced identifiers can facilitate tracking across time. It also explains that pseudonymity is strengthened when the same pseudonym is used less often and across fewer contexts, and identifies pseudonym lifetime and replaceability as design considerations (Cooper et al., 2013). A persistent identifier can remain accurate and non-secret while creating greater linkability because it remains stable.

Sensitive retained material presents a different risk. NIST requires credential service providers to maintain records of bound authenticators and significant lifecycle events, yet it also requires prompt suspension, invalidation, or destruction of compromised authenticators and risk-based decisions about optional records retention (NIST, 2025). Retention and invalidation are therefore both legitimate security operations, depending on the state and the purpose it serves.

Sometimes the correct lifecycle action is retention. Sometimes it is expiration, rotation, revocation, unlinkability, sanitization, or destruction. Those operations solve different problems and are not interchangeable.

The relevant question is whether continued retention still serves the security purpose for which the state is being kept, and whether the risks created by continued persistence remain acceptable for that purpose.

8. The Architecture Should Be Able to Explain Why

Existing security architecture did not forget how state works. The examples in this paper are useful precisely because mature systems already manage state deliberately.

NIST bounds authenticated sessions and separates access-token validity from subscriber presence. OAuth retains refresh-token relationships to detect reuse and provides introspection for current token state. TUF persists trusted metadata to resist rollback. Certificate Transparency preserves append-only history. Zanzibar treats

causal ordering of authorization state as security-relevant. Privacy guidance treats identifier persistence and linkability as design concerns.

The synthesis is not that these mechanisms are missing. It is that they expose the same reasoning boundary from different directions.

When retained state influences a later trust decision, the architecture or policy model should make the basis for that reliance explainable. What fact, relationship, version, permission, or security property is being carried forward? Where did the state come from? What ordering or context matters? What event would end permission to rely on it?

This does not require every request to recompute or log a full justification, nor does it require a universal schema, TTL, scoring formula, or protocol field. The requirement is conceptual before mechanical: the design should not confuse persistence or local validity with permanent permission to reuse state for whatever proposition a later decision happens to infer.

Once retained state changes a later trust decision, it is part of that decision's reasoning whether the architecture acknowledges that fact or not.

9. Memory Is Not Authority

Trustworthy systems cannot safely forget everything. Some security properties exist only because the system remembers prior use, newer versions, revoked authority, relationships, or history. A clean slate can be a security failure.

Memory has an opposite failure mode. A surviving session, token, signed record, policy snapshot, identifier, version, or historical fact does not acquire permanent decision weight merely because it remains intact. Technical validity answers one question. Current applicability answers another.

The practical distinction is justified persistence. Retain state when memory serves a security property. Preserve history when history is the property. Remember distrust when forgetting would resurrect old authority. But do not allow later decisions to rely on retained state merely because the state still exists or still passes its local validity rule.

The architecture should be able to answer the harder question: what is this remembered state still legitimate to establish here?

Eventually state becomes missing, corrupted, superseded, invalidated, or intentionally destroyed. At that point the problem changes. The system must decide what can still be carried forward, what must be distrusted, and what has to be established again.

Paper 10 asks how trust recovers when memory is no longer enough.

References

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., & Smith, R. (2013). Privacy considerations for Internet protocols (RFC 6973). Internet Engineering Task Force. https://doi.org/10.17487/RFC6973

Laurie, B., Messeri, E., & Stradling, R. (2021). Certificate Transparency Version 2.0 (RFC 9162). Internet Engineering Task Force. https://doi.org/10.17487/RFC9162

Lodderstedt, T., Bradley, J., Labunets, A., & Fett, D. (2025). Best Current Practice for OAuth 2.0 Security (RFC 9700). Internet Engineering Task Force. https://doi.org/10.17487/RFC9700

National Institute of Standards and Technology. (2025). Digital identity guidelines: Authentication and authenticator management (NIST Special Publication 800-63B-4). https://doi.org/10.6028/NIST.SP.800-63B-4

Pang, R., Caceres, R., Burrows, M., Chen, Z., Dave, P., Germer, N., Golynski, A., Graney, K., Kang, N., Kissner, L., Korn, J. L., Parmar, A., Richards, C. D., & Wang, M. (2019). Zanzibar: Google's consistent, global authorization system. In 2019 USENIX Annual Technical Conference (USENIX ATC 19) (pp. 33–46). USENIX Association. https://www.usenix.org/conference/atc19/presentation/pang

Richer, J. (2015). OAuth 2.0 token introspection (RFC 7662). Internet Engineering Task Force. https://doi.org/10.17487/RFC7662

The Update Framework. (2026). The Update Framework specification (Version 1.0.35). https://theupdateframework.github.io/specification/v1.0.35/