Authority Propagation Models: PoP vs PoC and the Confused Deputy Problem

Nicola Gallo

1 December 2025

1 Model

We formalize the PIC (Provenance Identity Continuity) Model as follows.

Let P be a finite set of principals, O a set of operations, and R a set of resources. Define a privilege as $(o, r) \in O \times R$.

Execution is modeled as a causal chain of hops:

$$\pi = \langle (p_0, ops_0), (p_1, ops_1), \ldots, (p_n, ops_n) \rangle$$

where p_0 is the originator and each $ops_{i+1} \subseteq ops_i$ (monotonicity). Here $ops_i \subseteq O \times R$ denotes the set of privileges that principal p_i may exercise at hop i.

Definition 1 (PIC Model)**.** *A system enforces Provenance Identity Continuity if, for every execution chain π, the set of privileges at the final hop is bounded by the privileges at the origin:*

$$ops_n \subseteq ops_0$$

and every privilege exercised at hop n is causally linked to the origin via a verifiable chain.

Theorem 1 (PIC Safety)**.** *If the PIC Model holds, no principal can exercise a privilege not present at the origin.*

Proof. By construction, $ops_{i+1} \subseteq ops_i$ for all i, so $ops_n \subseteq ops_0$. Thus, any $(o, r) \in ops_n$ must also be in ops_0. Therefore, no privilege can be gained beyond the origin.

Each principal p has an associated privilege set:

$$Priv(p) \subseteq O \times R.$$

A request is a message req sent between principals and may contain a payload.

2 Confused Deputy

Definition 2 (Confused Deputy)**.** *A confused deputy occurs when there exist principals U (user) and D (deputy) such that:*

1. $(o, r) \notin Priv(U)$,
2. $(o, r) \in Priv(D)$,
3. *U sends a request req to D,*
4. *as a consequence of req, D executes (o, r).*

This definition does not depend on implementation details, only on the mismatch of authority and causality.

Classical example (Hardy, 1988). A FORTRAN compiler `FORT`, installed in the privileged directory `SYSX`, holds ambient authority to write statistics to `(SYSX)STAT`. A user invokes the compiler and provides `(SYSX)BILL` as the debugging output filename. The compiler opens the target for output using its own authority over `SYSX`, thereby overwriting `(SYSX)BILL`. The user never possessed this privilege, but the deputy exercised it on the user's behalf.

3 Proof-of-Possession (PoP)

A token t grants a set of privileges:

$$Auth(t) \subseteq O \times R.$$

PoP Semantics. Possession implies usability: if a principal holds t, it may exercise all $(o, r) \in Auth(t)$.

PoP systems do not constrain authority by causality or provenance.

3.1 Vulnerability Condition

Assume:

$$(\text{write}, r) \notin \mathit{Priv}(U) \tag{H1}$$

$$(\text{write}, r) \in \mathit{Priv}(\mathit{COMP}) \tag{H2}$$

Here U is the user and COMP is a compiler-like service acting as deputy. Further assume:

> The payload of a request may influence control flow in COMP, including code paths that open a resource r using COMP's own authority.

Theorem 2 (PoP admits confused deputy)**.** *Under assumptions (H1)–(H2), there exists a request req such that COMP executes (write, r) as a result of processing req.*

Proof. Since $(\text{write}, r) \in \mathit{Priv}(\mathit{COMP})$, an internal code path may open r under COMP's ambient authority. Because user input influences that path, there exists a malicious payload req that triggers it. Thus COMP executes a privilege the user does not possess.

This applies to all artifact-based delegation models: JWTs, sealed capabilities, cryptographic bearer credentials, etc.

4 Proof-of-Continuity (PoC / PIC)

Execution is a causal chain:

$$p_0 \to p_1 \to \cdots \to p_n,$$

with $p_0 = U$.

Each hop transfers:

$$ops_{i+1} \subseteq ops_i. \tag{C1}$$

Let:

$$\pi = \langle (p_0, ops_0), (p_1, ops_1), \ldots, (p_n, ops_n) \rangle.$$

4.1 Authorization Rule

(o, r) is authorized at the final hop iff:

$$(o, r) \in \bigcap_{i=0}^{n} ops_i.$$

Since the chain is monotonic decreasing:

$$\bigcap_{i=0}^{n} ops_i = ops_n.$$

Thus no hop may acquire authority not already present at the origin.

Lemma 1 (Irreversible loss of authority)**.** *If $(o, r) \notin ops_j$ for some hop j, then $(o, r) \notin ops_k$ for all $k \geq j$.*

Proof. By monotonicity, $ops_k \subseteq ops_j$ for all $k \geq j$.

5 Safety Property

Definition 3 (Origin-bounded authority)**.** *A model enforces origin-bounded authority if every executable privilege at hop n was originally granted at hop 0.*

Theorem 3 (PoC prevents confused deputy)**.** *If $ops_0 = \{(convert, r)\}$, then $(write, r)$ can never be authorized.*

Proof. $(\text{write}, r) \notin ops_0$ and $ops_{i+1} \subseteq ops_i$. Thus $(\text{write}, r) \notin ops_n$.

6 Elimination of the Confused Deputy

The Confused Deputy requires an ontological assumption: that authority is a transferable or rebindable object that a principal may hold or reinterpret on behalf of another.

PIC eliminates this ontology entirely.

Theorem 4 (Confused Deputy is non-formulable under PIC)**.** *In any system satisfying PIC monotonicity $ops_{i+1} \subseteq ops_i$ and origin-bounded authority $ops_n \subseteq ops_0$, the Confused Deputy conditions cannot be jointly satisfied.*

Proof. The Confused Deputy requires:

$$(o, r) \notin ops_0 \quad \text{and} \quad (o, r) \in ops_k \text{ for some } k > 0.$$

But PIC enforces:

$$ops_k \subseteq ops_0 \ \ \forall k.$$

Thus if $(o, r) \notin ops_0$, it cannot appear in any ops_k. The privilege mismatch required for a Confused Deputy is impossible.

Interpretation. Authority in PIC is not something that a subject *has*; it is a continuity property of the execution chain. Since no hop may introduce or reinterpret authority absent at the origin, no deputy can be "confused." The attack does not occur; it is not even definable in the model.

7 Discussion

PoP systems conflate authority and possession, enabling confused deputy attacks. PIC/PoC systems propagate only non-expansive subsets of authority, ensuring no downstream service can perform operations not authorized at the origin.

Related Work

The confused deputy was formalized by Hardy (1988). Capability theory originates in Dennis & Van Horn (1966). Confinement was introduced by Lampson (1973). EROS (Shapiro et al., 1999) and seL4 (Klein et al., 2009) provide strong isolation. Distributed authorization theories include Abadi et al. (1993–2000) and SPKI (RFC 2693). Modern zero-trust models (BeyondCorp, SPIFFE) apply causal constraints similar in spirit.

Acknowledgments

The author used automated language assistance tools for grammar and phrasing. All conceptual contributions, models, and proofs are solely the author's.

References

1. N. Gallo. "PIC Model — Provenance Identity Continuity for Distributed Execution Systems." Zenodo (2025). `https://zenodo.org/records/17777421`.

2. N. Hardy. "The Confused Deputy." Operating Systems Review, 1988.

3. Dennis & Van Horn. "Programming Semantics for Multiprogrammed Computations." CACM, 1966.

4. B. Lampson. "A Note on the Confinement Problem." CACM, 1973.

5. Shapiro et al. "EROS: A Fast Capability System." SOSP, 1999.

6. Klein et al. "seL4: Formal Verification of an OS Kernel." SOSP, 2009.

7. Abadi et al. "A Calculus for Access Control." TOPLAS, 1993–2000.

8. C. Ellison et al. "SPKI Certificate Theory." RFC 2693, IETF, 1999.

9. Google. "BeyondCorp: A New Approach to Enterprise Security." 2014.

10. SPIFFE Working Group. "Secure Production Identity Framework for Everyone."