# 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:

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

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 $\pi$, 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`, the system billing file, as the name of the debugging output. 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 behalf of the user. The failure arises from ambient authority and lack of provenance.

## 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:

$$(\text{write}, r) \notin Priv(U) \tag{H1}$$
$$(\text{write}, r) \in Priv(COMP) \tag{H2}$$

Here $U$ is the user and $COMP$ is a compiler-like service acting as deputy, analogous to Hardy's original example.

Further assume:

> The payload of a request (e.g., an output file name supplied by the user) may influence control flow in $COMP$, including code paths that open a resource $r$ for output 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 $(\text{write}, r) \in Priv(COMP)$ (H2), an internal code path in $COMP$ may open $r$ for output using its own ambient authority. Because request processing is influenced by user-supplied parameters (such as the target file name), there exists an adversarial payload $req$ that causes $COMP$ to select $r$ and execute $(\text{write}, r)$. By (H1), the user $U$ does not possess $(\text{write}, r)$, yet by sending $req$ to the deputy $COMP$, $U$ causes $COMP$ to exercise this privilege on $U$'s behalf. All conditions of the Confused Deputy definition are therefore satisfied. □

This applies to OAuth bearer tokens and sealed capabilities: sealing protects transport, not authority semantics.

# 4 Proof-of-Continuity (PoC / PIC)

Execution is modeled as a causal chain of hops:

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

with $p_0 = U$.

Each hop transfers a privilege subset:

$$ops_i \subseteq O \times R$$

and must satisfy:

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

Let $\pi$ be a verifiable sequence:

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

### 4.1 Authorization Rule

The final service authorizes $(o, r)$ if and only if:

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

and $\pi$ is valid.

Since the chain is monotonic decreasing,

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

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

## 5 Safety Property

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

**Theorem 3** (PoC prevents confused deputy)**.** *If $ops_0 = \{(convert, r)\}$ and (C1) holds for every hop, then $(write, r)$ can never be authorized at hop $n$.*

*Proof.* Authorization requires:

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

But $(\text{write}, r) \notin ops_0$ and each $ops_{i+1} \subseteq ops_i$. Therefore $(\text{write}, r) \notin ops_i$ for all $i$, hence not in the intersection. The request is rejected.

## 6 Discussion

PoP systems conflate authority and possession, enabling confused deputy attacks whenever privileged internal code paths exist. PIC/PoC systems propagate only non-expansive subsets of authority, ensuring that downstream services cannot perform operations not explicitly authorized at the origin.

## Related Work

The confused deputy was originally formalized by Hardy (1988), showing how ambient authority enables unintended privilege escalation. Earlier foundations of capability systems date back to Dennis and Van Horn (1966), while confinement and controlled execution were explored by Lampson (1973). Modern capability systems such as EROS (Shapiro et al., 1999) and seL4 (Klein et al., 2009) provide strong isolation properties and monotonic privilege enforcement. The formal theory of distributed authorization and delegation traces back to Abadi et al. (1993–2000) and the SPKI certificate model (RFC 2693). Recent identity systems such as BeyondCorp and SPIFFE apply similar causal principles to zero-trust environments.

## Acknowledgments

The author used automated language assistance tools, including large language models, to help with grammar, wording, and formal phrasing. All ideas, models, proofs, and conclusions in this document are solely the responsibility of the author.

## 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 (or why capabilities might have been better than addresses)." *Operating Systems Review*, 22(4), 1988.

3. J. B. Dennis and E. C. Van Horn. "Programming Semantics for Multiprogrammed Computations." *Communications of the ACM*, 9(3), 1966.

4. B. W. Lampson. "A Note on the Confinement Problem." *Communications of the ACM*, 16(10), 1973.

5. J. S. Shapiro, J. M. Smith, and D. J. Farber. "EROS: A Fast Capability System." *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, 1999.

6. G. Klein et al. “seL4: Formal Verification of an OS Kernel.” *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

7. M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. “A Calculus for Access Control in Distributed Systems.” *ACM Transactions on Programming Languages and Systems*, 1993–2000.

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

9. Google. “BeyondCorp: A New Approach to Enterprise Security.” Google Whitepaper, 2014.

10. SPIFFE Working Group. “Secure Production Identity Framework for Everyone (SPIFFE).”