

Need-to-Know vs. Least Privilege: Two Different "No"s

Mon, Sep 7, 6:00 PM EDT · <https://scottslab.io/posts/need-to-know-and-least-privilege>

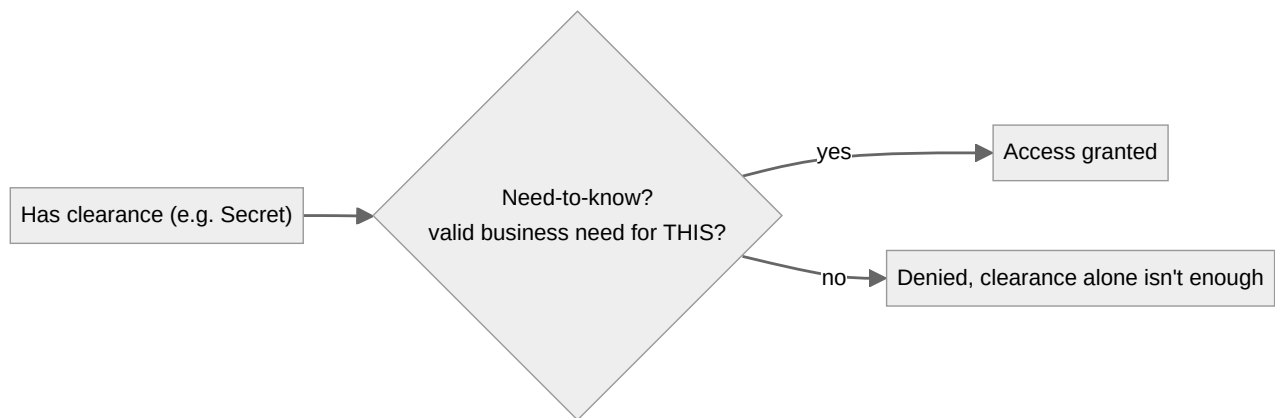


TL;DR

They sound alike but they're different gates. Least privilege is about the permissions your *role* needs; need-to-know is stricter and case-by-case, gating each specific thing regardless of clearance. Conflating them produces two predictable failure modes, and the same failure mode is what lets privilege creep accumulate unnoticed through every role change an employee ever has. Access recertification is the control that's supposed to catch it. Most implementations rubber-stamp instead. Separation of duties is the backstop for the specific transactions where a single over-privileged account is a fraud risk, not just an exposure risk.

People use "least privilege" and "need-to-know" interchangeably, and they're two different controls doing two different jobs. Least privilege is about role and permissions: give a person the minimum access their job function requires, nothing "just in case." NIST's framing of it (SP 800-53, control AC-6) is blunt: "employ the principle of least privilege, allowing only authorized accesses for users... that are necessary to accomplish assigned organizational tasks." Need-to-know is stricter and situational: even holding the right clearance, access isn't automatic. It's granted case-by-case, only for a valid business need, right now. Clearance says you *could* be trusted with Secret material; need-to-know says you only see the Secret documents relevant to your actual task. That's why it's

the default in classified environments, where "I have the clearance" is explicitly not the same as "I get to look."



Need-to-Know vs. Least Privilege: Two Different "No"s

Where conflating the two actually breaks a design

The failure isn't academic; it shows up as a specific shape of over-exposure. A system built on least privilege alone tends to grant access at the role level and stop there: an HR analyst gets "HR system access," scoped correctly to the HR function, and that's treated as done. But the role doesn't need every employee's record for every task; it needs *this* employee's record for *this* ticket. Least privilege was satisfied and the design is still wrong, because nothing gates the individual record. That's the gap need-to-know closes: least privilege sets the outer boundary of what a role could ever touch, and need-to-know narrows it to what's relevant right now.

Run the failure the other way: pure need-to-know with no role-based floor. Every access becomes a one-off grant argued from scratch, turning the approval queue into the bottleneck for ordinary work. The working answer is layered, not either/or: least privilege sets the role boundary, and need-to-know gates only the data that actually deserves the extra friction: the individual record, the specific customer's PII, the specific case file.

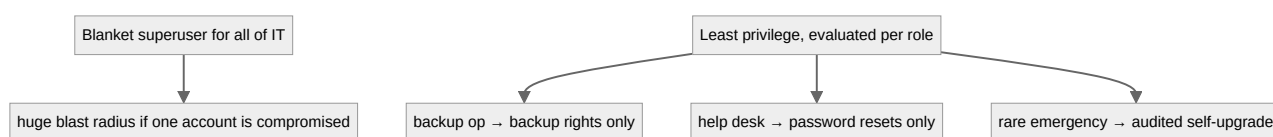
Privilege creep is what happens when nobody revisits the grant

Privilege creep isn't usually one bad decision; it's the sum of many reasonable ones nobody revisited. Someone moves from support to engineering and keeps their support tooling access because nobody remembered to remove it. A person covers a colleague's project for a quarter and stays on the shared drive two years after it ends. A contractor converts to full-time and simply inherits both the contractor group and the employee group, since merging them cleanly was more work than adding the new one. None of these are attacks. Each is a small, defensible exception. But the account that results from stacking a career's worth of them is the one an attacker wants to land on: broad, quiet, never audited as a whole because no single grant looks wrong alone.

Recertification that works vs. recertification that's theater

Access recertification exists to catch exactly that accumulation, and most implementations of it don't work. The failure pattern is familiar: a manager gets a screen listing forty permissions for their team, one "approve all" button, and no context on what any entry grants. Nobody can judge blast radius from a permission name, so the path of least resistance is to click through, and the review becomes a compliance artifact, not a control.

What actually works differs in three ways. The reviewer understands the permission: often the system or data owner, not the line manager, since a manager can vouch for the person but not the access. The default is removal, not confirmation: the reviewer has to affirmatively justify *keeping* an entitlement, because silence should mean no. And the justification recorded at grant time travels with the entitlement, so the reviewer checks whether the original reason still holds instead of reconstructing it from nothing.



Recertification that works vs. recertification that's theater

Separation of duties: the control for the transactions that matter most

Separation of duties is a narrower, sharper tool than either of the above. NIST's framing (SP 800-53, AC-5) describes it as addressing "the potential for abuse of authorized privileges" and reducing "the risk of malevolent activity without collusion." The point isn't that one person is untrustworthy. It's that no single person should be able to complete a sensitive transaction alone.

The transactions that need it share a shape: creation and approval of the same thing by the same person. The classic case is accounts payable: the person who creates or edits a vendor record shouldn't also approve payment to that vendor, since together those two permissions let one person invent a vendor and pay it. Payroll has the same shape (whoever processes a compensation change shouldn't approve their own raise), and so does a developer merging their own change straight to production with no second reviewer, even when nobody calls it a separation-of-duties gap. NIST names one more directly: the people who administer access control shouldn't also administer the audit logs. The person who can grant themselves access shouldn't be the person who can erase the evidence of it.

The sysadmin problem, and the valve that makes it survivable

This bites hardest exactly where it's most tempting to skip: sysadmins and IT staff, where the job's whole point is broad power, and the lazy default is blanket superuser access for the entire team:

precisely the account an attacker dreams of landing on. The discipline is to evaluate per role, not per department: a backup operator needs backup rights, not domain admin; a help-desk tech needs password-reset rights, not the keys to the directory. Pure least privilege taken to its extreme grinds operations to a halt, so the honest target is "as little as works," not "as little as imaginable."

The pressure-release valve for that tension is a controlled emergency-access path: instead of standing broad access "in case something breaks at 3am," people get a self-upgrade procedure for the real emergency, logged and reviewed afterward, so the everyday state stays locked down without anyone having been over-privileged the whole time waiting for a crisis that might never come. Vaulting the credential, brokering the elevation, and auditing every use of it is its own topic, the one privileged access management tooling exists to solve.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — <https://scottslab.io/posts/need-to-know-and-least-privilege>