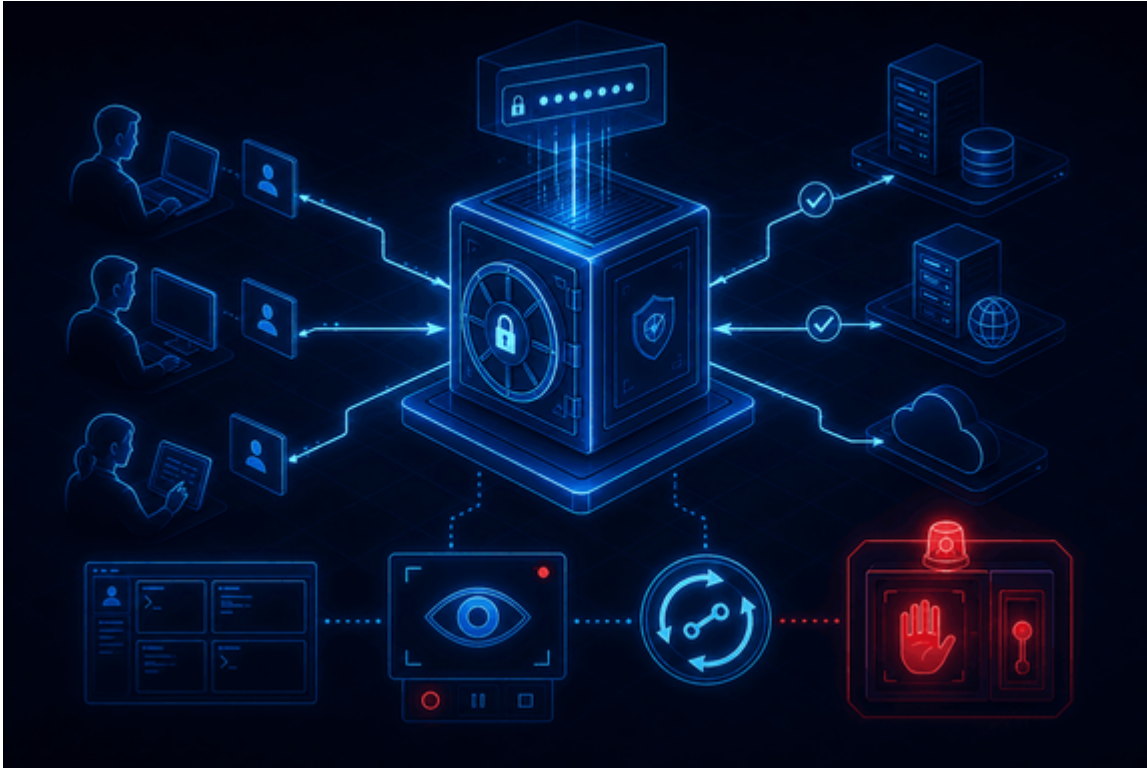


Privileged Access Management: Taking the Keys Out of People's Hands

Wed, Sep 9, 10:00 AM EDT · <https://scottslab.io/posts/privileged-access-management-pam>



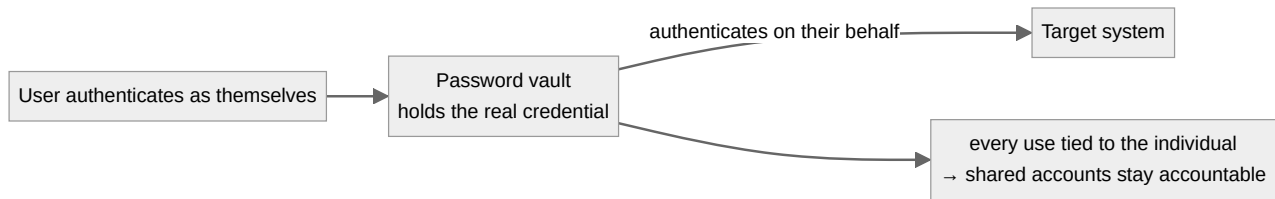
TL;DR

PAM exists because standing privileged access is the highest-risk thing in most environments, and the toolset works by taking credentials out of human hands: vaulting, automated rotation, session recording and brokering, and just-in-time elevation that turns "always admin" into "admin for the next 30 minutes, logged." Break-glass accounts cover the genuine emergency, heavily monitored so they can't quietly become standing access. Service accounts are the part nobody has a clean answer for. And most PAM programs fail for the same reason: they make people's jobs slower, so people route around them.

Privileged accounts, the admin and root credentials that can override everything, are the single biggest unmanaged risk in most organizations. NIST's glossary defines one plainly, as "a system account with the authorizations of a privileged user," and its least-privilege control goes further, calling for privileged accounts to be restricted "to organization-defined personnel or roles" rather than handed out by default. PAM is the tooling built to enforce that restriction at scale, by taking the credentials out of humans' hands entirely.

The foundation is password vaulting: an encrypted repository holds the privileged credentials, and no user ever knows the actual password. When you need a privileged account, the vault

authenticates on your behalf. You prove who *you* are to the vault, and it uses the secret you never see. That solves the oldest accountability problem in IT: shared admin accounts. Normally "admin did it" is useless because five people know the password; with a vault, every checkout is tied back to the individual who requested it, so even a shared account stays fully accountable.



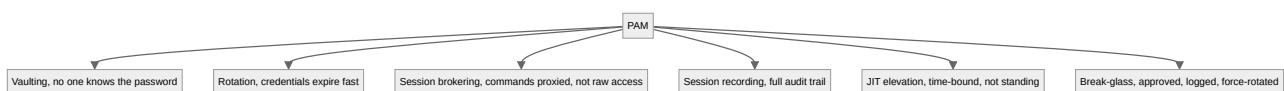
Privileged Access Management: Taking the Keys Out of People's Hands

Rotation, and why it has to be automated

Vaulting alone still leaves a static secret sitting there. Automated rotation is what keeps it from going stale: the vault changes the credential on a schedule, or immediately after every checkout, without a human doing it by hand. It has to be automated rather than a quarterly ritual because a leaked or reused credential is only dangerous for as long as it stays valid; rotation shrinks that window from "however long until someone remembers" to "one session." The trade-off is real, and it's why rotation projects break things: anything hardcoded against the old password (a script, a scheduled task, a monitoring agent) fails the moment the vault rotates it out from under that integration.

Brokering the session, not just the password

On top of the vault, PAM adds a layer that matters just as much: it sits *in the middle* of the session rather than handing over the raw credential at all. Command proxying and session brokering mean the PAM system receives your commands, checks them against policy, and issues them to the target, so it can allow, deny, or record at the command level, not just at login. The user and the target system never actually share a secret directly; the broker is the only thing that does. Full session recording rides along with this, often a literal video-like capture of everything done under privilege: exactly what you want handed to an auditor or an incident responder instead of someone's recollection of what happened.



Brokering the session, not just the password

Just-in-time: turning "always admin" into "admin until 2:30"

Standing privilege is the thing most of this is trying to eliminate, and just-in-time elevation is the direct answer to it. Instead of holding an admin role permanently, a person requests it, gets approved (often automatically against policy, sometimes requiring a second person to sign off), and receives access bound to a start and end time: commonly enforced with multi-factor authentication on activation and a required justification logged with the grant. When the window closes, the privilege is gone, not just unused: an account that's admin for thirty minutes a week is thirty minutes a week of attack surface, not all year. The operational cost is real, too. Every elevation request adds friction, and if the approval step is slow, people learn to request access well before they need it, which quietly turns "just-in-time" back into "standing" with extra paperwork.

Break-glass: the emergency path, kept honest

PAM formalizes the emergency-access workflow that standing least privilege needs as its escape hatch. These are accounts and procedures used only when the normal path is unavailable: an identity provider outage, a lockout, a genuine "the building is on fire and we need root now" situation. Good practice treats these as different from everyday privileged accounts in kind, not just in name: kept separate from the normal account lifecycle so they don't get caught up in routine offboarding or sync jobs, credentials stored so no single person controls them alone, and every sign-in alerted on and reviewed, because a break-glass account used quietly with no follow-up has stopped being an emergency control and become an unmonitored backdoor. The workflow forces a password change immediately after use, so the elevated access can't linger. It's "break glass" done right: available, but visible enough that nobody reaches for it casually.

Service accounts are the part nobody has fully solved

Everything above assumes a human at the keyboard, and service accounts break that assumption. They're non-interactive, so MFA on activation and a live session to record don't map cleanly onto a script or a scheduled job. They tend to be shared across systems, embedded in config files and connection strings, provisioned once by whoever needed them working by Friday. So the first real challenge is just finding them all; an inventory nobody trusts is an inventory that isn't one. Rotating them still matters, but rotation needs an owner who can verify the dependent system survives the change, or "more secure" turns into "outage." The one control that transfers cleanly is anomaly detection: a service account authenticating interactively, from a new location, or outside its normal pattern, is one of the highest-signal indicators available, precisely because it should never happen.

Why PAM projects fail

Most PAM rollouts fail the same way, and it's rarely the technology: they make privileged work slower, so people route around them. An admin who used to type a password now checks out a

credential through a portal, waits on an approval, and works inside a recorded, sometimes-laggy proxied session. During an actual incident, when speed matters most, that admin reaches for the local admin account or the shared password still stashed in a wiki instead. Onboarding is the other failure mode: bringing every legacy system, network device, and forgotten service account into the vault is a large, unglamorous migration, and it's common for a program to vault the easy majority and stall on the rest, leaving two paths into production, the sanctioned brokered one and the old direct one, which quietly defeats the investment. The programs that hold up are built with the operations team rather than imposed on them, and honest that a control nobody can tolerate using is a control that gets bypassed.

The honest bottom line: privileged users are a serious risk if they go unmanaged, and PAM absorbs most of the operational burden of managing them: vaulting, rotation, brokering, controlled elevation that no team wants to do by hand. It doesn't eliminate the risk; nothing does. It's a clear case of a broader pattern: layered controls reduce exposure even when no single control closes every door, and the layer only holds if people can still get their job done through it.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — <https://scottslab.io/posts/privileged-access-management-pam>