The evolution of Privileged Access Management

Why PAM matters more than ever

Privileged Access Management (PAM) was born from a simple operational need: to manage and rotate shared admin passwords on servers. But today's infrastructure looks nothing like it did twenty years ago. We now have ephemeral microservices, automated CI/CD pipelines, and production systems spanning cloud consoles, Kubernetes clusters, and APIs. This explosion in complexity—and the increasing number of identities, both human and machine—has outgrown the assumptions of legacy PAM.

What was once a vault for root passwords must now be an orchestration layer for short-lived, least-privileged access across hybrid and cloud-native environments. PAM is no longer just about compliance—it's critical to reducing access risk in production systems.

First, what is PAM?

Privileged Access Management (PAM) is how organizations control access to sensitive production systems—like servers, databases, cloud consoles, and Kubernetes clusters. These systems carry high risk: a single security incident or outage can cause significant business disruption. PAM ensures that only the right identities—human or machine—get access, only for what they need to do, only when they need it, and always under audit.

It addresses two control layers:

- **Authentication (authn): who** is requesting access
- **Authorization (authz): what** they're allowed to do

Earlier solutions focused on Privileged Account Management—securing high-risk accounts and their credentials, like root or admin passwords. These tools emphasized storing, rotating, and tracking use of privileged accounts. Privileged Access Management is broader: once the entity has verified their identity with a privileged account, what access do they receive? PAM manages the full lifecycle of access—who can reach sensitive systems, when, and with what permissions.

The constants and the evolution

While infrastructure has evolved—from static on-prem servers to dynamic, cloud-native services—the core goals of PAM have remained the same:

- **Short-lived access** (across authn and authz)
- **Least privilege**
- **Auditability**

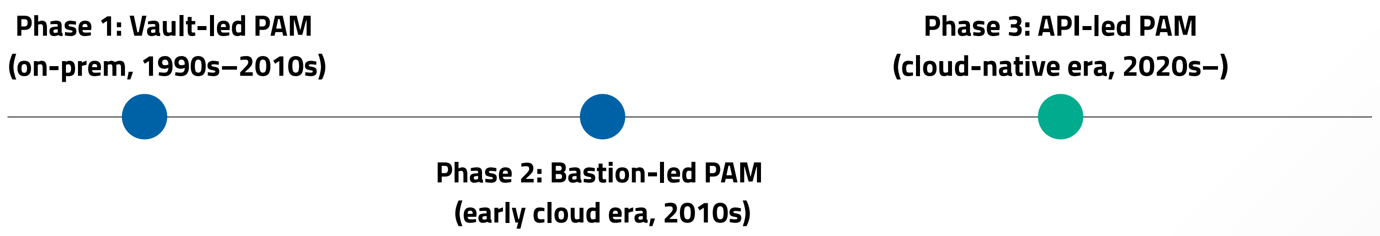
What **has** changed are the assumptions: how identities are provisioned, what counts as a production system, and how infrastructure behaves. Each shift introduced new risks—and new patterns for implementing PAM.

Contents

What got us here

Each phase in the evolution of PAM emerged to address new business, security and governance challenges. In the following pages, we'll explore these phases: vault-led PAM (1990s–2010s), bastion-led PAM (early cloud era, 2010s), and API-led PAM (cloud-native era, 2020s-).

We'll conclude with an explanation of what today's modern PAM solutions must do to adapt to hybrid and cloud-native environments, as well as widespread use of AI and non-human identities.


Phase 1: Vault-led PAM (on-prem, 1990s–2010s)
Phase 2: Bastion-led PAM (early cloud era, 2010s)
Phase 3: API-led PAM (cloud-native era, 2020s–)

Phase 1: Vault-led PAM

(on-prem, 1990s–2010s)

PAM began in the late 1990s, when enterprise infrastructure was mostly static. Vendors like CyberArk pioneered vault-based tools to manage administrator access to servers, network devices, and databases inside corporate data centers.

Authentication was handled via vaulted credentials: admins checked out root passwords or SSH keys, which were automatically rotated after use. This created short-lived authentication by design.

Authorization was straightforward: keys mapped to specific systems, and access was granted per system or role. Least privilege was relatively easy to enforce—access was deterministic and scoped to a known set of assets.

Auditability came from credential logs and session recordings. Vaults tracked who accessed which credentials and when. These patterns aligned with compliance needs such as SOX and PCI-DSS.

The on-prem environment—fixed IPs, static servers, and centralized perimeters—fit this model well. In this phase, Privileged Account Management and Privileged Access Management were often interchangeable.

Why this broke: As infrastructure grew, so did the number of credentials, systems, and users. Manual rotation and secret sprawl became operational bottlenecks. PAM remained effective—but increasingly fragile and slow-moving.

Phase 2: Bastion-led PAM

(early cloud era, 2010s)

In the early 2010s, enterprises began moving production workloads to public cloud platforms like AWS and Azure. The systems needing privileged access—VMs, databases, and storage—remained familiar, but the environment became ephemeral and elastic. IPs were dynamic, and servers could be spun up and shut down in minutes.

Phase 1 vault-based PAM solutions struggled to keep up. The sheer number of hosts and speed of change made managing static credentials impractical. Admins either reused shared secrets or skipped vaults entirely for operational speed.

Authentication still relied on SSH keys and static credentials. SSO adoption had started but wasn't yet widespread. Authorization challenges began to emerge. Most teams lacked tools to assign fine-grained, temporary access. As a result, users were granted standing access to VMs—broad permissions that persisted indefinitely.

Why did authorization lag? Because infrastructure was changing too quickly for static access lists to keep up. The scale, ephemerality, and manual provisioning meant that once access was granted, it was rarely revoked. Enforcing least privilege became difficult in practice.

To compensate, teams introduced bastion hosts—jump servers at the edge of cloud networks that funneled admin traffic. Engineers logged into the bastion (typically via SSH key or enterprise SSO), then into the target systems. Some organizations adopted Teleport, an identity-aware proxy that authenticated users via SSO and granted access based on role and context.

These solutions improved control, but came with trade-offs:

- Vaults were still used, but secret sprawl was common.
- Bastions introduced friction: multi-hop workflows and brittle configurations.
- Authorization remained coarse: access often implied full admin privileges within large scopes.

Why this broke: Bastions didn't address the core issue—persistent entitlements in dynamic environments. Developers started bypassing controls. PAM solutions became less effective.

Phase 3: Next-gen, API-led PAM
(cloud-native era, 2020s–)

By the 2020s, infrastructure had transformed again. Organizations were fully cloud-native. Production systems were no longer just VMs—they included cloud consoles, serverless workloads, Kubernetes clusters, and APIs. “Logging into a box” became just one of many access flows.

Authentication matured. Human users authenticated through SSO providers like Okta and Azure AD. Machines and workloads used short-lived credentials—AWS STS tokens, Azure Managed Identities, GCP workload identity federation. Static secrets were increasingly deprecated. Passwordless, ephemeral authentication became standard practice.

The challenge shifted decisively to **authorization**: what should an authenticated identity be allowed to do, and for how long?

Persistent access to cloud IAM roles or permissions became one of the biggest risks. Breaches like **SolarWinds (2020)** and **Uber (2022)** showed how attackers could exploit standing entitlements or leaked credentials to move laterally and escalate privilege across cloud accounts.

Technologically, this period saw the emergence of **sophisticated cloud APIs** for managing IAM. These made it possible to move away from infrastructure-based solutions like bastions or proxies. Building on these APIs, modern PAM practices took shape:

- **Just-in-Time (JIT) access:** Grant access only when requested, for a defined task and duration.
- **Granular entitlements:** Define narrowly scoped policies—e.g., read-only access to an S3 bucket, temporary admin to a Kubernetes namespace.
- **Least privilege enforcement:** Not just for users, but also for non-human identities like service accounts and IAM roles, and eventually, for AI agents
- **Cloud-native auditability:** Use AWS CloudTrail, GCP Audit Logs, or equivalents to track access and feed signals to SIEMs and SOAR tools.

In parallel, a new category—**CIEM (Cloud Infrastructure Entitlement Management)**—emerged as part of CSPM/CNAPP platforms. These tools helped visualize over-permissioned identities and reduce sprawl. However, CIEM tools focused on **visibility**—they surfaced risks but did not control or provision access. PAM platforms, by contrast, serve as the **enforcement layer**—deciding who gets access, when, and how.

At the same time, developer expectations changed. With faster shipping cycles and cloud-native workflows, PAM could no longer slow teams down. Modern solutions had to integrate with CLI tools, CI/CD pipelines, and approval systems—delivering JIT access without compromising developer velocity.

In this era, PAM is no longer about secrets. It is about governing privilege dynamically—across identity, context, and time.

API-led PAM vs legacy PAM: What's different?

Modern PAM is no longer a vault—it's an orchestration layer:

- Identity providers (e.g., Okta, Azure AD) handle authn for users and devices.
- Cloud IAM APIs handle JIT role provisioning and deprovisioning.
- Approval workflows initiate JIT access and revoke it when the task ends.
- Audit pipelines stream events to SIEM/SOAR platforms for detection and response.

The result: ephemeral, least-privileged, and fully auditable access—by design.

But in modern environments, enforcement alone is not sufficient. Access is provisioned dynamically across multiple control planes—cloud IAM, Kubernetes, GitHub, CI/CD pipelines—many of which operate outside the direct reach of the PAM tool. To enforce policies like least privilege or JIT access, PAM needs to understand the full picture of what access already exists, how it's granted, and whether it's still needed.

This is why next-gen PAM platforms rely on a continuously updated **access data layer**—a model of all human, non-human and agentic identities, their effective entitlements, associated credentials, and the systems they can access. We refer to this as **Access DNA**.

Earlier generations of PAM didn't need this layer—because access passed through a vault or a bastion that also served as the point of enforcement. But in today's decentralized, API-driven infrastructure, **enforcement is only effective when built on top of access-aware analysis**.

In addition, API-led PAM platforms will use Gen AI to improve the capabilities of the Access DNA layer. These capabilities include, but are not limited to:

- Predictive insights on where standing access or secrets will become a risk.
- Smart recommendations for role refinement, privilege reduction, or access expiry.
- Automating access decisions using ML-models trained on signals such as prior usage, peer / role comparison and identity behaviors

What’s next for API-led PAM? Securing AI agents

As organizations adopt AI agents across engineering and operations, PAM systems must adapt—not with new principles, but by applying existing ones to a new identity type.

Tools like Claude, Cursor, and others may access privileged systems like GitHub directly or through a Model Context Protocol (MCP).

As with users and service accounts, this access must be governed. **The same principles apply:**

- Short-lived access: Agents should use ephemeral credentials, never persistent secrets or PATs.
- Least privilege: Access should be narrowly scoped to the specific task or repository, not broad admin rights.
- Auditability: Actions taken by agents must be logged and attributable—especially when access involves sensitive systems or data.

In some cases, privileged actions—such as modifying infrastructure or accessing production databases—should be subject to just-in-time (JIT) elevation and require human approval.

API-led PAM systems will evolve to treat agents as first-class identities.

The core principles—short-lived access, least privilege and auditability—remain the same.

Modern PAM platforms will extend their control planes to manage agents (like Claude) and provision access to privileged systems (like GitHub) through native IAM APIs.

MCPs may act as intermediaries, brokering these access requests in real time.

Conclusion

PAM’s core objectives haven’t changed: enforce least privilege, ensure short-lived access, and maintain auditability. But the context in which PAM operates has shifted dramatically.

Today, organizations manage a mix of cloud-native systems as well as on-prem infrastructure. Legacy PAM tools—built for static environments and credential vaulting—can’t meet the scale, speed, or integration needs of these environments.

Modern PAM must:

- Operate across hybrid infrastructure without requiring proxies or bastions
- Integrate with identity providers and cloud IAM APIs
- Support fine-grained, just-in-time access for all identity types
- Deploy quickly and scale without extensive services or operational overhead

The cost of getting this wrong isn’t just complexity—it’s persistent privilege, unmanaged access, and security gaps that legacy systems can’t see or fix.

P0 SECURITY

Contact Us

Email: info@p0.dev
Web: www.p0.dev

P0 Security is the unified access privilege platform built for modern hybrid and multi-cloud environments.

Where legacy IAM, PAM, and IGA tools fall short — particularly around non-human identities, ephemeral infrastructure and developer velocity — P0 delivers orchestration and governance, visibility and risk posture across all cloud environments.

With an agentless, API-native architecture, P0 helps teams enforce least privilege by default through short-lived, just-in-time access for both human and machine identities. Security and platform teams rely on P0 to reduce blast radius, streamline audits and eliminate manual provisioning without slowing down development.

P0 is trusted by leading organizations in fintech, healthcare, AI, and cloud-native tech, with full enterprise deployments completed in under 60 days. Learn more at www.p0security.dev.