The path to least privilege: a practical approach to Just-in-Time access

PO SECURITY

Executive summary

The way most teams handle access today is broken.

Static roles. Standing privileges. Long wait times for approvals. Manual access cleanup that rarely happens. Security teams are underwater, and engineering teams are stuck waiting on tickets just to do their jobs.

Meanwhile, identity has become the new perimeter—and the attack surface is exploding. Cloud environments now hold thousands of identities, most of which are over-permissioned, under-governed, and invisible until something goes wrong.

Just-in-time (JIT) access is the shift forward. Instead of granting standing access that lingers indefinitely, JIT lets teams request the access they need, when they need it, and only for as long as they need it. The result: lower risk, faster approvals, and better visibility—without slowing anyone down.

Moving to JIT is not as simple as flipping a switch.

Most teams don't know where to start. They aren't sure how JIT should be implemented in their organization, how to create policies, or how to get buy-in. Teams push back on losing "always-on" access. Ownership is murky. And there's the fear that security might block progress rather than unlock it.

This guide cuts through that noise. It lays out a practical path to least privilege through JIT—starting small, proving value, and scaling up. It's based on what we've seen in the field: what works, what doesn't, and how teams go from concept to production.

You don't have to boil the ocean. You just have to start.

And P0's solution to JIT gives you a way to get there without slowing teams down or giving attackers more surface to target.

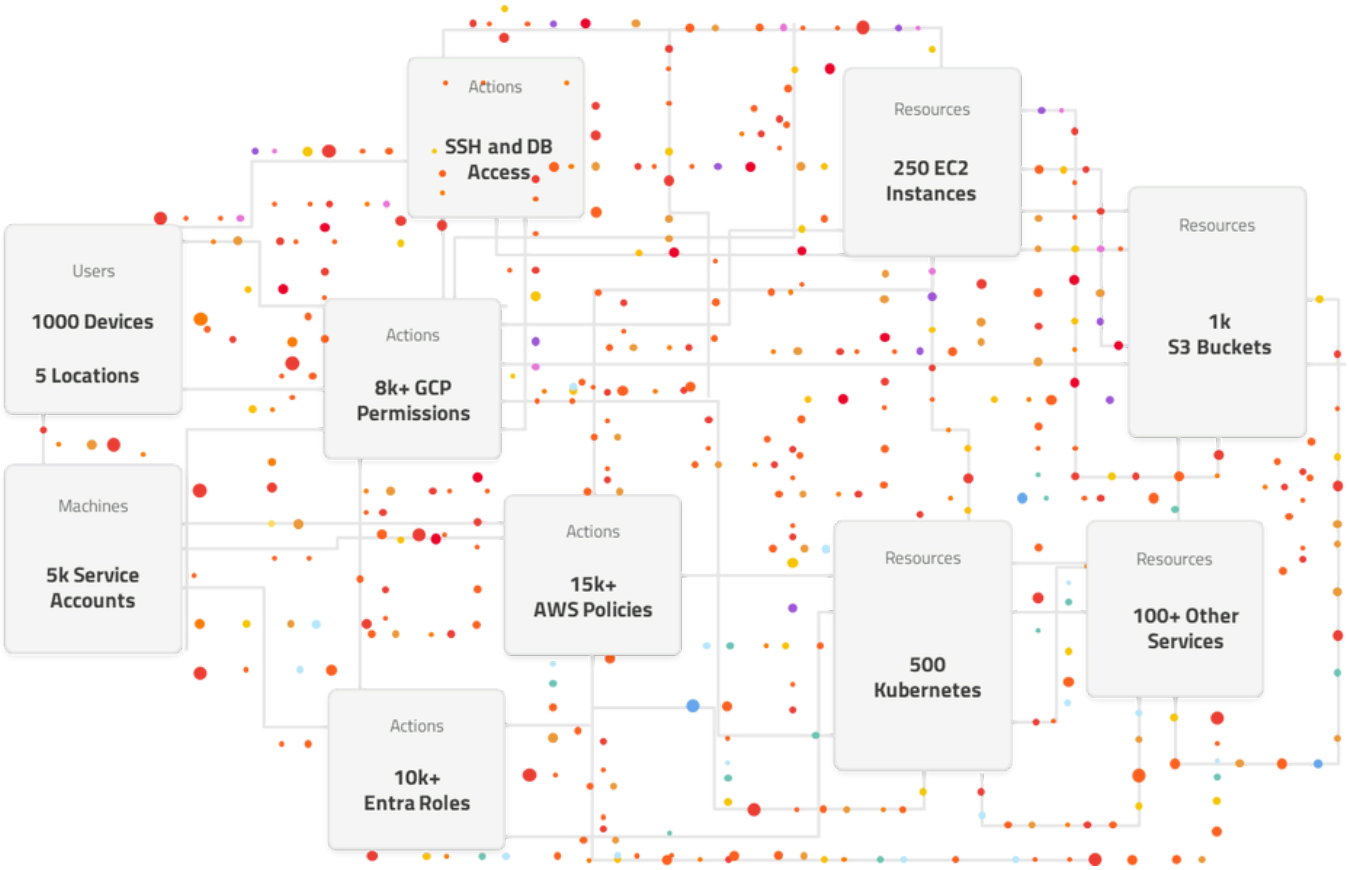
Problem landscape

Most access today is a mess. It's too slow, too broad, and not nearly secure enough. Developers and SREs are stuck in outdated workflows—think ticket-based approvals in ServiceNow or Jira. The process can take hours or even days, while developers are stuck, blocked on their work, waiting for access to be manually provisioned. Meanwhile, security teams have no real visibility into who is accessing what or why, let alone any ability to enforce policy in real time.

Once this access is granted, it's often permanent. Access that was meant to be temporary often persists - indefinitely. Roles are over-permissioned. Admin privileges go unrevoked. Static credentials and hardcoded keys are embedded in scripts and stored in repositories. All of this adds up to a huge blast radius and a real audit headache.

To make things worse, the access patterns themselves don't reflect modern security best practices. People get broad, persistent access instead of narrow, short-lived access. Groups and roles are managed manually and inconsistently. There's no continuous validation. No accountability. And no easy way to clean things up.

An ever-growing web of users, machines, service accounts - each wired to thousands of permissions, roles and credentials that reach deep into infrastructure. It's not just over-permissioned—it's overgrown, fragmented and nearly impossible to govern at scale.



Even when manual workflows are in place to remove access, they often fail. Someone was supposed to go clean up that role—but didn't. The access stays in place. No one notices. Until the wrong person uses it at the wrong time.

It's not just a tooling problem. It's organizational. Security wants to lock things down. Developers want to move fast. Nobody wants to be the person who tells a team they're losing access. So nothing changes.

All of this makes it harder to adopt least privilege, because to get there, you first need to understand your access landscape: what permissions are actually being used and what can safely be removed Most teams don't have that kind of visibility, let alone the political capital to make sweeping changes.

Legacy orgs struggle with binary access models—you either have access or you don't. These systems weren't built for today's layered cloud environments or the explosion of identity types.

On the other hand, startups often over-provision developers early to stay fast and unblocked. But when it's time to rein that access in, teams hit pushback: developers resist losing standing access, and security doesn't want to be the bottleneck. That tension makes enforcing least privilege feel risky—even when everyone agrees it's necessary.

The result? A fragile access model that everyone hates and attackers love.

71%

surge in attacks using valid credentials

(IBM X-Force, 2024)

88%

of web app breaches involved stolen / compromised credentials

(Verizon DBIR, 2025)

77%

of attacks begin with credential compromise; 56% directly caused by it

(Expert Insights)

The journey to JIT

If you try to roll out just-in-time access across your entire org on day one, it'll fail. Not because JIT doesn't work—but because access is messy. Teams are different. The politics are real. And change always comes with pushback.

That's why we suggest a phased approach.

Start small, prove value, build momentum. Some customers have run this as an 8-day sprint. Others stretch it into a multi-week rollout with phased cleanup and enforcement. You can flex the timeline—the structure stays the same.

In the next pages, you'll see how each stage comes together—mapped to actions, outcomes and measurement.

Stage 1: Prove

Goal: Understand current elevated access flows and test JIT with a small team.

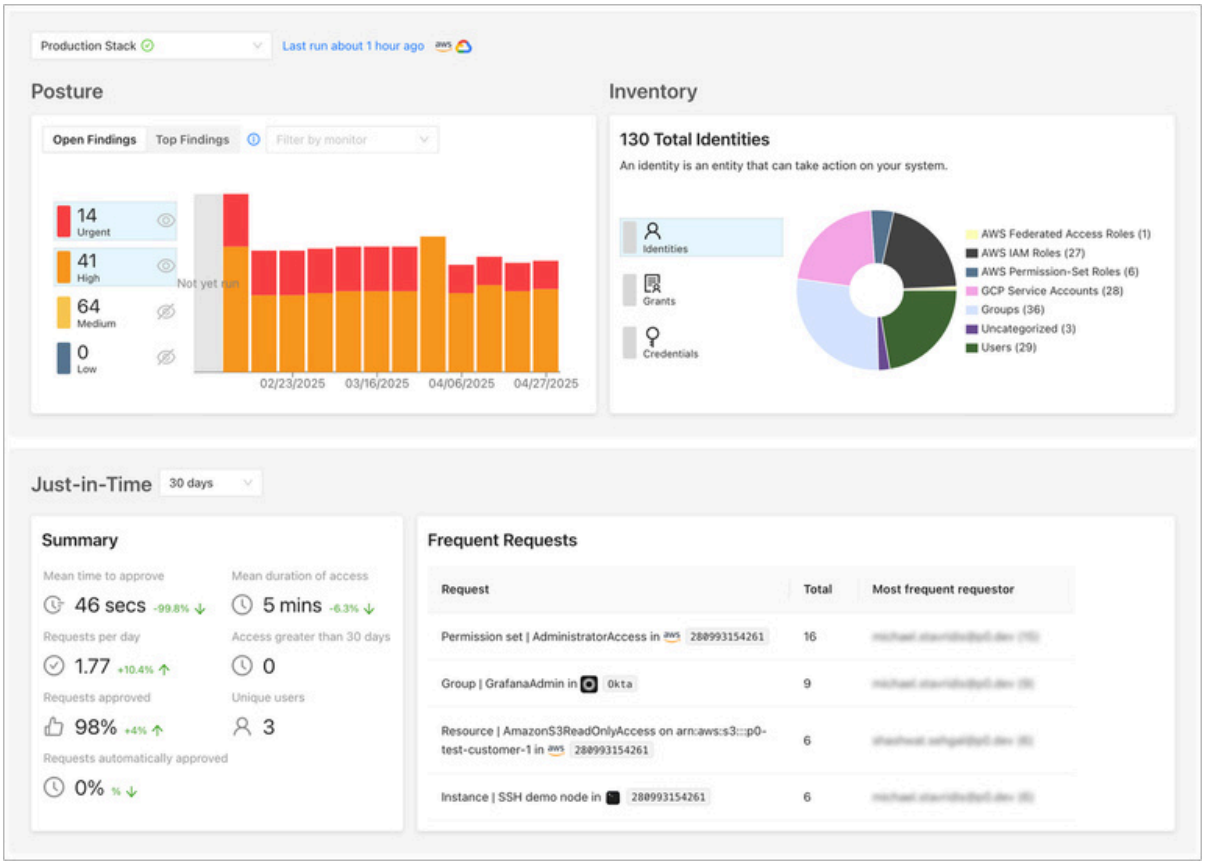
P0 action plan

1. **Audit your current elevation flows:** Look at how people are getting access today. Pull data from Jira, ServiceNow, or whatever system you're using for manual access requests. How long does it take to get access? How much time is wasted manually provisioning access? What's the denial rate? Where are unexpected one-off approvals happening?

2. **Pick a friendly team to pilot JIT:** Choose a team that wants better access, not one that's likely to resist. Identify a use case they currently have for elevated access (something they're currently making tickets for), and replace that with P0's JIT access. Set them up with P0's Slack, Teams, or CLI integrations. Let them request and approve access in real time.

3. **Measure impact:** Track MTTA (mean-time-to-access) before and after. Get NPS from pilot users. Compare the JIT access to the historically granted access: you should see more granular resource-based permissions rather than broad admin access.

   [real-life datapoint: One customer's first policy cut MTTA by 90% — just by granting staging DB access automatically during work hours, and requiring peer approval after hours.]



Key considerations

- **Start simple, not perfect:** Don't wait to revoke every permission before piloting JIT. Start by only replacing current elevation flows– nobody loses access, so you'll get less resistance. Once JIT is implemented, you'll revoke risky standing access in stage 3.
- **Baseline the right metrics:** Benchmark MTTA. Identify frequent access escalation paths.
- **Start with a motivated team:** Platform or infra teams usually feel the pain and are ready to try something new.
- **Policy uncertainty is normal:** If you're unsure about your JIT configuration, start with what users are already requesting.
- **Pushback is likely:** Developers may worry JIT slows them down. Show them how it works - instant access, without requiring slow ticket-based approvals

Signals that you're in this stage:

- No clear access ownership
- Approvals are rubber-stamped or consistently delayed
- Standing access is rarely cleaned up
- Security can't prove who has access to what
- Devs complain about access friction, but nobody owns fixing it

What to do next

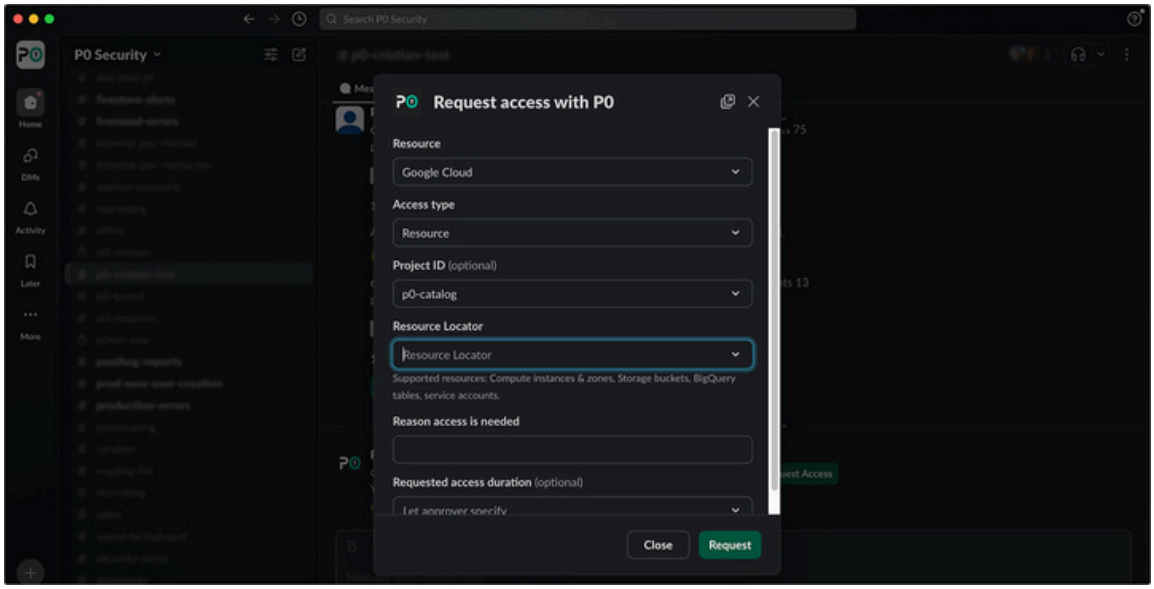
- Identify an access process that already exists and can be transitioned to JIT
- Look for frequent escalations or delayed approvals
- Start with a motivated team and a narrow scope
- Show that JIT doesn't mean slower access—just smarter access

Stage 2: Deploy

Goal: Roll out JIT org-wide with real policies and real accountability.

P0 action plan

1. **Build org-wide access policies** using identity metadata like team tags, cloud labels, and historical request data. P0 will recommend scoped, repeatable JIT policies using this data. Define who can request what, when, and from whom.
2. **Integrate P0 across workflows:** Enable JIT access via Slack, Teams, or CLI. Integrate P0 with your production environments, including cloud and on-premise systems.
3. **Track adoption and progress:** Use P0 dashboards to measure % of requests flowing through P0, monitor lagging teams, and surface legacy paths (e.g., Jira, ServiceNow, static keys).
4. **Intercept legacy workflows:** Send automatic nudges (via Slack, JIRA, or other tools) when users default to old request methods. Redirect them to P0.



Key considerations

- Use P0's suggested policy templates to avoid drift; some common configurations are:
  - Low-risk access with no approval (e.g., staging read-only)
  - Production access with team lead approval (e.g., 1-hour max, justification required)
- Be prepared to handle:
  - Inconsistent metadata (e.g., bad tags, orphaned accounts)
  - Teams resisting change
  - Confusion from parallel or duplicate access processes

Symptoms when you are in this stage

- You've successfully piloted JIT, but adoption isn't widespread.
- Different teams still rely on Jira or ServiceNow for access requests.
- There's a lack of consistent policy enforcement across environments.
- You see some wins (e.g., MTTA improvements), but cleanup and standardization are lagging.

What to do next

- Prioritize sensitive environments (e.g., prod or customer data systems).
- Deprecate legacy flows—don't just block them, intercept and reroute users contextually.
- Socialize the new model; use docs, Looms, and tooltips to reinforce how access should work.
- Show clear value; highlight MTTA reduction (e.g., 10 hours to 3 minutes).
- Accelerate with templates to roll out consistent logic across teams.

Stage 3: Secure

Goal: Clean up access risk and enforce least privilege in production.

P0 Action Plan

1. **Remove risky access**

   Start with permissions that haven't been used in 90 days. Prioritize high-sensitivity access like prod admin roles and static credentials (keys, tokens).

2. **Provide an easy path to re-request access**

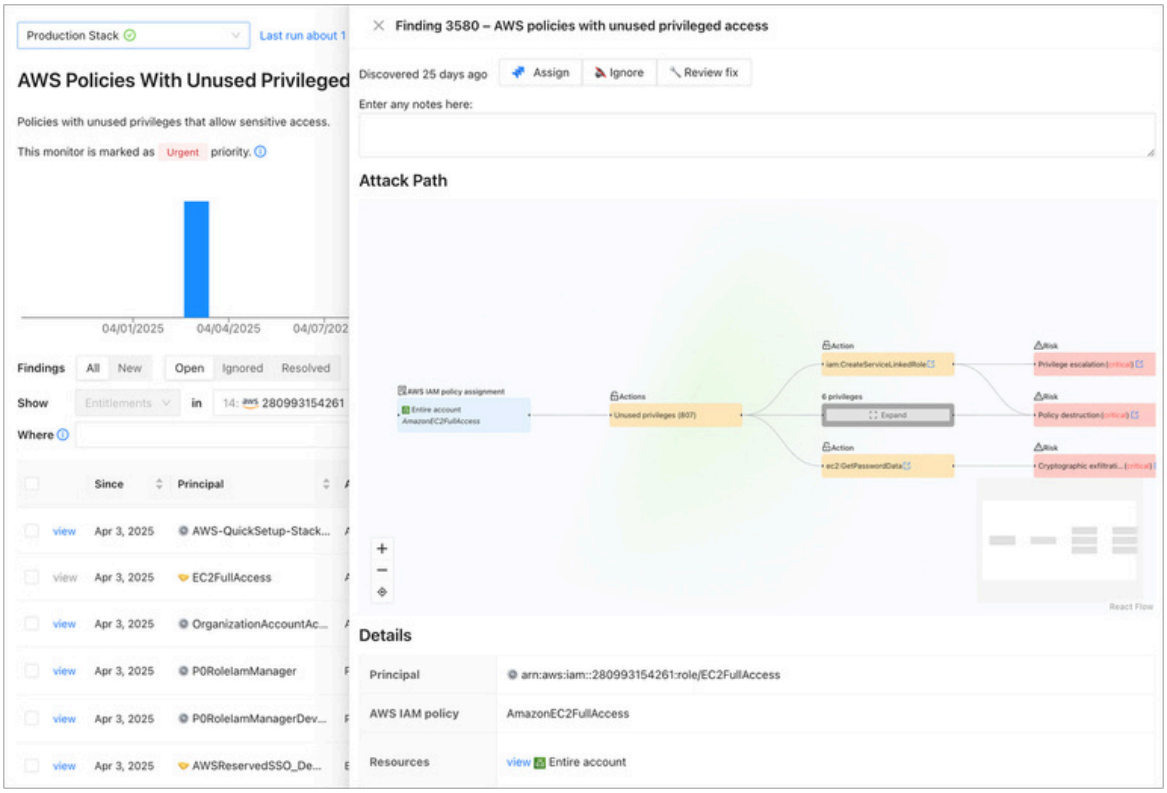
   When cleaning up access, allow users to request JIT access to the same permissions in P0 instead. This minimizes pushback and builds confidence in enforcement.

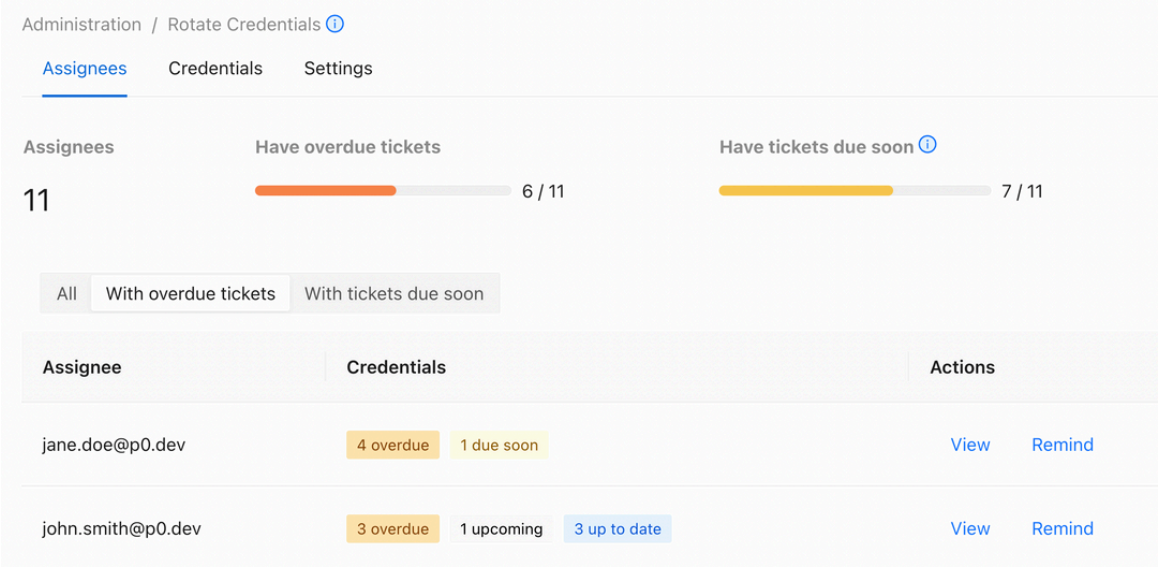
3. **Automate enforcement**

   Set policies in P0 to continuously remove unused access and trigger cleanup workflows when violations are detected—no manual review required.

4. **Rotate and revoke credentials continuously**

   Use P0 to auto-rotate secrets and remove static keys across environments—ensuring no long-lived machine access remains.





Key considerations

- **Start with easily revertible changes to build confidence**
  - Begin with changes that can easily be reverted, such as disabling accounts and keys rather than deleting them. If they turn out to be necessary, you can quickly revert them through P0.
- **Automate cleanup and provide a path back**
  - Access removal feels risky without a path to recovery. Ensure that users can easily request removed access again, without opening tickets.
- **Don't stop at humans**
  - Machines are the next surface. Look for static credentials in CI pipelines, bots with always-on IAM roles, or ephemeral jobs running with broad scopes. JIT works here too—and often yields the biggest security wins.

Symptoms when you are in this stage

- Your environment is littered with unused access, static credentials, and legacy roles.
- Devs and security teams worry that enforcing least privilege might break something.
- Your security team is manually cleaning up access—or avoiding it due to lack of ownership.
- There's no centralized accountability for ongoing IAM hygiene.
- You're aware that non-human identities (NHIs) are completely unmanaged—and you're starting to feel the risk.

What to do next

- **Remove risky access**
  - Start with access that hasn't been used in 90 days. Then target high-sensitivity roles (e.g., prod admin or customer data). Don't forget static keys.
- **Automate enforcement**
  - Set rules to continuously remove unused access. Flag violations. Trigger cleanup workflows automatically.
- **Minimize pushback**
  - Reinforce that access isn't being taken away—it's just being requested when needed.
  - Use data (e.g., "This access hasn't been used in 120 days") to back your decisions.
  - Lean on automation to avoid manual intervention and politics.
  - Tie the effort to audit milestones or compliance initiatives to reinforce urgency.

This is where the value of automation and policy enforcement compounds. Access gets safer, faster and easier to govern. Teams move faster. Security breathes easier. And you've got a real foundation for least privilege that doesn't depend on good intentions and manual cleanups.

Beyond humans: Just-in-time for machines

Once you've cleaned up human access, you'll start noticing the other half of the problem—the part nobody wants to deal with: machine identities.

These are your service accounts, IAM roles, tokens, CI/CD pipelines, Kubernetes workloads, Lambda functions, Terraform plans, bots, scripts, and so on. They're everywhere. And they usually have more access than any human ever would.

That makes them one of the fastest-growing and riskiest surfaces in the cloud. And in most orgs, they're completely unmanaged. Static keys remain active indefinitely. Machine identities are created once and never revisited. Expiration is rare. Rotation is manual at best.

Here's where JIT comes in.

The same just-in-time principles that improve human access can be applied to workloads and automation:

- **Short-lived tokens** instead of long-lived keys
- **Access granted on a just-in-time basis**, not static assignments
- **Key rotation and auto-expiry** for credentials that cannot be replaced with short-lived tokens

It's the same security outcome—lower standing access, smaller blast radius, better governance—just applied to code instead of people.

Our platform was purpose-built to manage JIT access for both human and machine identities. P0 ties together identity metadata, cloud IAM permissions, service accounts, ephemeral jobs, and secrets - and applies real-time policy and rotation across the entire lifecycle. This eliminates static keys, manual scripts, and access blind spots.

Once you've got a handle on human access, non-human identities are the next frontier. And JIT is how you secure them without slowing anything down.

Most organizations manage non-human identities as an afterthought. Native tools might help you spot static keys or unused roles, but they rarely show the full picture—let alone enforce least privilege across multiple clouds. And they don't provide a unified system to govern, revoke, and monitor access in real time. That's where P0 comes in: one identity graph, one policy engine, and one path to enforce least privilege for every principal—human or machine.

Metrics that matter

You can't manage what you don't measure—and access is no exception. As you roll out just-in-time, it's important to track real impact. That means going beyond surface-level adoption and digging into what's actually improving across security, productivity, and user experience. You can track these metrics over time using P0's reporting dashboards.

Here are the human requestor metrics that matter most.

- **MTTA (mean-time-to-access)**
  - This is your north star for user experience. How long does it take to go from "I need access" to "I have it"? Track this before and after JIT rollout. You'll likely see days turn into minutes—and that matters when you're debugging, responding to an incident, or trying to ship on a deadline.

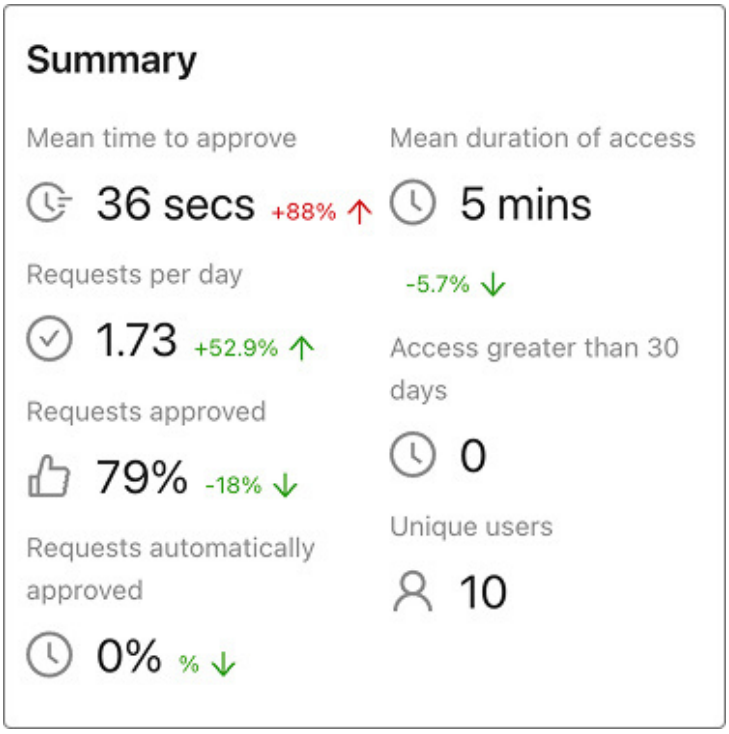
- **Standing access footprint**
  - Measure the total number of standing (always-on) permissions across your cloud environment, broken down by role type, sensitivity, or business unit. The goal is to shrink this over time—and JIT should be the driver that lets you do it safely.

- **Access request coverage**
  - What percentage of access requests are now flowing through P0 instead of legacy tools like ServiceNow or Jira? This tells you how much of your org is actually using the new path—and where you still have holdouts.

- **NPS of the access experience**
  - This is your gut check. Ask engineers and admins: how is this working for you? Are requests faster, more predictable, less annoying? You'll get a mix of feedback—and that's gold. Use it to improve the experience and build advocates.

- **Access granularity**
  - Are you granting more specific, time-bound access than you were before? Look at how many JIT policies are scoped to single roles or resources, versus broad groups or admin access. Granularity is a proxy for maturity.

- **Privileged access cleanup rate**
  - Track how many stale, risky, or unused permissions have been revoked since JIT was implemented. If you're doing this right, that number should keep climbing.



Non-human identity metrics that matter

- **Reduction in long-lived credentials**
  - Track how many hardcoded tokens, static access keys, and long-duration secrets you've replaced with short-lived, time-bound alternatives. This is often your first big win—especially in Cloud environments or CI/CD systems.

- **Secrets rotation frequency**
  - How often are machine credentials rotated today? After JIT rollout, you should see this move from quarterly (or never) to automatic and frequent (e.g., per deployment or per session).

- **JIT coverage for automation workflows**
  - What percentage of pipeline jobs, bots, or workload executions now use JIT-generated access rather than pre-assigned roles?

- **Standing access reduction for service accounts**
  - Just like with humans, you can track which service accounts had broad, permanent roles—and how many of those have been scoped down or moved to request-based flows.

- **Mean time to clean-up NHI access risks**
  - Time from "last seen" to "access revoked" matters. Before JIT, this could be months. With P0, unused roles or secrets can be automatically removed—and you should be tracking how fast that cleanup happens.

- **Access variance across environments**
  - Are test/staging/prod environments all using the same high-privilege service accounts? Post-JIT, you should see more environment-specific, purpose-built access scopes with less duplication.

These aren't vanity metrics. They help prove to your leadership team—and your developers —that this isn't just another security initiative. It's a better way to grant access.

Conclusion

You don't need a three-year roadmap or a new security team to get started with just-in-time access. You just need a use case, a willing team, and a little bit of pressure to do things better.

The goal here isn't perfection. It's progress. Replace one ticket-based workflow. Clean up one over-permissioned role. Pilot one policy with one team. You'll learn more in that process than in months of planning.

Because once teams see how fast and seamless access can be—without the overhead of manual provisioning and risk of standing permissions—they don't want to go back. Just-in-time access isn't some abstract future state. It's something your team can run today. It scales. It makes audits easier. It reduces blast radius. And it gives developers and security teams what they both want: speed and safety.

Least privilege isn't a blocker anymore. With P0 Orchestration and JIT, you can stop waiting and start moving.

P0 SECURITY

Contact Us

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

P0 Security is the unified access privilege platform built for modern cloud infrastructure. Where legacy PAM, IGA, CIEM, NHIM and IAM 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.