hoop.dev

WHITEPAPER

Comparing hoop.dev and CyberArk

Brokering access vs controlling actions on the wire

The credential was never the risk. The action is.

hoop.dev Layer 7 access governance · SOC 2 Type II · CNCF member · open source

CONTENTS

What this paper covers

EXECUTIVE SUMMARY

The perimeter moved from the credential to the action

For two decades, responsibly controlling infrastructure access meant controlling credentials. A privileged access management platform like CyberArk decides who gets a key, vaults it, brokers the connection, and records the session.

Because of advancements in machine learning inference and protocol reverse engineering, a new category of security is available for the first time that enables better reliability and security controls: runtime governance.

There are two reasons why credential brokering is no longer an effective posture as a standalone measure.

First, it is the actions that cause real damage, destructive commands, out-of-band changes, sensitive data leaving a session, all of which happen after the credential is granted, where a vault cannot see or stop them.

Second, AI agents now act on infrastructure using credentials they inherit from engineers, multiplying the identities with live access faster than any door-level control can keep up.

The control point is moving from the credential to the action. **hoop.dev governs the action at runtime, in the data path, while it happens.** It blocks or routes risky commands before they execute, masks sensitive data inline, and records every action against a single identity. It does this for human and non-human identities through one policy, and can run alongside the PAM you already operate. The result is a smaller blast radius, faster compliance evidence, fewer incidents, and AI adoption that is not blocked by access risk.

This paper explains where credential brokering breaks, how runtime governance works, and how the two fit together.

SECTION 01

The control point is moving

Picture the perimeter as a building. Credential brokering controls access to the door: who holds a key, who is allowed in. Runtime governance controls actions in the room: what each identity does once it is inside, while it is doing it.

hoop.dev controls both. It grants and expires access at the door, the job a vault also does, then keeps governing the action inside the room. A vault does the first and stops at the door. That is the gap this paper is about, because every modern access risk now lives inside the room.

> **A credential is not dangerous. The actions taken with it are.**

A vault that confirms a valid key was used answers the wrong question when production has already been changed, when material non-public information has already left a query, or when an agent has already run a command no one approved.

FIGURE 1 · CONTROL-POINT PLACEMENT

Where the control point sits

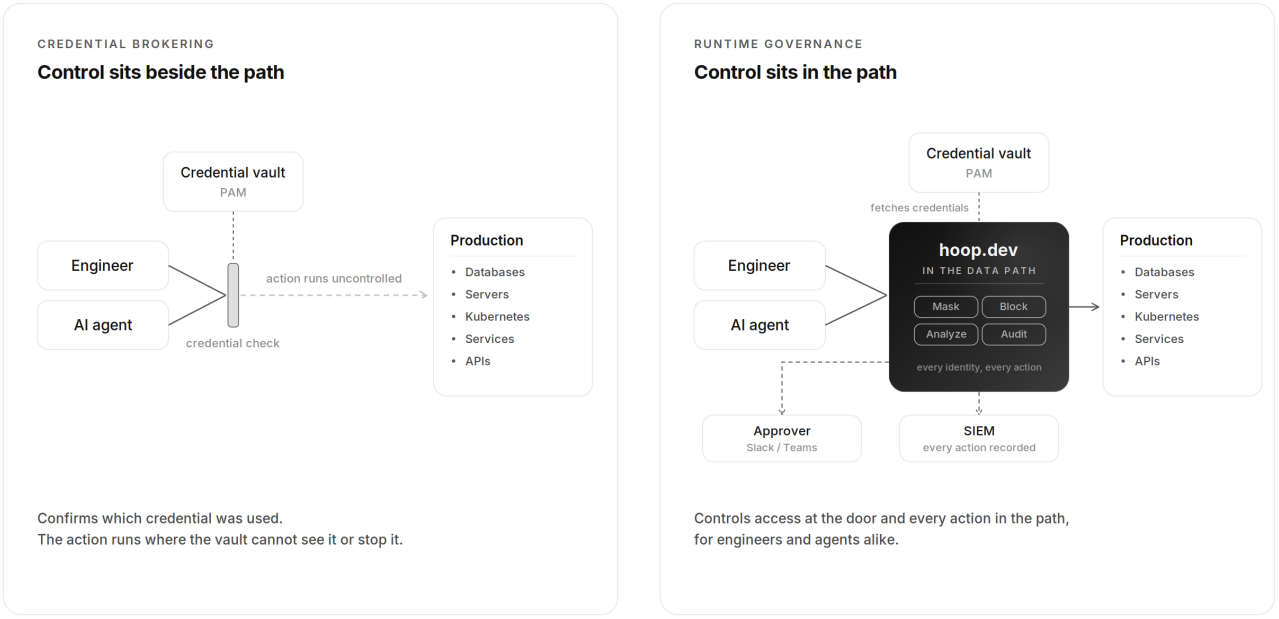

FIGURE 1 A credential vault sits beside the path at the door, and the action runs uncontrolled to production. hoop.dev sits in the path and gates every action, for engineers and agents alike.

SECTION 02

How credential brokering works, and where it ends

CyberArk and tools like it do three things well: they vault credentials, broker access to target systems, and record sessions for later review. For known humans reaching known systems, this was sound. It answers the question it was built for: which credential was used.

The model ends at three boundaries.

> **It sits beside the session, not inside it**
>
> A vault brokers the connection and steps aside. It does not read each command as it runs, so it cannot evaluate or stop the command itself.

> **It records for review, not for prevention**
>
> Session recording produces an artifact you read after the fact. It tells you what happened. It does not decide, in the moment, whether the action should happen at all.

> **It was built before agents existed**
>
> PAM assumes one credential maps to one accountable human. Agentic workflows break that assumption, because a single engineer's credential can be inherited by many automated actors.

None of this is a flaw in execution. It is the boundary of the design. Controlling the door was the right call when that was the primary risk.

SECTION 03

Four failure modes

Each scenario below is drawn from the field and anonymized. Each is a place where the control point sat at the door while the damage happened in the room.

> **Out-of-band changes**
>
> At a publicly listed financial-data company, engineers make changes through cron jobs and bash scripts that do not always flow through the privileged access tool. When something breaks, the security team can confirm a change occurred but cannot reliably say who made it or when. The vault holds the credential record. It never saw the command.

> **Destructive commands**
>
> With sufficient permissions in a cloud console, a single identity can delete production systems and their backups in a few commands. In one real case, a routine certificate update wiped an entire certificate store, and the company temporarily lost the ability to decrypt customer data. A vault attributes the action afterward. It cannot intercept the keystroke, because it is not in line to catch it.

Sensitive data leaving a session

Some sensitive data is a known type, an email or a phone number. Some is contextual: a single field that names an unannounced acquisition, or a query an agent uses to compute a gross margin it should not see. Column-level blocking does not catch this, because the data is legitimate and the risk is in how it is used. A vault does not inspect the data flowing through a session at all.

Non-human identities

As teams adopt coding agents and automated services, each one acts using credentials inherited from an engineer. A team with thirty privileged humans can quietly become a far larger population with hundreds of non-human actors, each able to touch production. The control-the-door model responds by issuing more keys, and the blast radius grows with every one.

SECTION 04

How runtime governance works

hoop.dev is a Layer 7 gateway that sits in the data path between any identity and the infrastructure it reaches: databases, servers, Kubernetes, and APIs. Because it sits inline on the live connection, it can read what is being requested and act on it in real time. That placement is the mechanism. Inline enforcement is only possible when the enforcement point is in the session, not next to it.

From that position, hoop.dev does four things a vault cannot.

Assess and gate the command before it runs

When an operation changes from reading a status to restarting a service, hoop.dev scores the risk inline. Low risk runs. Medium risk routes to a named approver in Slack, Teams, or your ticketing system before execution. High risk is blocked. The destructive command never reaches the target, because it passes through hoop.dev first.

Mask sensitive data inline

hoop.dev identifies known sensitive types using machine learning inference, with no requirement to map your schema first. For data that is not a known type, you author a contextual policy that uses a model to score the request and decide whether it runs, routes for approval, or is denied. The decision happens before the data leaves the session.

> **Record every action with full context**
>
> Each action is logged with who ran it, what it did, when, why, the risk assessment, the data involved, and whether it was masked, all tied to a single identity. Session data streams to your SIEM in real time. "A credential was used" becomes a complete, queryable record of what was done with it.

> **Govern humans and agents through one policy**
>
> The same rules that gate a human engineer gate an agent on the same connection. As non-human identities multiply, the control point stays in one place instead of fragmenting into one key per actor.

hoop.dev also controls access at the door. It grants access just in time and expires it on a schedule or after a set window, so the number of identities with standing access to production stays small. Teams turn ad hoc scripts into reviewed, no-code tools through a standard GitOps workflow, so even break-glass access runs through code review. The door is shared ground with a vault. The runtime is where hoop.dev goes further.

SECTION 05

CyberArk vs hoop.dev, side by side

	Credential brokering (CyberArk)	Runtime governance (hoop.dev)
Where the control point sits	At the door, on the credential	At the door and in the running action
What it can answer	Which credential was used	Who ran what, when, why, on which data, masked or not, tied to one identity
Real-time prevention	Attributes a change after the fact	Blocks or routes the command for approval before it executes
Destructive command	Tells you who did it, afterward	Sees it inline, stops it, alerts security in real time
In-session data exposure	Not inspected	Masks known types via inference; contextual policy for novel sensitive content
AI and service identities	One credential, one assumed human	Governs human, agent, and service identities through one policy
Deployment	Programs measured in quarters; dedicated staff to operate	Inline at the wire, no infrastructure cataloging up front, cloud agnostic
Coexistence	Often a rip-and-replace program	Runs alongside CyberArk, feeds your SIEM
Verifiability	A closed platform you trust	Open source: read the enforcement code

SECTION 06

The four outcomes leadership cares about

Runtime governance is the mechanism. These are the outcomes it produces.

AI acceleration

The fastest way to slow AI adoption is to make production data unreachable, which pushes engineers toward stale copies and data lakes that do not reflect reality. hoop.dev lets engineers and agents reach real production data safely, so the secure path becomes the fast path. Teams adopt AI faster because access risk stops being the blocker.

Developer productivity

Access that used to wait on a ticket and a human gatekeeper becomes a self-service just-in-time grant with masked data that expires on its own. Developers reach what they need through their existing tooling, often without knowing hoop.dev is in the path. Less waiting, no standing access to manage.

Compliance evidence

Every action, the data it touched, and the approval behind it is recorded automatically. When an audit arrives, the evidence is already assembled rather than reconstructed. hoop.dev generates evidence for SOX, HIPAA, GDPR, SOC 2, and other audits. It does not make an organization compliant. It gives the auditor the record.

Risk mitigation

The blast radius shrinks on two fronts: fewer identities hold standing access, and the actions those identities take are gated in real time. A destructive command is stopped before it runs, not investigated after.

SECTION 07

hoop.dev can run alongside CyberArk

This does not have to be a rip-and-replace decision.

hoop.dev deploys at the wire, plugs into the identity provider and permissions you already run, and works across AWS, GCP, Azure, and OCI without a months-long cataloging project. It runs alongside your existing PAM. CyberArk keeps brokering credentials at the door. hoop.dev adds the layer the vault was never positioned to provide: real-time prevention, inline data masking, and action-level audit for every identity, including the agents now entering your stack.

> THE TAKEAWAY
>
> For a security team, that means a new layer of control and a complete audit trail with no migration and no disruption to the access program already in place. Value on day one, alongside what you have, not instead of it.

SECTION 08

You should not have to trust a black box

A vault asks you to trust its decisions without showing you how it reaches them. hoop.dev takes the opposite stance. The enforcement engine is open source, so a security team can read exactly what governs its engineers and its agents before deploying it. The code is public at github.com/hoophq/hoop, and small teams can run it without a contract.

This matters most for the newest part of the problem. When a model decides what your agents can and cannot do with production data, taking that decision on faith is the wrong posture. Open enforcement lets you verify it instead.

SECTION 09

Compliance posture, stated plainly

hoop.dev generates evidence for SOX, HIPAA, GDPR, SOC 2, and other audits by recording every action, the data it touched, and the approvals behind it. It does not certify or guarantee compliance with any framework. The distinction is deliberate: hoop.dev produces the audit record; your auditor reaches the conclusion.

SOC 2 TYPE II | CNCF MEMBER | OPEN SOURCE · MIT

CONCLUSION

Credential brokering tells you who held the key. Runtime governance decides what happens in the room.

A vault knows which credential was used. It cannot tell you what was done with it, and it cannot stop the next destructive command before it runs, because its control point stops at the door. That was the

right boundary for an era of known humans and known systems. It is the wrong boundary for an era of engineers moving at the speed of AI and agents acting on production with inherited credentials.

hoop.dev moves the control point into the running action, for every identity, while keeping the access controls and the PAM you already rely on.

See it in the path

Read the enforcement code at **github.com/hoophq/hoop**, or book a walkthrough and we will map it to your stack.