The Context Layer

Engineering Business-Aware Vulnerability Prioritization at Scale

mauvehed & Nick Eckert

LASCON 2026

Agenda

1	**Framing and requirements**	mauvehed	~10 min
2	**Building the context layer**	Nick	~25 min
3	**What we got wrong**	Both	~5 min
4	**Q&A**	Both	~10 min

Who we are

mauvehed (job seeking!)

Security leadership, engineering, operations, infrastructure and more

20+ years in tech

A lifetime of shenanigans

Austin Hackers Anonymous (AHA!)

Hire me: https://resume.mvh.dev/

Nick Eckert

Senior Security Engineer

3 years defining, designing, and deploying a VM program built on business context, working at the data and asset level

Avid motorcyclist

This is part 3 of a LASCON series on vulnerability management

2024

Vendor Nightmares and Scoring Scuffles

The vendor critique

2025

Navigating the Challenges of Risk-Based VM in a Cloud-Native World

Joint talk with Phoenix Security

2026

The Context Layer

How we built it

Threat context tools have matured

Exploitability

Is it exploitable?

Is it in an active campaign?

Is there a PoC or known exploit?

Threat intel

Enrichment has matured across the industry

Reachability

Reachability analysis is now standard

What’s missing is the other half - Business Context

Data Classification

Is the data part of our core business?

Does it contain PII?

Is it subject to regulation?

Location

Is it client or Internet facing?

Is it secluded in a private VLAN that’s only VPN accessible?

Ownership

Is it actively owned by a team?

Does someone maintain and keep it up to date?

The same CVE in the same library can carry very different business risk

Finding A (low severity)

VPN access required

Internal-only employee service

Owned by an active team

Finding B (high severity)

Revenue-generating service

Contains PII

Runs in production

Nobody remembers who owns it

Less than 4% of scanner "criticals" stay critical with business context

< 4%

of findings scanners call critical stay critical

Fix 3 things this month, not 3,000

Vulnerability management at scale is a metadata problem

- Better security tools can't solve it
- **Developers/System managers own the metadata that drives prioritization**
- Stale catalogs, stale ownership, and drifting inventories break prioritization
- No amount of threat intel fixes bad asset data

Developers have more power over bad vuln reports than they think

The complaint

Low-quality vulnerability reports

False positives

Misrouted tickets

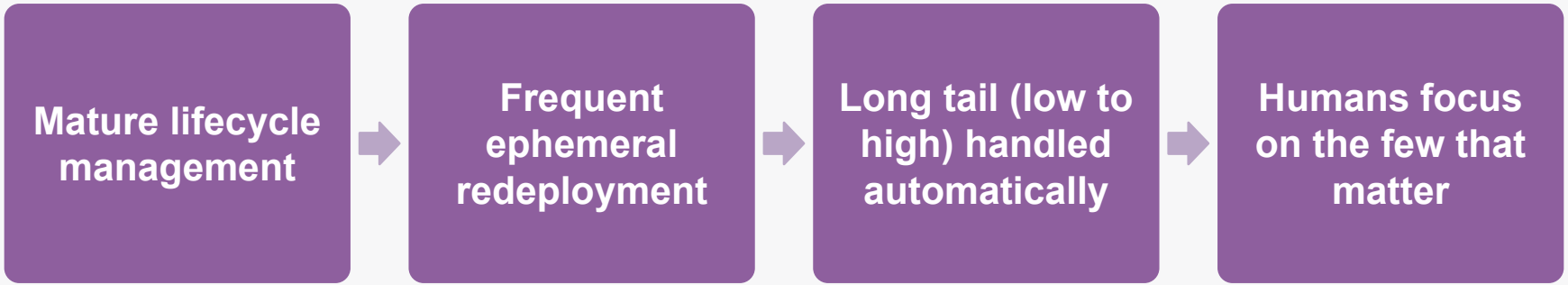
The fix

Lives in data owners already maintain

Standard operating procedure

Leverage your lifecycle management

DevOpsDays Austin 2025: "Give Up! Just Stop Patching Vulnerabilities!"



The goal: 1 to 3 real criticals per team per month

1-3

real criticals per team per month

Critical findings should mean something real

Teams shouldn’t be overwhelmed by volume

Response should be immediate and intentional

PART 2

Building the context layer

Nick Eckert

Phoenix Security as our prioritization engine



- Ingestion of findings data from Snyk and WIZ
- Prioritization based on business context
- Handles ownership and remediation tracking
- How do you assign ownership and define structure?

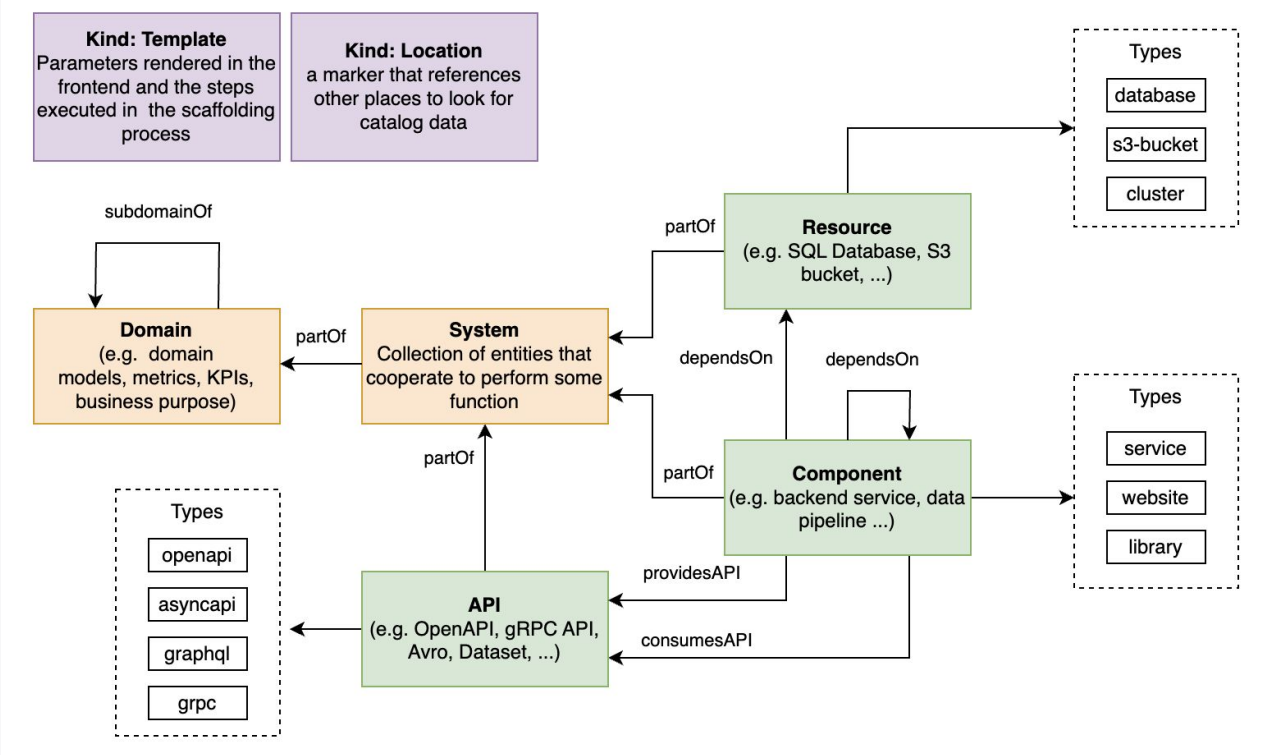
Service catalogs are an underused source of security data

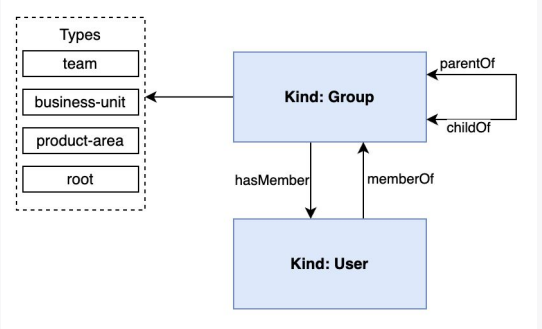
- Software catalogs get built for developer experience, not for security.
- The catalog already holds the hardest data to get
- We build the Phoenix structure and define ownership from the software catalog

Backstage holds the inventory we prioritize on

- Groups
- Systems
- Components
- Source repos
- Team ownership
- Component relationships

The Backstage System and Component model





What we pull and normalize from Backstage

Groups → Teams

- Groups created as teams in Phoenix if they don't already exist

Members → Users

- Members created as users if they don't exist then added to an existing teams

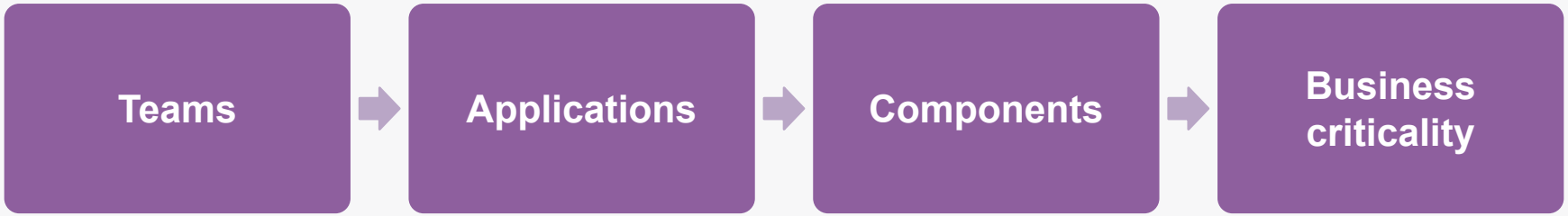
Systems → Applications

- New Application is create for every system in Backstage if it doesn't already exist.
- Application's owning team is updated to match the owning group in Backstage.

Repo-linked components

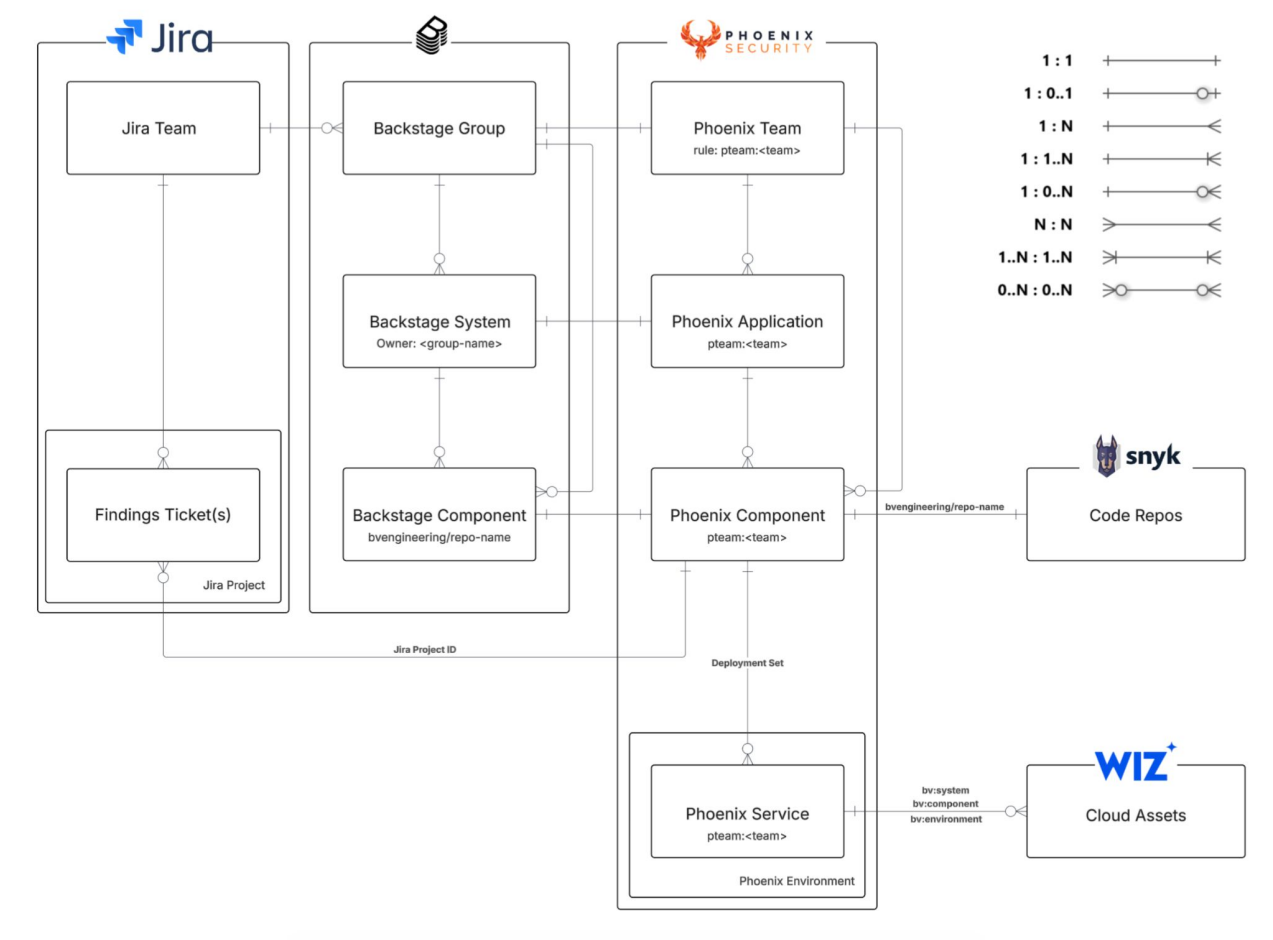
- The repo slug is read from the `backstage.io/source-location` annotation. The GitHub org name is prepended, giving `org/repo`, the format Phoenix uses for repository names.
- Components are identified by **name + application** rather than name alone, so two systems can each have a component called `api` without colliding.

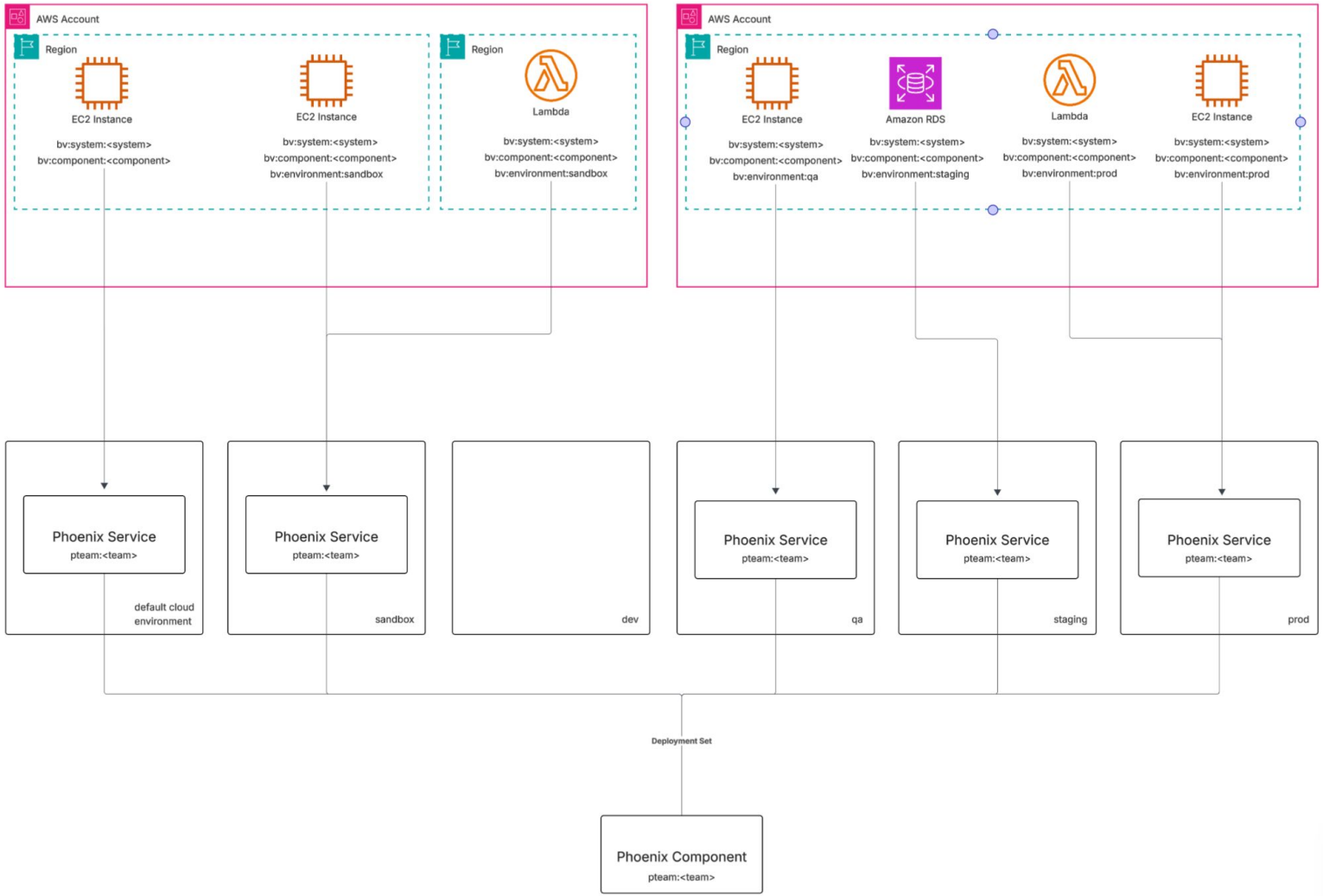
The sync script maps teams to applications, components, and business criticality



A custom script polls the Backstage catalog and continuously syncs into Phoenix

How Backstage data maps to Phoenix




AWS Account
Region
EC2 Instance
bv:system:<system>
bv:component:<component>
EC2 Instance
bv:system:<system>
bv:component:<component>
bv:environment:sandbox
Region
Lambda
bv:system:<system>
bv:component:<component>
bv:environment:sandbox
AWS Account
Region
EC2 Instance
bv:system:<system>
bv:component:<component>
bv:environment:qa
Amazon RDS
bv:system:<system>
bv:component:<component>
bv:environment:staging
Lambda
bv:system:<system>
bv:component:<component>
bv:environment:prod
EC2 Instance
bv:system:<system>
bv:component:<component>
bv:environment:prod
Phoenix Service
pteam:<team>
default cloud environment
Phoenix Service
pteam:<team>
sandbox
dev
Phoenix Service
pteam:<team>
qa
Phoenix Service
pteam:<team>
staging
Phoenix Service
pteam:<team>
prod
Deployment Set
Phoenix Component
pteam:<team>

Company tagging standard

Tag	Status	What it does
`bv:system`	**Required** (enforced by SCP)	Maps the asset to its Backstage **System** and owning group
`bv:environment`	**Required** (enforced by SCP)	Places asset in the right **environment**: prod · staging · qa · dev · sandbox
`bv:component`	**Optional** (encouraged)	Maps it to a specific Backstage **component** for more precise ownership

Tags catch drift between declared ownership and what's deployed

- Backstage records what developers *say* they own; tags record what’s actually running.
- An Application with critical findings but no live assets (discovered through tags) is not as risky as one that has internet connected assets.

Where the integration breaks

Matching cloud assets to component by name was a guess

Team: Security

Component: `tomcat` → rule: `*tomcat*`

True positive: `security-team-tomcat-test`
False positive: `data-science-team-tomcat-test`

Now imagine: `*api*` · `*worker*` · `*web*`

Let owners tell us with tags

`bv:system` → **Application** *(required: every tagged asset has this)*

`+ bv:component` → **Component** *(optional but more precise)*

`+ bv:environment` → **Environment** *(environment categorization)*

A tag is a statement from the owner and is something that can be standardized and updated retroactively

Untagged assets have potential oversight risk

- **Code-only apps are low risk.**
- **“Deployed” depends on tags.**
- **The blind spot: SCPs only help going forward.**
- **Keep an eye on untagged assets.**

Making it reliable enough to prioritize on

- Using exact bv:* tag matching in place of wildcard matching
- Notifying teams to update untagged resources
- Prevent the creation of untagged resources using org-level service control policies
- Treating Backstage as the source of truth for ownership and asset discovery

Context answers four business questions

Environment

Prod, qa, staging, sandbox, or dev?

Deployment

Actively deployed assets or dormant repo?

Criticality

How critical is this app to the business?

Ownership

Who owns it today, not two years ago?

AWS component tags mirror the Backstage catalog

Backstage

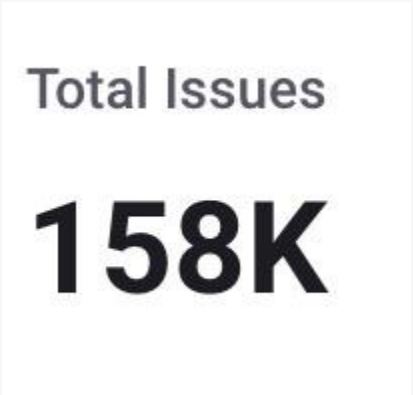
What developers say they own

AWS tags

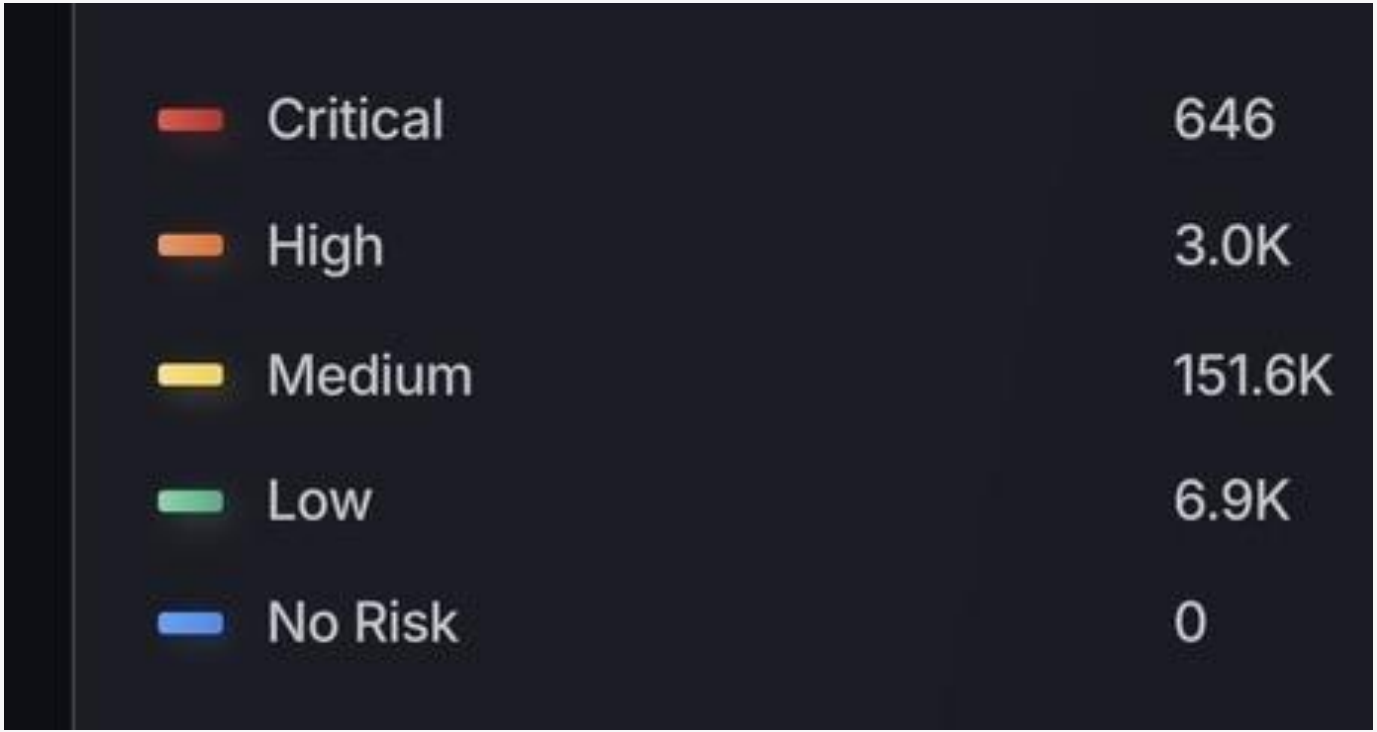
Ownership and environment at the infrastructure layer

Before: raw code findings by severity

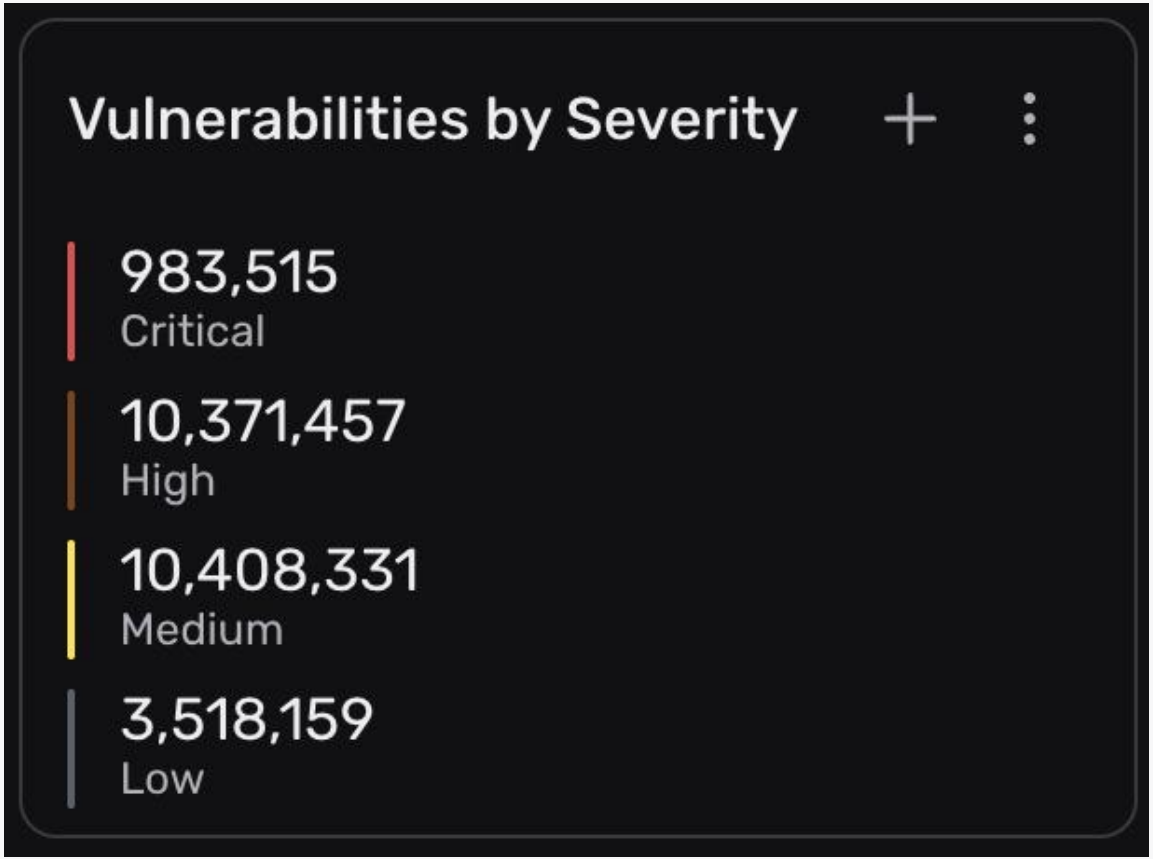




After: raw code findings by severity



Before: raw cloud findings by severity



After: total code + cloud findings



Less than 4% of scanner "criticals" stay critical with business context

< 4%

of findings scanners call critical stay critical

PART 3

What we got wrong

mauvehed & Nick Eckert

We didn't communicate the vision broadly enough

18 months is a long time to keep stakeholders aligned

Stakeholder and partner expectations were misaligned

Tool usage and expectations were misunderstood

Process and procedures were being redefined on the go

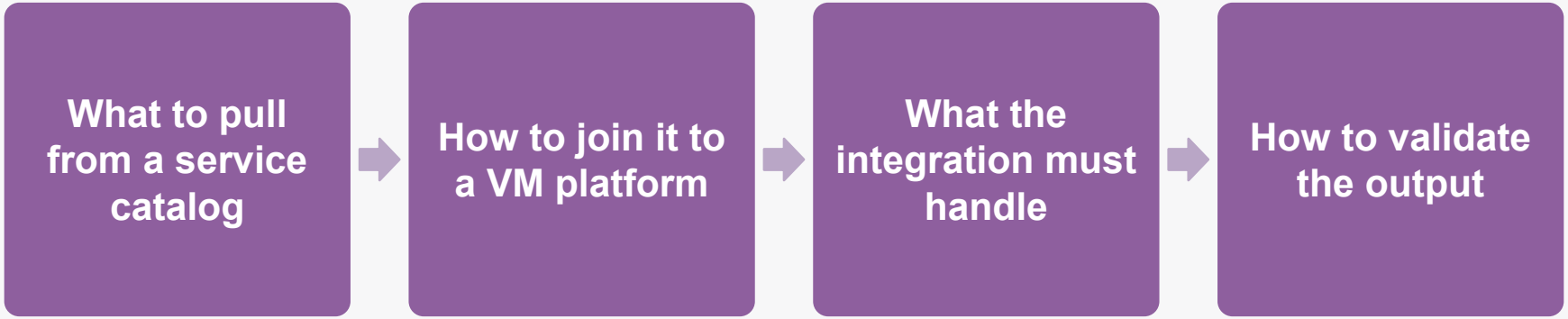
The technology worked; program management is the next problem

Tighter integration with program/project management

Greater visibility at management/principal levels

Better understanding and buy-in/support from partners

Practitioners: the pattern works beyond Phoenix and Backstage



Leaders: security can't solve the metadata problem alone

- Business context is the missing ingredient in most VM programs
- **The metadata problem cannot be solved unilaterally by security**
- Align developers, infrastructure, and security around a shared inventory of truth

Questions?

mauvehed - nate@mvh.dev

Nick Eckert - n.eckert70@gmail.com



HIRE MAUVEHED