TROOPERS

# Attacking and Defending Kubernetes Cluster with **KubeHound**, an Attack Graph Model


# $ whoami

## Julien Terriac

Team Lead, Adversary Simulation Engineering (ASE)
*Repented pentester*

# $ cat /etc/group

## Jeremy Fox

Staff Security Engineer @Oracle
*Repented Datadog engineer :sad-panda:*

## Edouard Schweisguth

Senior Security Engineer, Adversary Simulation Engineering (ASE)
*Repented pentester*

# Agenda


01 Introduction

02 The Problem Space

03 The Solution

04 KubeHound In Action

05 Introducing KHaaS

06 Under the hood

07 Development Process Retro

08 Future Vision

# Introduction

Kubernetes, graphs and their combined power

# Kubernetes 101

## Kubernetes

Open-source container orchestration platform

- Automates the deployment, scaling, and management of **containerized applications**
- High availability and auto-scaling

## Container

Lightweight, standalone, and executable software packages

- Encapsulate an application and its dependencies
- **Sandboxed** execution

## Pod

Smallest **deployable unit** in Kubernetes

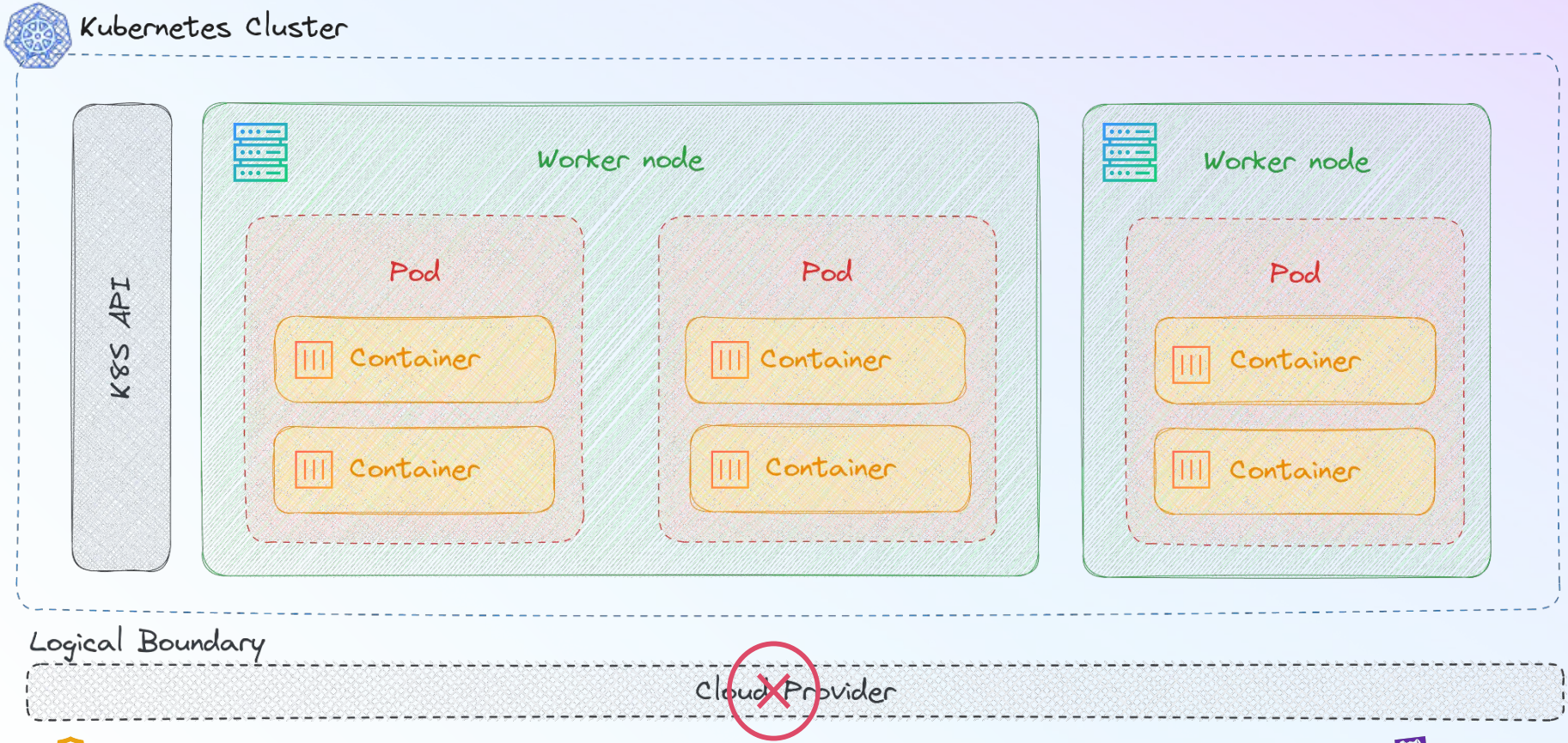
- Contain one or more containers that share the same network namespace and storage volumes
- Designed to run a single instance of an application and are scheduled to *nodes*

## Node

Worker **machines** within a Kubernetes cluster

- Host *pods* and provide the necessary resources (CPU, memory, storage) for running containers
- Grouped together in a **cluster**

# Kubernetes 101



# Kubernetes Security 101

### Container escape

Exploit a container misconfiguration to gain node access

- Multiple avenues
- Very **powerful** - grants access to all node resources

### Kubernetes Identity

Define **service accounts** (robot), users (humans) and groups (both)

- Service accounts linked to pods

### Kubernetes Roles

Set of permissions granted to an identity on specific resources

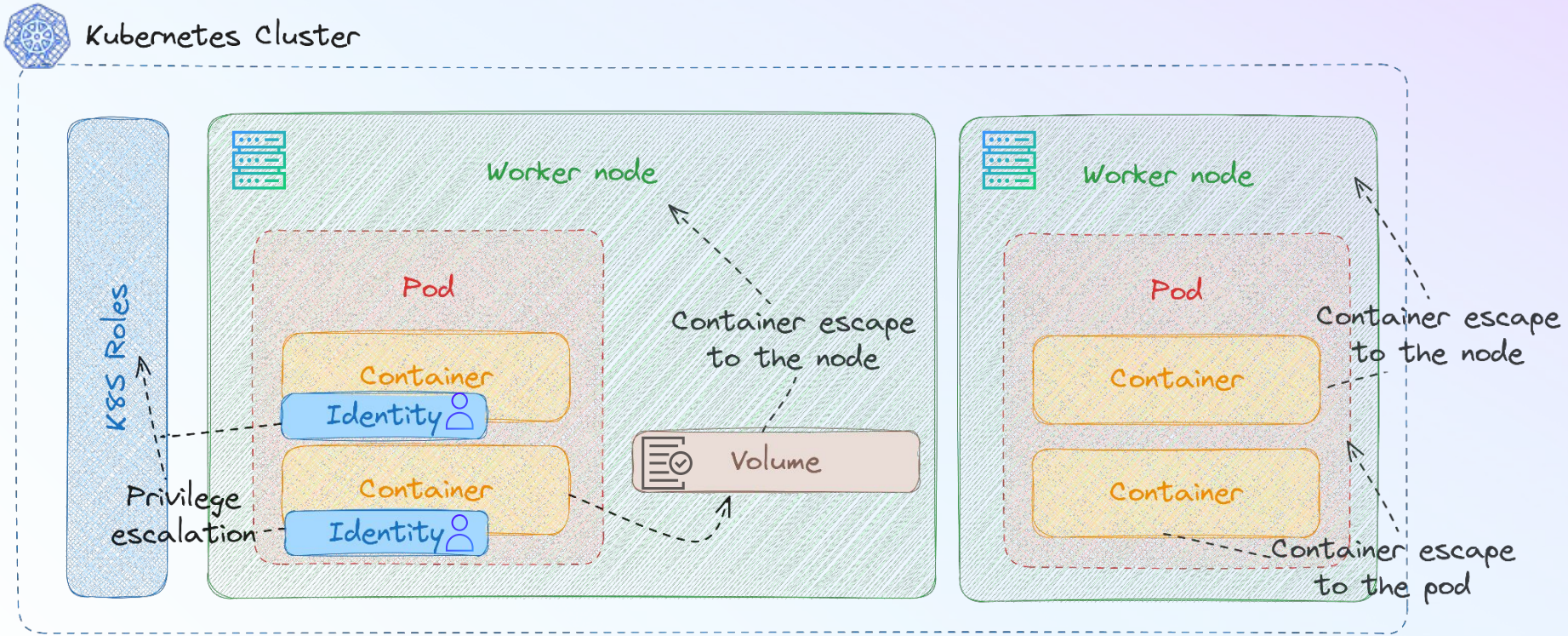
- Addition only (**no deny**)
- Certain permissions are very **powerful** - *secrets/list, pods/exec, etc.*

### Mounted Volumes

Node or “projected” directories can be mounted into the container

- Mounting the wrong directory = **container escape**
- Projected directories contain service account **tokens**

# Kubernetes Security 101



Of course **there are a lot more attacks path** but we will not have time to cover all of them …

# The Problem Space

Scale, complexity and quantifying security

# Vulnerability Context

Manual processing takes time

**FINDING: Container escape**

Web application exposed to the internet running inside a container with `privileged: true`

- Internet facing
- Privilege is not necessary
- Limited auditing

**FINDING: Container escape**

Control plane DNS container running with `CAP_SYS_MODULE` enabled

- Internal service
- Restricted, audited access
- Privilege is necessary

# Can you do it at scale ?

# Let’s play a game …

*Let’s assume we have a cluster with …*

**14** **container escapes** are present in my kubernetes cluster.

---

**32** **privilege escalations** through RBAC issues.

---

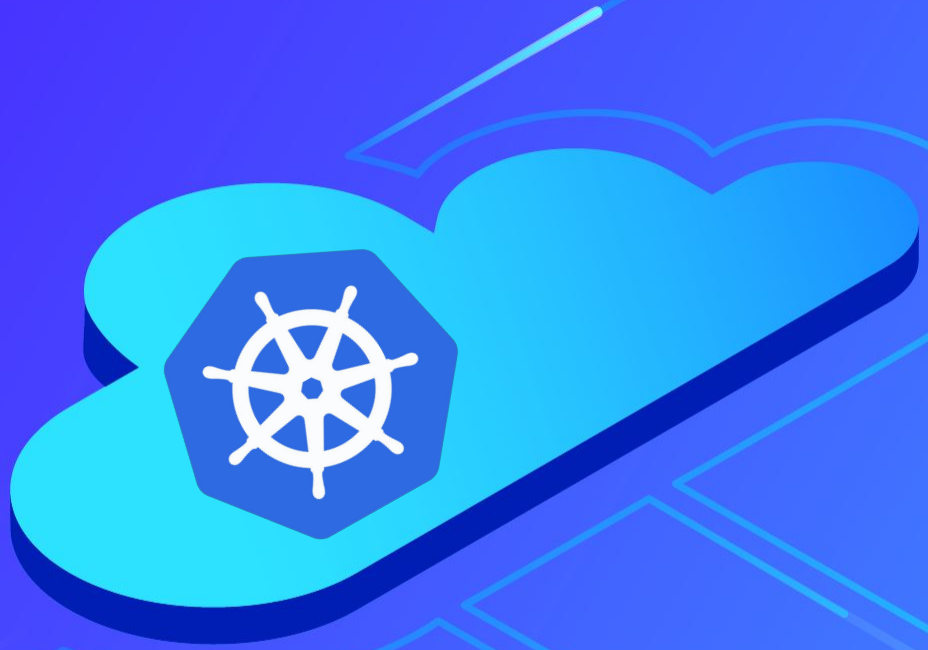
**34** **escape to host t**hrough weak vulnerables volumes configurations.

---

**72** **lateral movement** between containers (Share Process Namespace for instance)

? 🤷 ?

**How secure** is this cluster ?
(on scale  1 to 10)



April 26, 2015

**John Lambert**

Corporate Vice President, Security Fellow, Microsoft Security Research

TROOPERS

> “**Defenders think in lists, attackers think in graphs**; as long as this is true, **attackers win**.”

# Need to Quantify a Security Posture

The old way

# List approach

How many vulnerabilities ?

How many misconfiguration ?

How many outdated/CVE ?

The new way

# Graph approach

Public facing ?

Can have the most significant impact on my cluster security ?

Lead to a critical attack path ?

# Quantifying Security Posture

If you cannot measure it, you cannot improve it

**Current state**

What is the **shortest exploitable path** between an internet facing service and cluster admin?

What **percentage of internet-facing services have an exploitable path** to cluster admin?

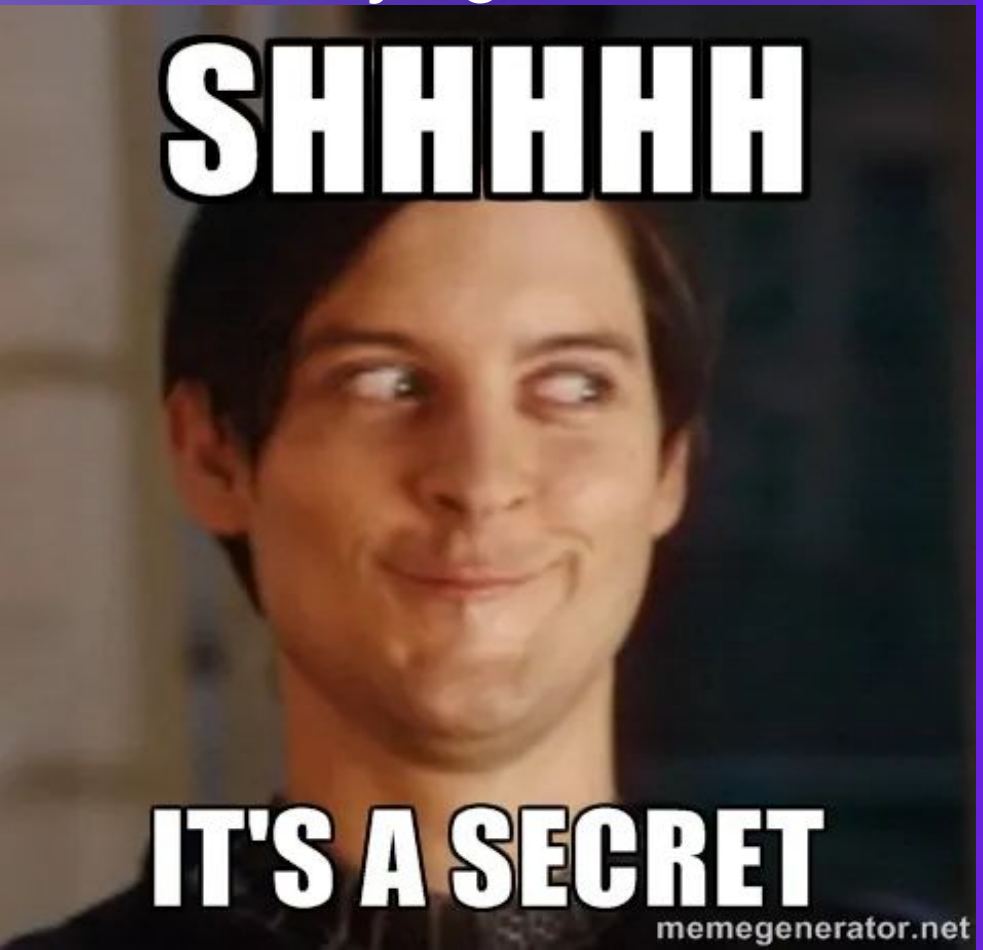
---

**Measuring Change**

What **type of control would cut off the largest number of attack paths** in your cluster?

By what percentage did the introduction of a security control reduce the attack surface in your environment?

# Quantifying at scale at Datadog …




Datadog environment is **vast**:

- **“tens of thousands of nodes"**
- **"hundreds of thousands of pods"**
- **“multi-cloud”**

Traditional **penetration testing does not scale** to this level.

# Demo

## Security metrics calculation


LET'S PRAY
TO THE DEMO GODS
imgflip.com

# Quantitative Analysis of Security Posture

Demo time

Can we use KubeHound to answer the question of “how secure is my cluster” and track that metric over time?

- ✅ Quantifying security posture
- ✅ Democratising offense (reducing from days to instant findings)
- ✅ Exhaustiveness at scale (finding all of the attack paths)

# The Solution

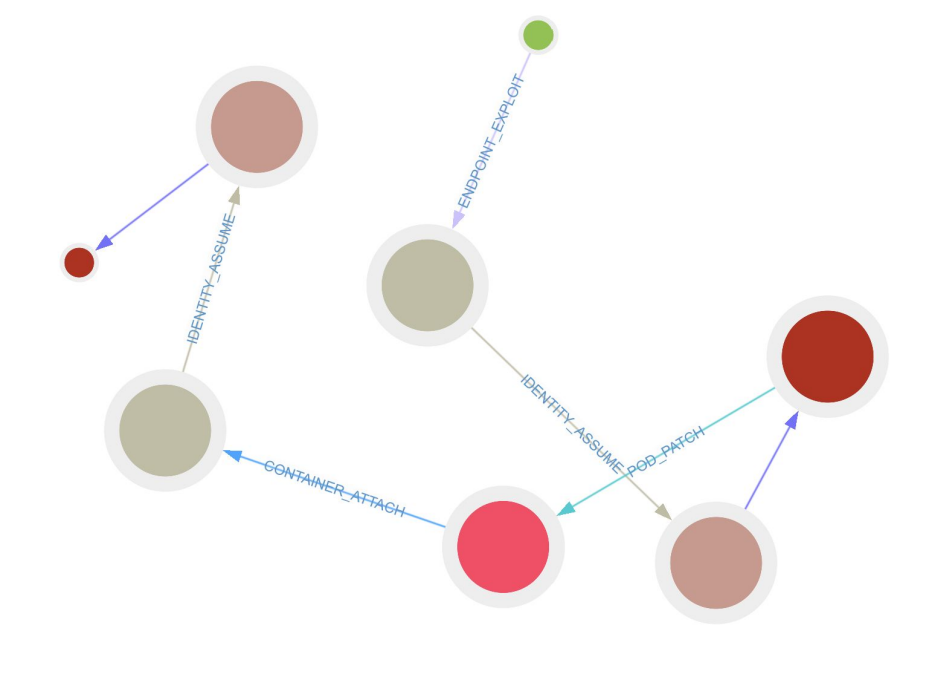
Graph theory + **Off**ensive **Sec**urity = KubeHound

# Graph Theory I

Sorry about that …


# Graph Theory I

Bla bla bla




> **The study of graphs, mathematical structures used to model pairwise relations between objects.**

*Wikipedia*

# Graph Theory 101

Taxonomy is always important

### Graph

A data type to represent complex, relationships between objects.

- In KubeHound: a Kubernetes cluster at a specific time

### Vertex

The fundamental unit of which graphs are formed (also known as "node").

- In KubeHound: containers, pods, endpoints, nodes, permissionsets, identity and volumes

### Edge

A connection between vertices (also known as "relationship").

- Automates In KubeHound: a container escape (e.g CE_MODULE_LOAD) connects a container and a node

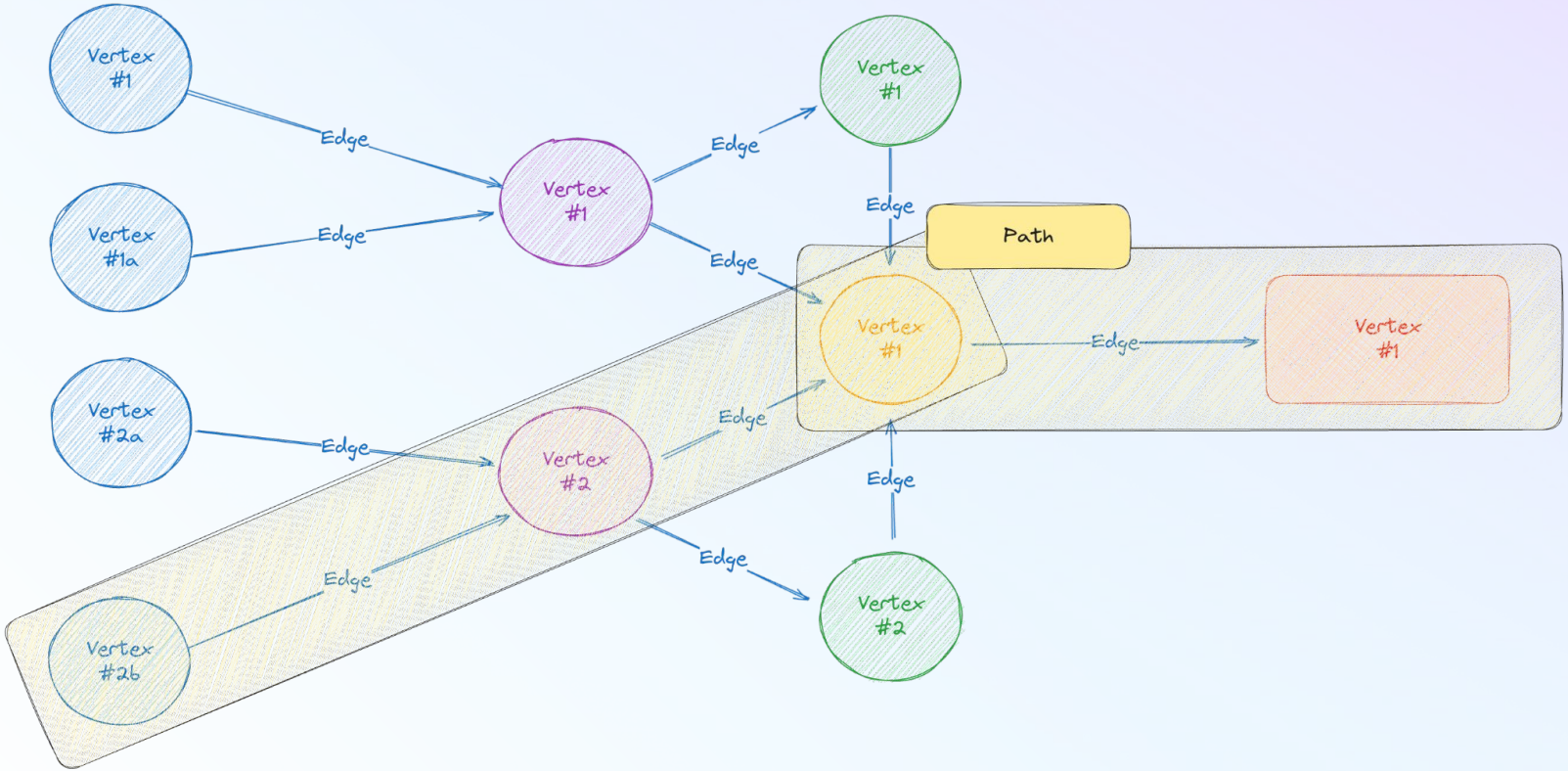
### Path

A sequence of edges which joins a sequence of vertices.

- In KubeHound: a sequence of attacks from a service endpoint to a cluster admin token

# Graph Theory 101

Sample graph



# Why/What is KubeHound ?

Yet another tool …

### What is the goal of KubeHound ?

The aim of KubeHound is to identify security gaps and real attack vectors using a **graph** to visualize **attack paths** presents in a Kubernetes cluster.

### Why create KubeHound ?

Current Kubernetes auditing tools output security information from clusters in a “list”. There are no links between findings. They cannot produce an attack path like **BloodHound**, which **changed the game of Windows Domain security**.

# KubeHound 101

Taxonomy is always important

### Entity

An abstract representation of a Kubernetes component that form the vertices of the graph.

- For instance: PermissionSet is an abstract of Role and RoleBinding.

### Critical Asset

An entity in KubeHound whose compromise would result in cluster admin (or equivalent) level access

- For now it only covers a subset of roles which are not namespaced (like `cluster-admin` or `kubeadm:get-nodes`).

### Critical Path

A set of connected vertices in the graph that terminates at a critical asset.

- This is the treasure map for an attacker to compromise a Kubernetes cluster.

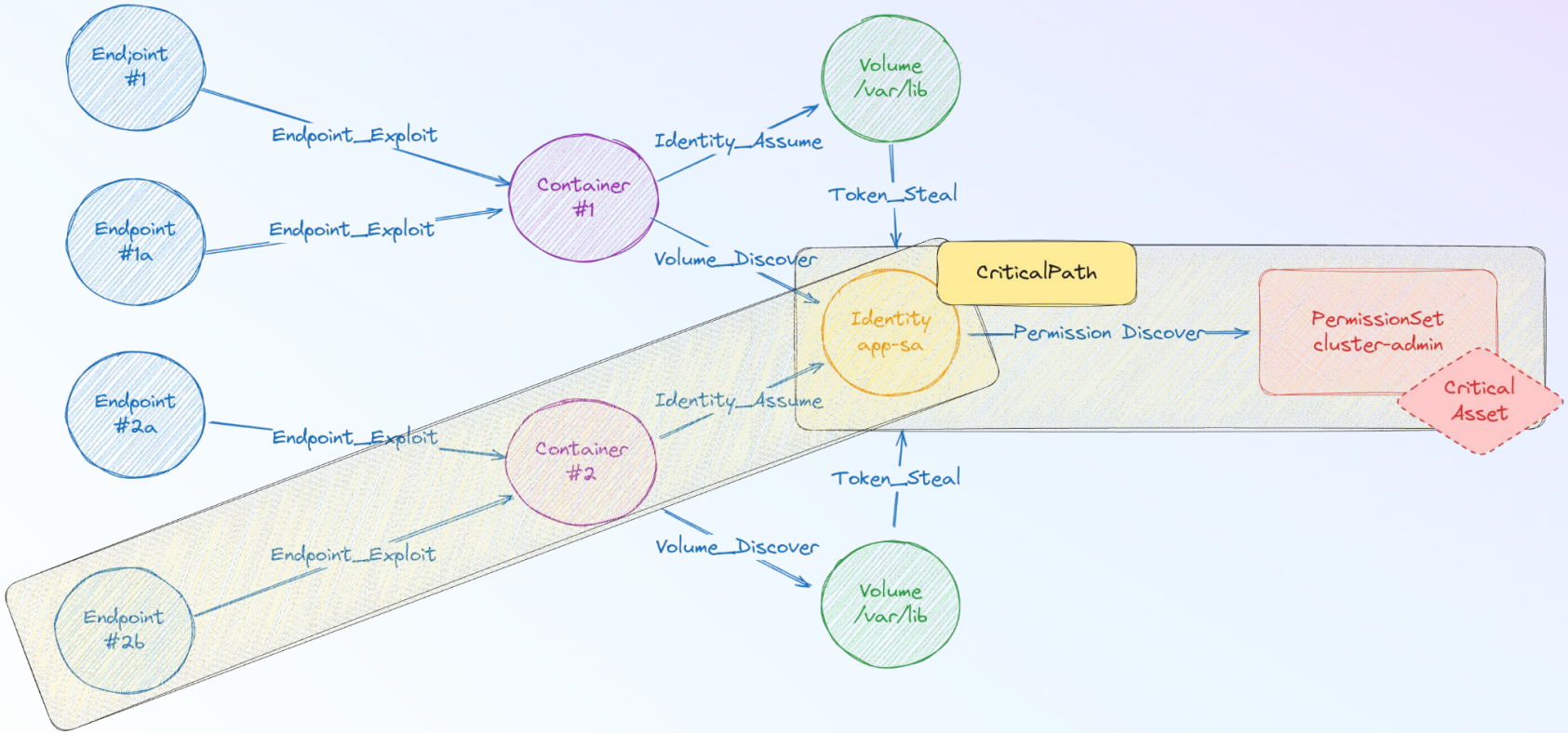
### Attacks

All edges in the KubeHound graph represent attacks with a net "improvement" in an attacker's position or a lateral movement opportunity.

- For instance, an assume role is considered as an attack.

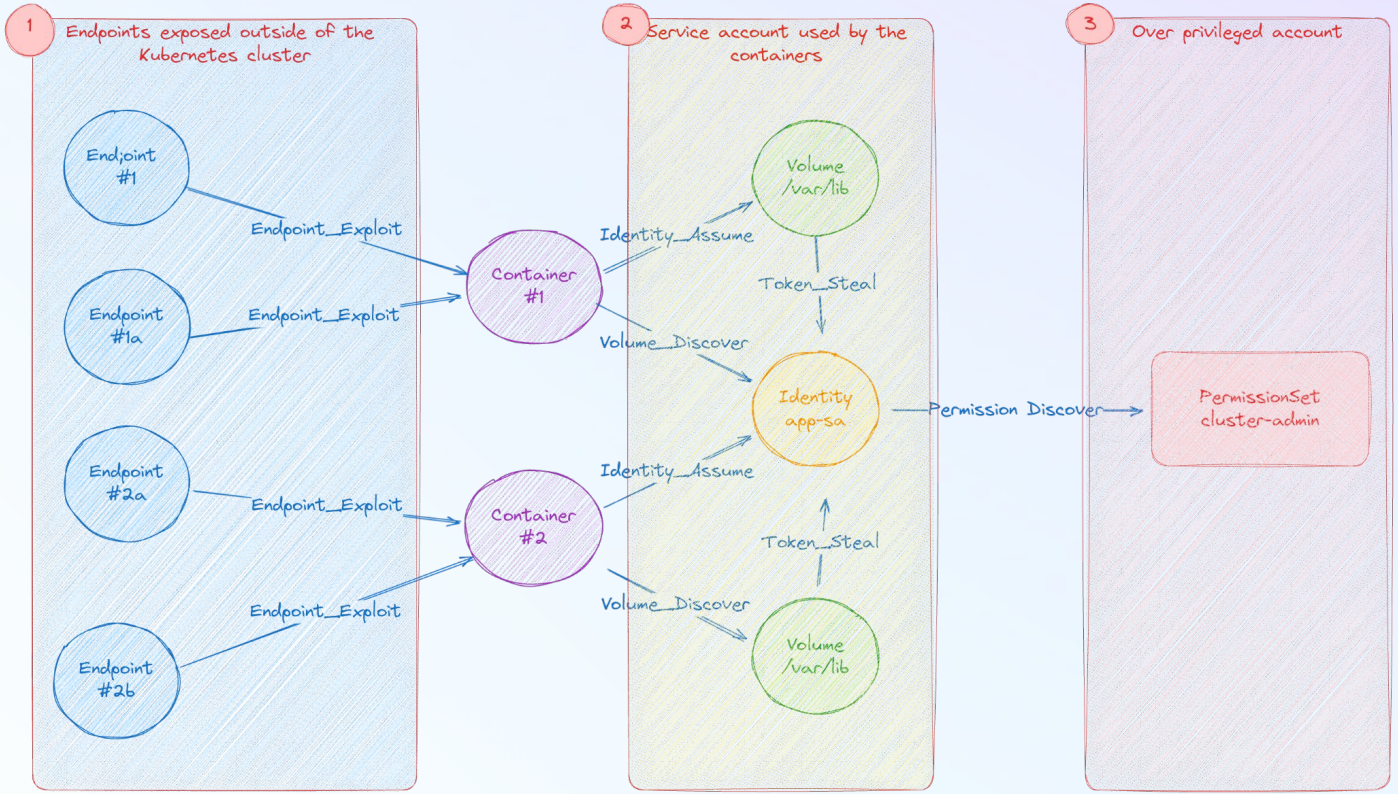
# Attack Graphs

Sample graph



# Attack Graphs

Sample graph



# KubeHound in a nutshell

The best defense is a good offense

### Attack Graph

KubeHound creates a graph of attack paths in a Kubernetes cluster, allowing you to identify direct and multi-hop routes an attacker is able to take, visually or through graph queries.

### Runtime Calculation

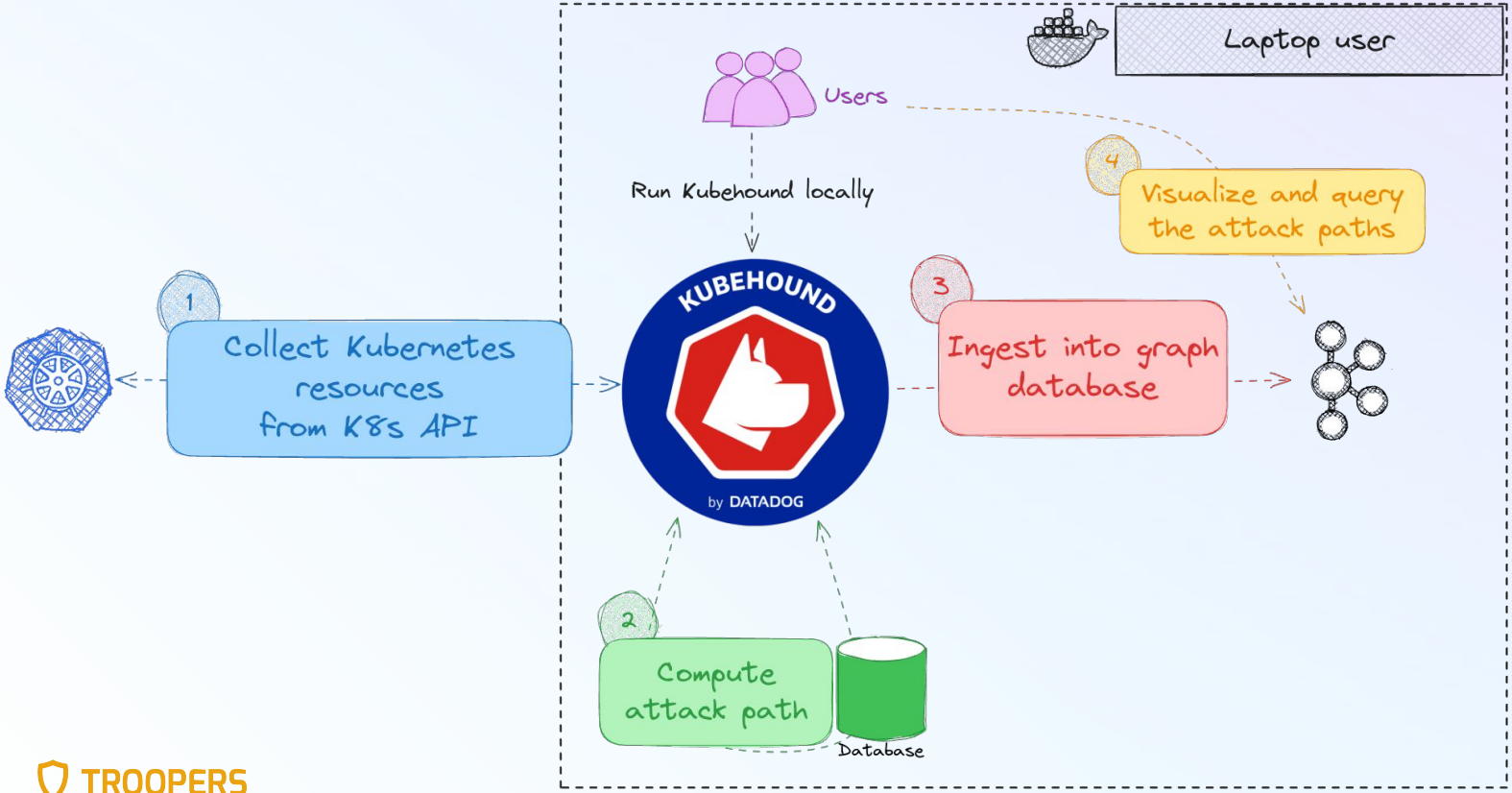
If any entity is connected to a critical asset in our attack graph - a compromise results in complete control of  the cluster.

### Snapshot

KubeHound analyze a snapshot of your Kubernetes cluster. It dumps all the assets needed to create an “image” of it.

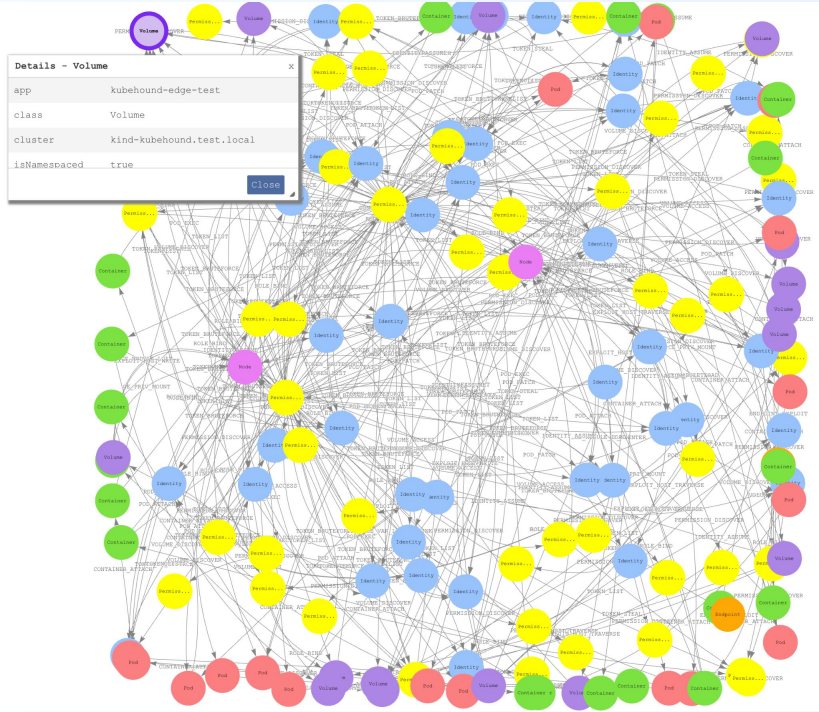
# KubeHound in a nutshell

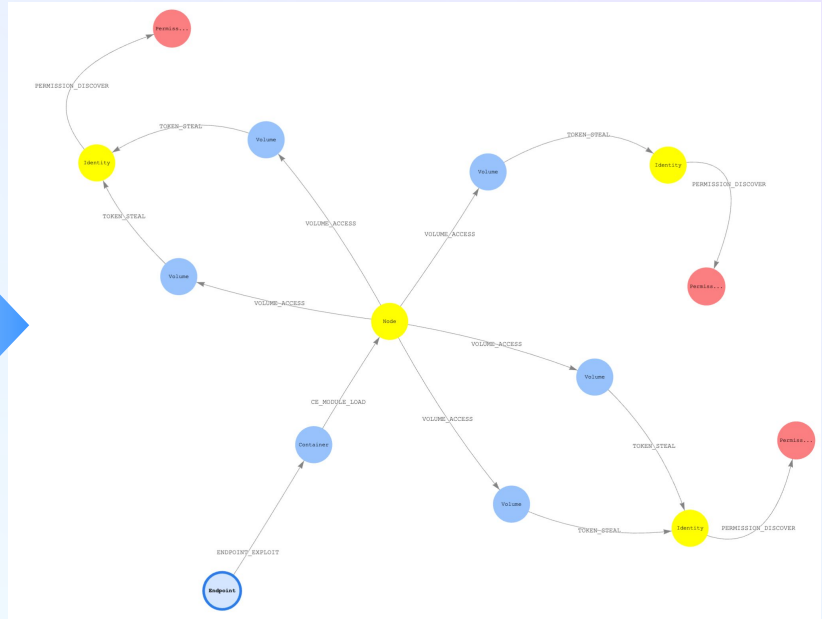
A diagram is worth a thousand words



# KubeHound in a nutshell

Pinpoint where the security failures are.



# KubeHound in Action

Capability showcase

# User Experience (UX)

Gremlin a tough query language

### A really powerful language …

All k8s data is being ingested into Janusgraph which is powered by Gremlin a powerful query language.

```
g.V().hasLabel("Pod").dedup().by("name")
```


### … but really hard to master

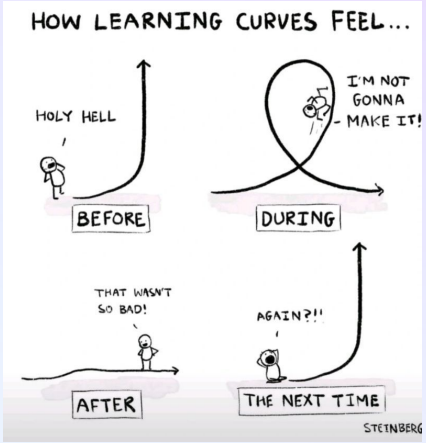
```
g.V().hasLabel("Pod").dedup().by("name")
.repeat(outE().inV().simplePath()).until(
hasLabel("Container").or().loops().is(10).or().
has("critical", true)
).hasLabel("Container").path().tail(local,1).va
lues("name").dedup()
```

# KubeHound DSL

UX above all

In order to improve the User Experice (UX) we **developed a custom Domain Specific Language (DSL)** on top of the Gremlin language.

The DSL has more than **20 custom wrappers** that allow a user to generate attack paths really easily.




**Raw Gremlin request**

```
g.V().hasLabel("Pod").dedup().by("name")
.repeat(outE().inV().simplePath()).until(
loops().is(10).or().has("critical", true)
).has("critical",true).path()
.by(elementMap()).limit(100)
```

**KubeHound DSL equivalent**

```
kh.pods().criticalPath().limit(100)
```

# Full doc

**https://kubehound.io/queries/dsl/**

***All DSL queries are described with proper examples.***

**Query Library**
KubeHound DSL
Metrics
Sample queries

# KubeHound DSL

The KubeHound graph ships with a custom DSL that simplifies queries for the most common use cases

```
// Example returning all attacks from containers running the cilium 1.11.18 image
kh.containers().has("image", "eu.gcr.io/internal/cilium:1.11.18").attacks()
```

## Using the KubeHound graph

The KubeHound DSL can be used by starting a traversal with `kh` vs the traditional `g`. All gremlin queries will work exactly as normal, but a number of additional steps specific to KubeHound will be available.

```
// First 100 vertices in the kubehound graph
kh.V().limit(100)
```

## KubeHound Constants

### Endpoint Exposure

Represents the exposure level of endpoints in the KubeHound graph

```
// Defines the exposure of an endpoint within the KubeHound model
public enum EndpointExposure {
  None,
    ClusterIP,                    // Container port exposed to cluster
    NodeIP,                       // Kubernetes endpoint exposed outside the clu
    External,                     // Kubernetes endpoint exposed outside the clu
}
```

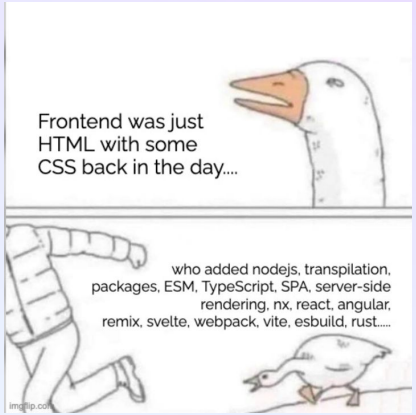
Table of contents
Using the KubeHound graph
KubeHound Constants
Endpoint Exposure
Traversal Source Reference
Run Step
Cluster Step
Containers Step
Pods Step
Nodes Step
Escapes Step
Endpoints Step
Services Step
Volumes Step
HostMounts Step
Identities Step
SAS Step
Users Step
Groups Step
Permissions Step
Traversal Reference
Attacks Step
Critical Step
CriticalPaths Step
CriticalPathsFilter Step
HasCriticalPath Step
MinHopsToCritical Step
CriticalPathsFreq Step

# KubeHound UI

Why did frontend development become so complicated?

We tried to avoid creating a fancy/Minority report style UI. **Focus most of our energy on backend and performance**, because we are not frontend developers.

Frontend development is hard, really hard …




## KubeHound v1.0

**Cons**:

- Not free anymore
- Lack of prebuilt queries
- Developers oriented
- Not available as a Service (rich client only)

## KubeHound v1.3

Pros:

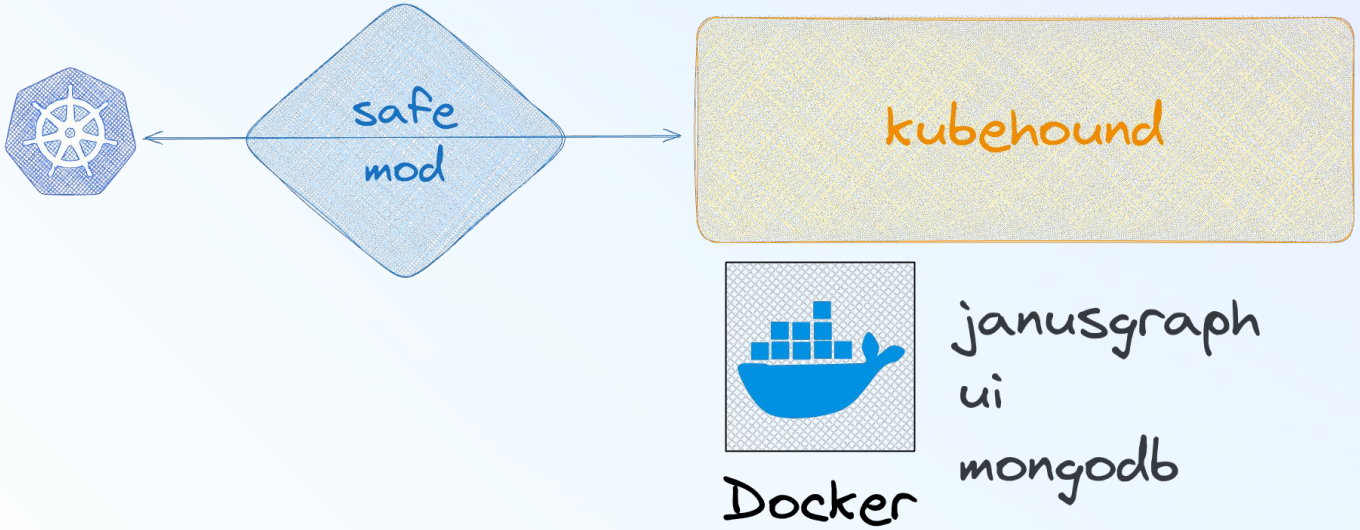
- Share results
- As a Service frontend
- Highly customizable
- Prebuilt queries through notebooks

# Auto mode (new)

Who does not like auto-pilot ?

**Only one binary and one command**

For local usage just do *./kubehound* and enjoy the result on *127.0.0.1:8888*



# Demo

## From can of worms to critical vulnerability

Fr lity

# From can of worms to critical findings

Demo time

Can we use KubeHound to pinpoint where are the most critical vulnerability and therefore help the remediation team as much as the attacker ?

- ✅ Vulnerability context
- ✅ Democratising offense (reducing from days to instant findings)
- ✅ Exhaustiveness at scale (finding all of the attack paths)

# Under the hood

How does this magick happen ?

# Simple architecture

Taxonomy is always important

## Collector

Collect all Kubernetes objects needed to create the attack path

- There is no filtering (collecting raw elements)
- Multiple input support:
  - k8s API collector
  - File collector
  - etcd collector (not implemented yet)

## Ingestor

Pull the data from the collector and ingest them in the database (mongodb for now)

- Parallelized ingestion if no explicit dependencies

## Builder

Query the database to build the graph

- Build the vertices, the “node” representing the elements of the cluster (pod, role, …)
- Build the edges, the relation representing the attacks
  - CE_NSENTER
  - POD_CREATE
  - …

# Example:

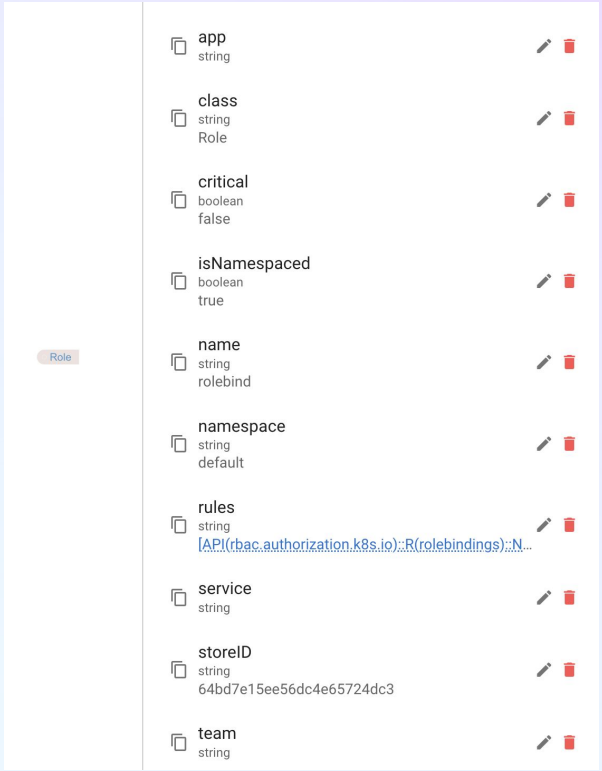
How the data is being processed

```yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: rolebind
  namespace: default
rules:
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["rolebindings"]
    verbs: ["create"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles", "roles"]
    verbs: ["bind"]
    resourceNames: []
```

yaml k8s config file

```
_id: ObjectId('64bd7e15ee56dc4e65724dc3')
name: "rolebind"
is_namespaced: true
namespace: "default"
▾ rules: Array (2)
  ▾ 0: Object
    ▾ verbs: Array (1)
        0: "create"
    ▾ apigroups: Array (1)
        0: "rbac.authorization.k8s.io"
    ▸ resources: Array (1)
      resourcenames: null
      nonresourceurls: null
  ▾ 1: Object
    ▾ verbs: Array (1)
        0: "bind"
    ▾ apigroups: Array (1)
        0: "rbac.authorization.k8s.io"
    ▾ resources: Array (2)
        0: "clusterroles"
        1: "roles"
      resourcenames: null
      nonresourceurls: null
▾ ownership: Object
    application: ""
    team: ""
    service: ""
```

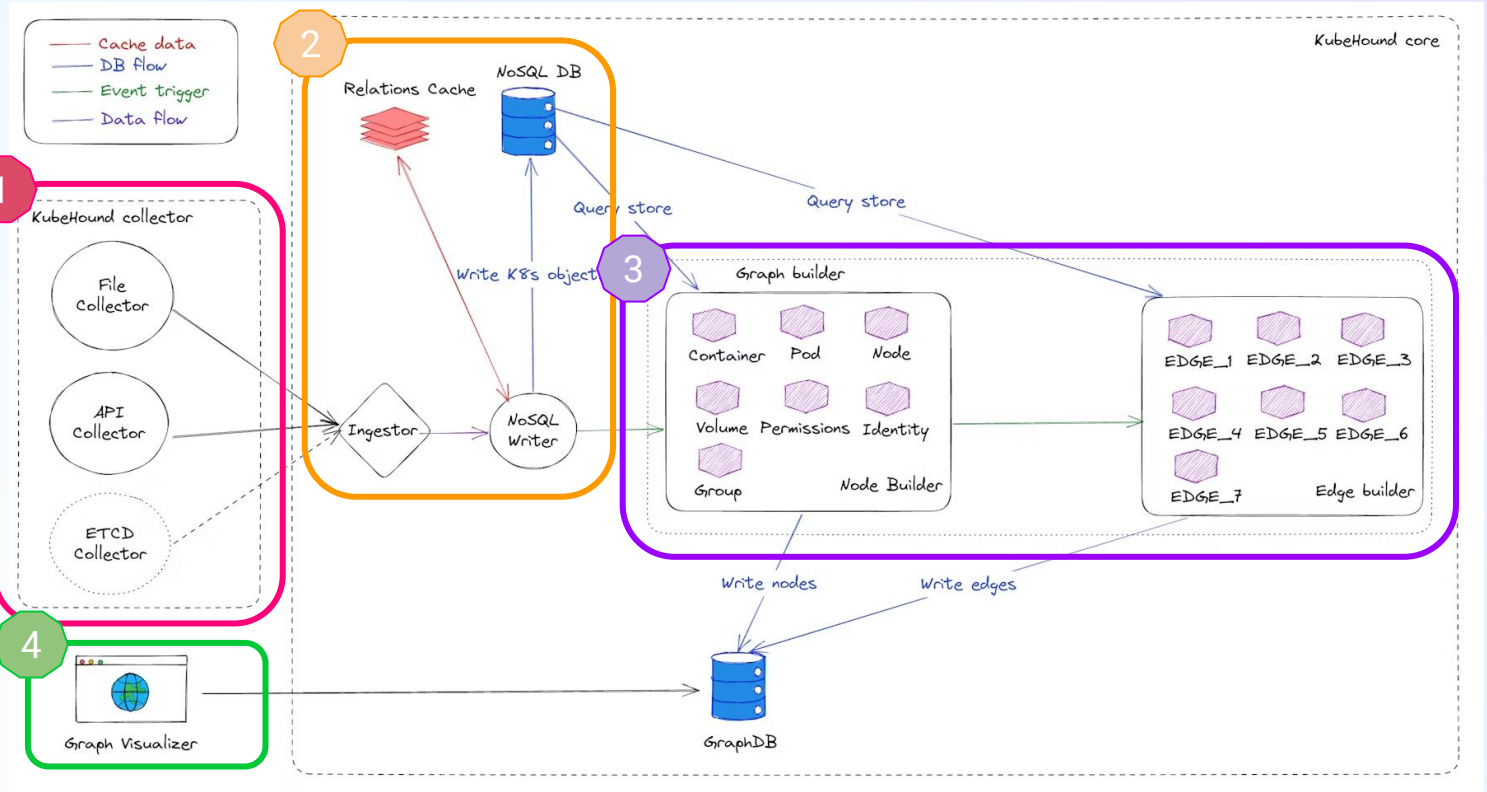
mongodb



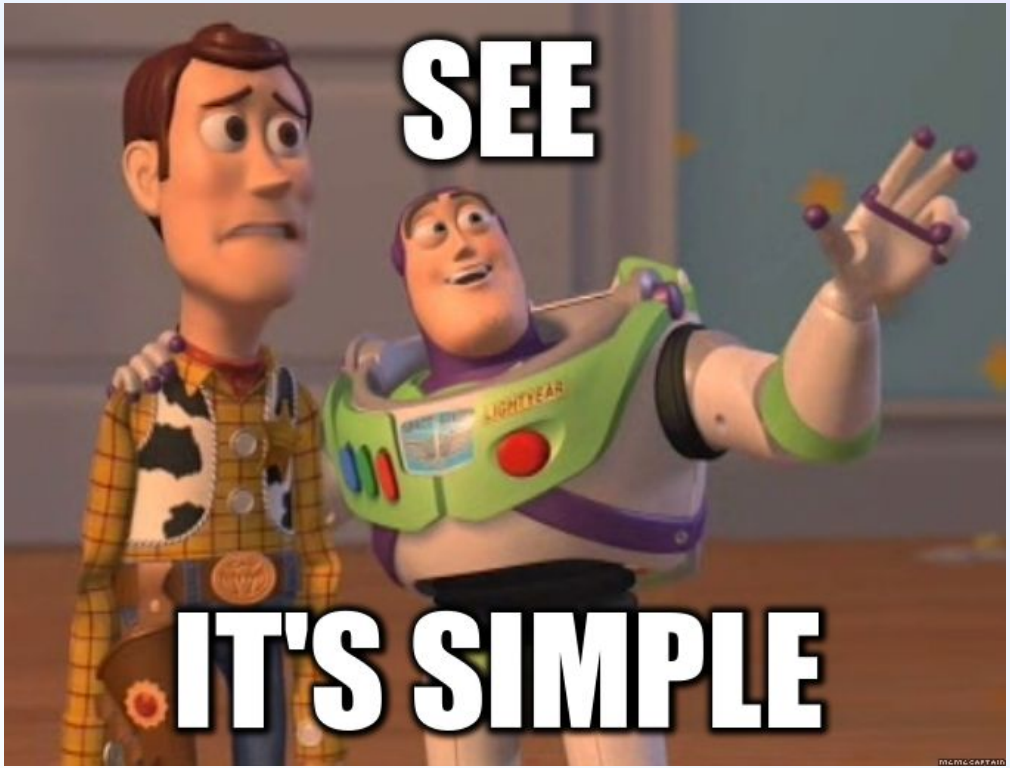

janusgraph

# Full architecture

Almost everything

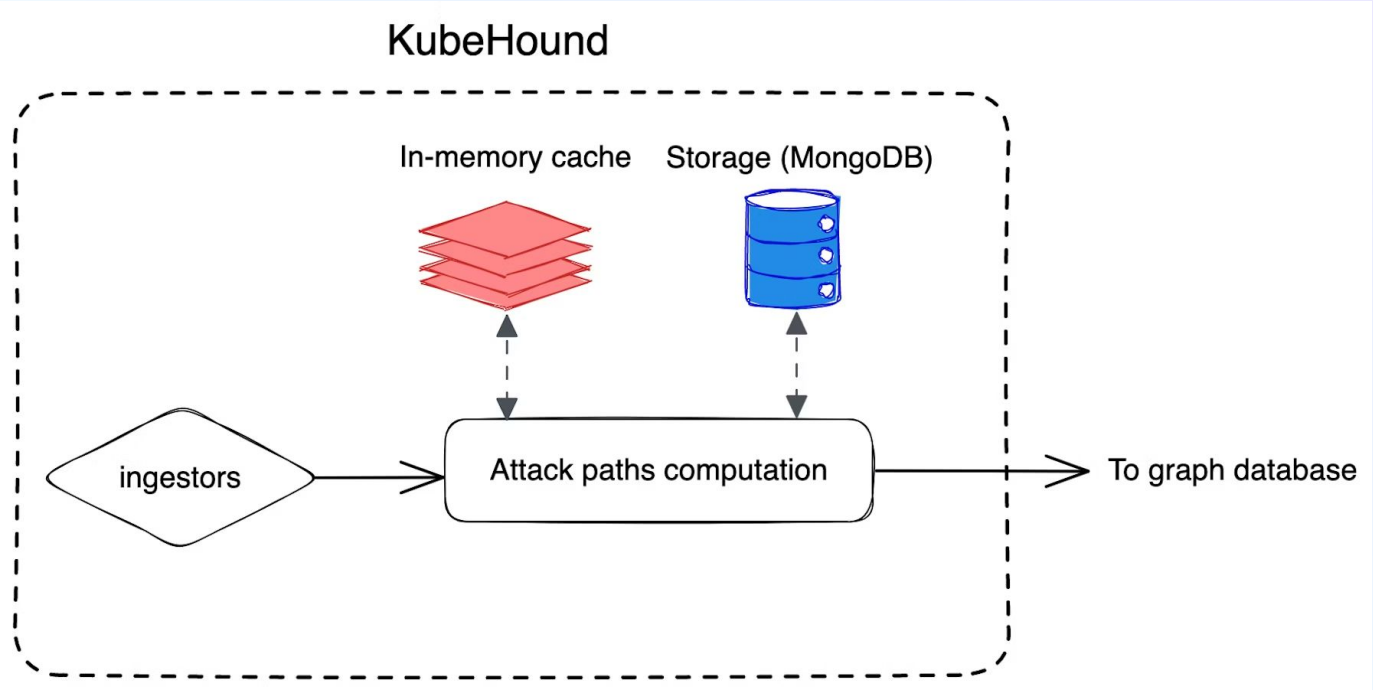


# Full architecture



# Summarized architecture

Less is better



# K8s API collector - Safe to use :)

API rate limit (100 req/sec)

---

Buffer page size limited (10mb)

---

Number of element per page limited (500)

Good to have a **rich application** but it **does not really scale** well

...

# Introducing KHaaS

**K**ube**H**ound **a**s **a** **S**ervice

# How can use KubeHound at a big scale ?

# What need to be changed ?

From one to many

### RBAC

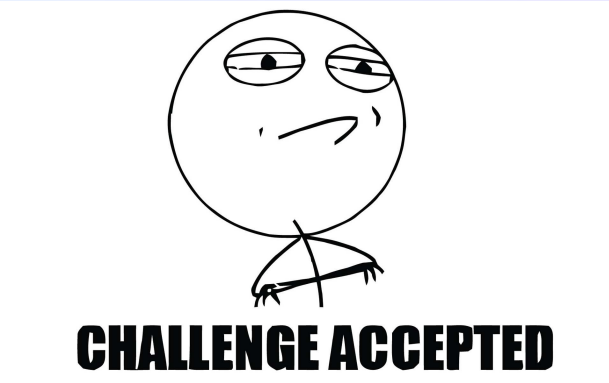
Service account with limited right to access only the k8s resources needed.

### Deployment

Has to be a job that can be easily scheduled and does not impact the Kubernetes cluster.

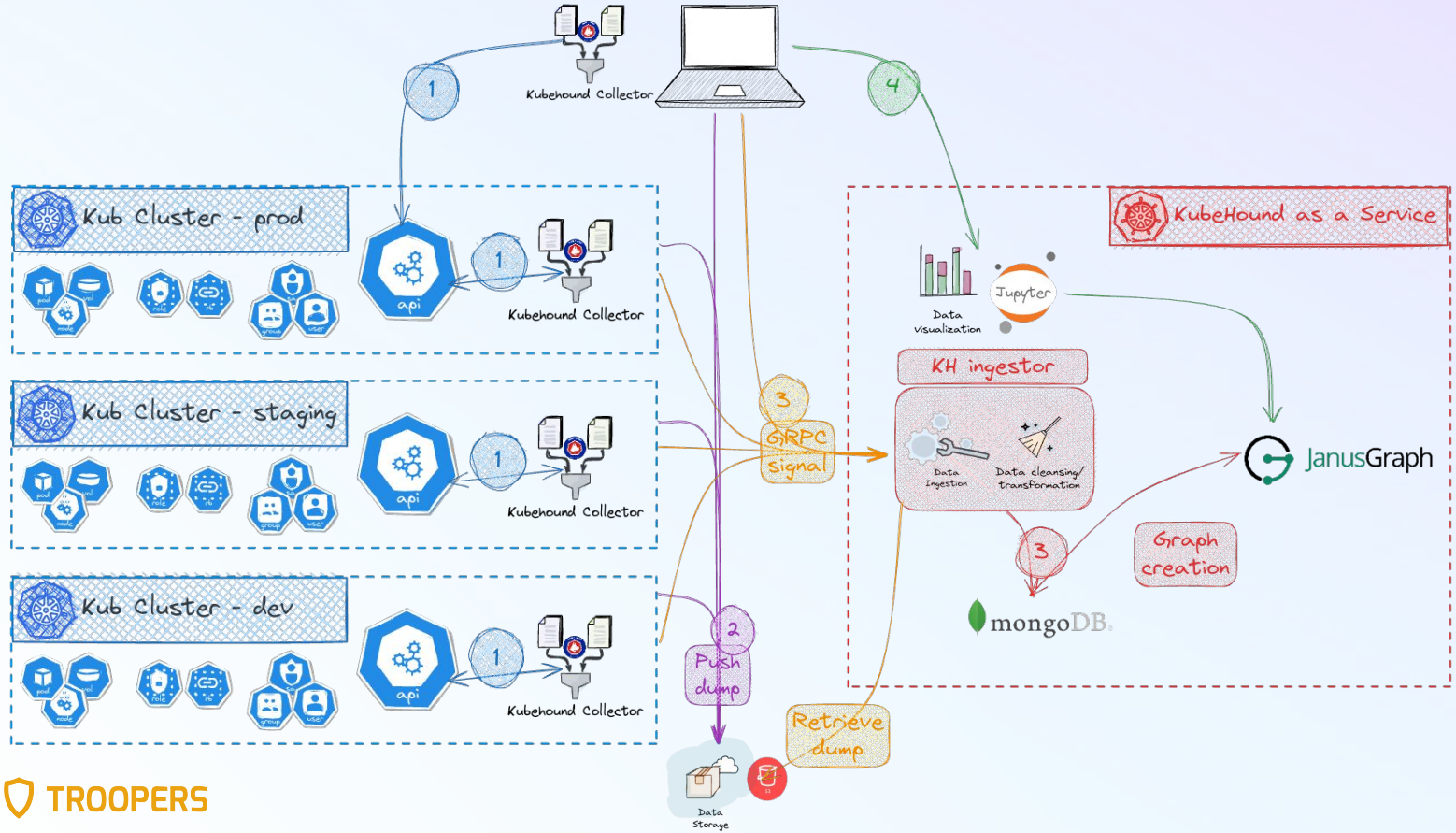
### Scalable

Can handle “unlimited” clusters to generate automatically its security posture.



# Kubehoud as a Service (KHaaS)

Scale architecture



# KubeHound as a Service is …

**Distributed collector**

**+**

**Centralized ingestion processing**

**+**

**Unified source of information**

# Some metrics in Datadog

**60gb** To **gain performance** we are using **memory only backend** for Janusgraph. It can holds all Datadog clusters.

---

**20cpu** Total numbers of CPU used in production to process all the data, from the ingestor to the Databases.

---

**10gb** The **size of all daily snapshots** in our s3 bucket. The k8s resources compressed well.

---

**1min** Average time to rehydrate a dump into KubeHound as a Service.

# How **KubeHound** can be used by security consultant ?

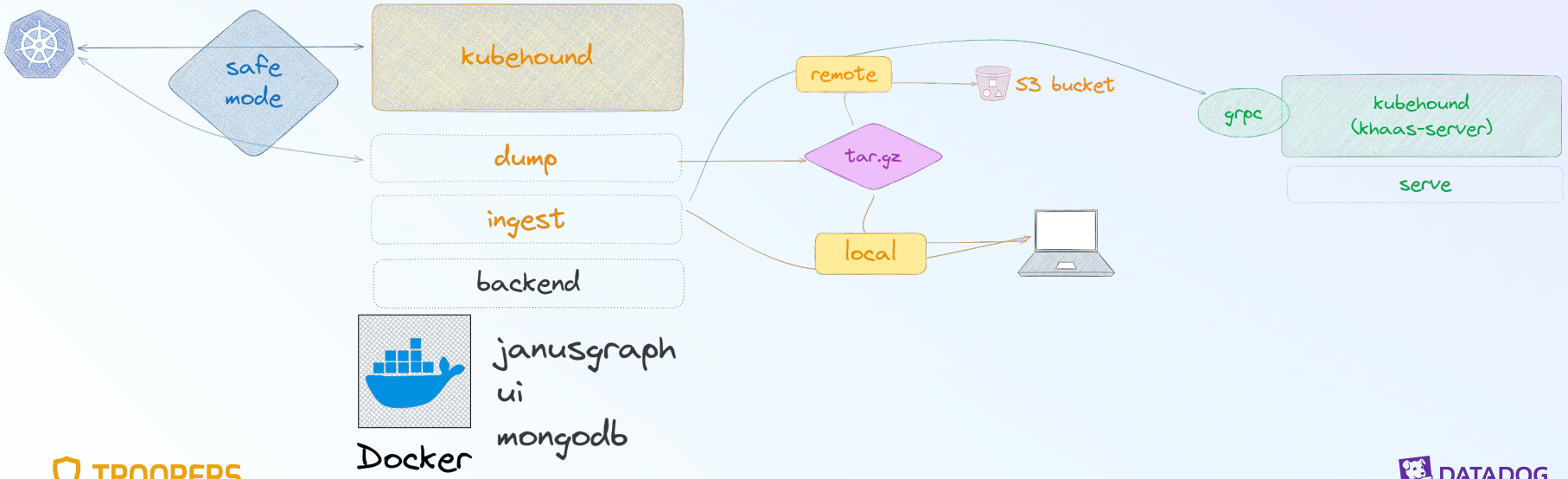
# Asynchronous usage

Home sweet home

**Snapshot a cluster and rehydrate it locally easily**

You can create a snapshot with KubeHound dump local/remote.

Reload the data using KubeHound ingest local/remote.



# Development Process

Research, design, implement, iterate

# Why am I talking about this?

Powerful approach

**The approach I will outline can be applied to create attack graphs of any systems (AWS, Hashicorp Vault, …)**

Step #1: **Research**

Collate, ingest and categorize all the Kubernetes security research.

Step #2: **Design**

Sketch attack components (vertice needed ? properties ?)

Step #3: **Implementation**

Port to graph database

# i.e. RBAC

> “Compromising Kubernetes Cluster by Exploiting RBAC Permissions”
>
> CyberArk @ RSAC 2020

23 blog articles

12 Hours of Youtube

12 PoC

RESULTS

**With the study of RBAC attacks, we added 11 attacks in KubeHound’s model.**

## Research

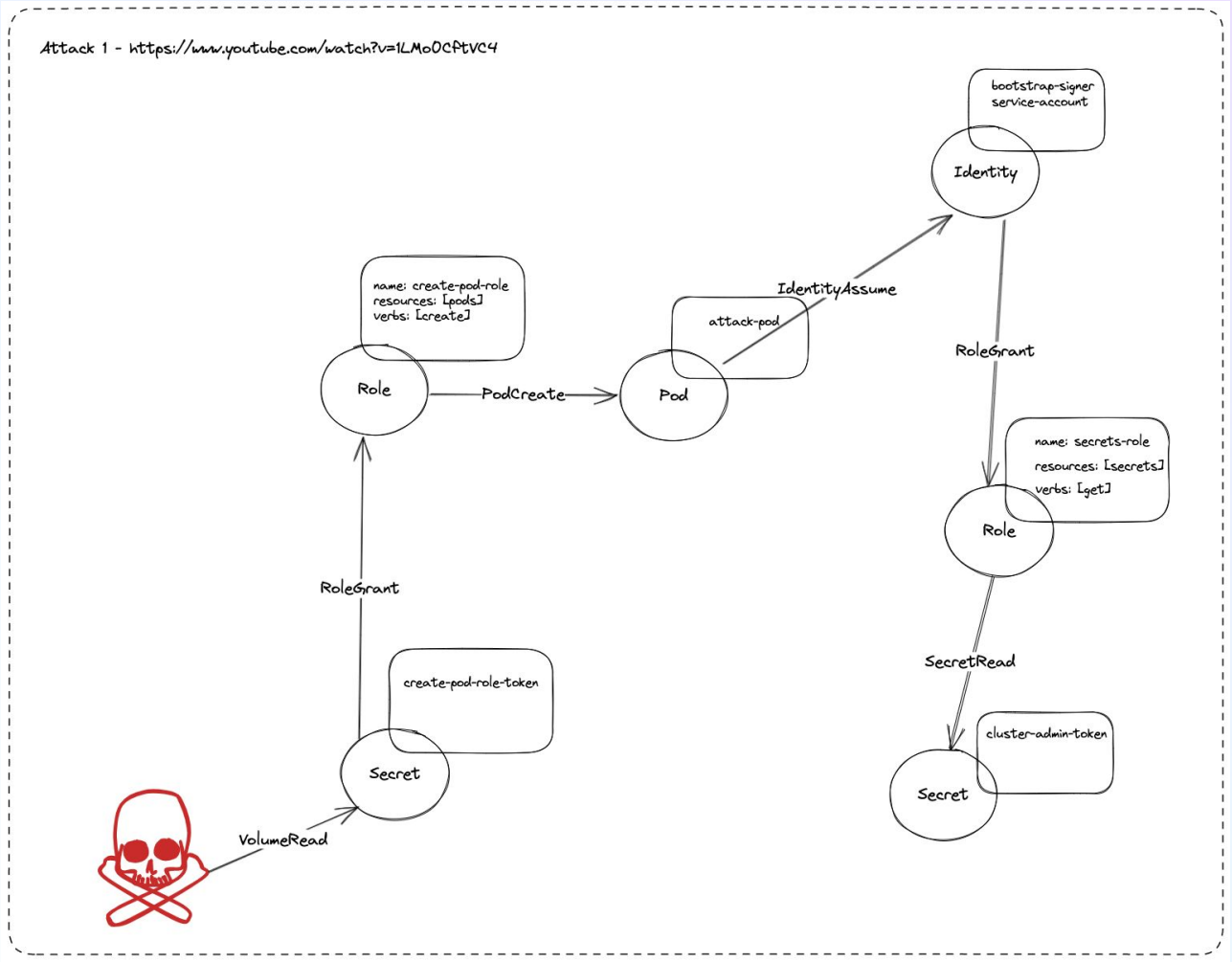
Reading a lot the official Kubernetes documentation and PoCing locally to test our assumptions.

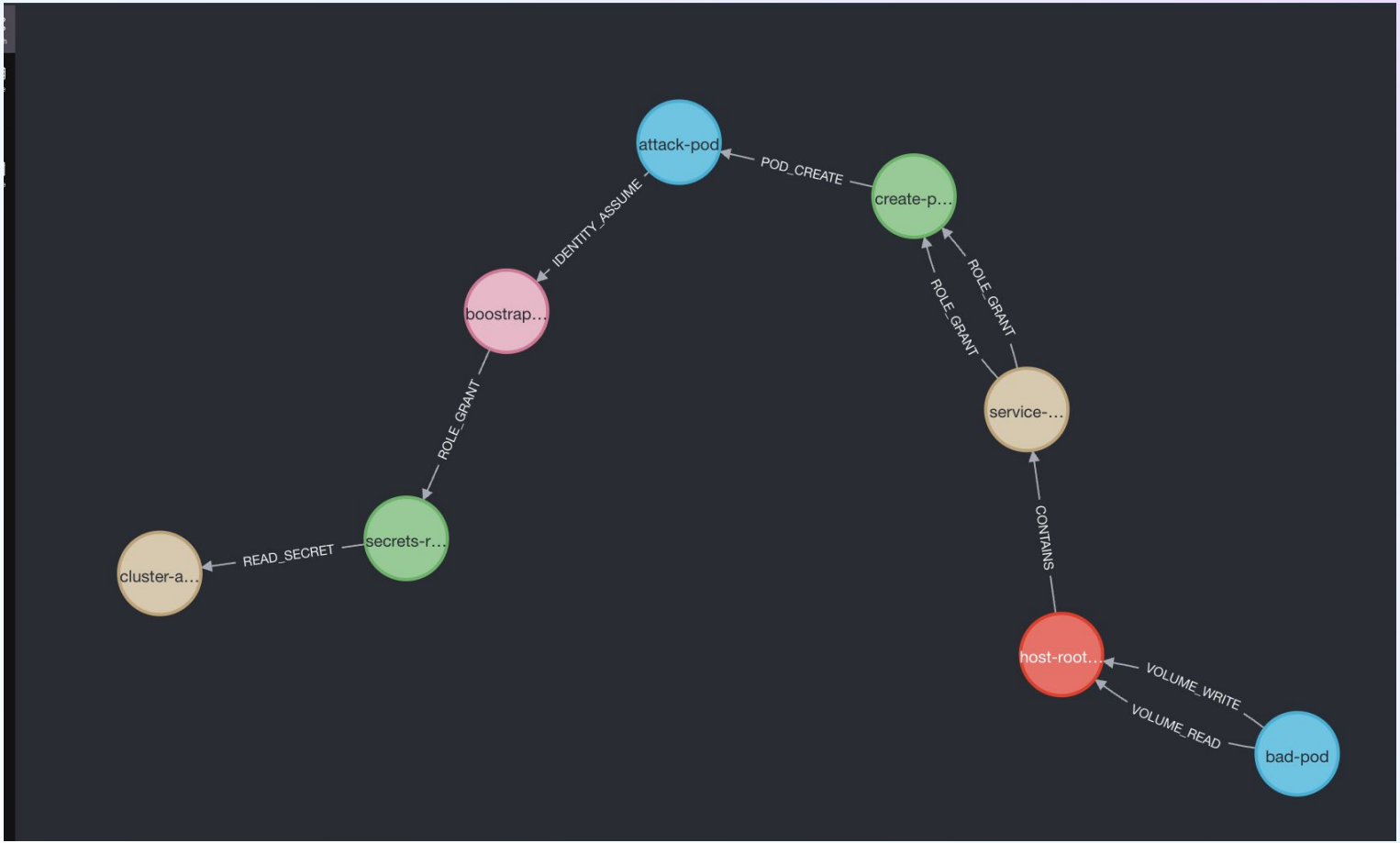
## Design

Create a specific abstraction to describe role and rolebinding: PermissionSet

## Implement

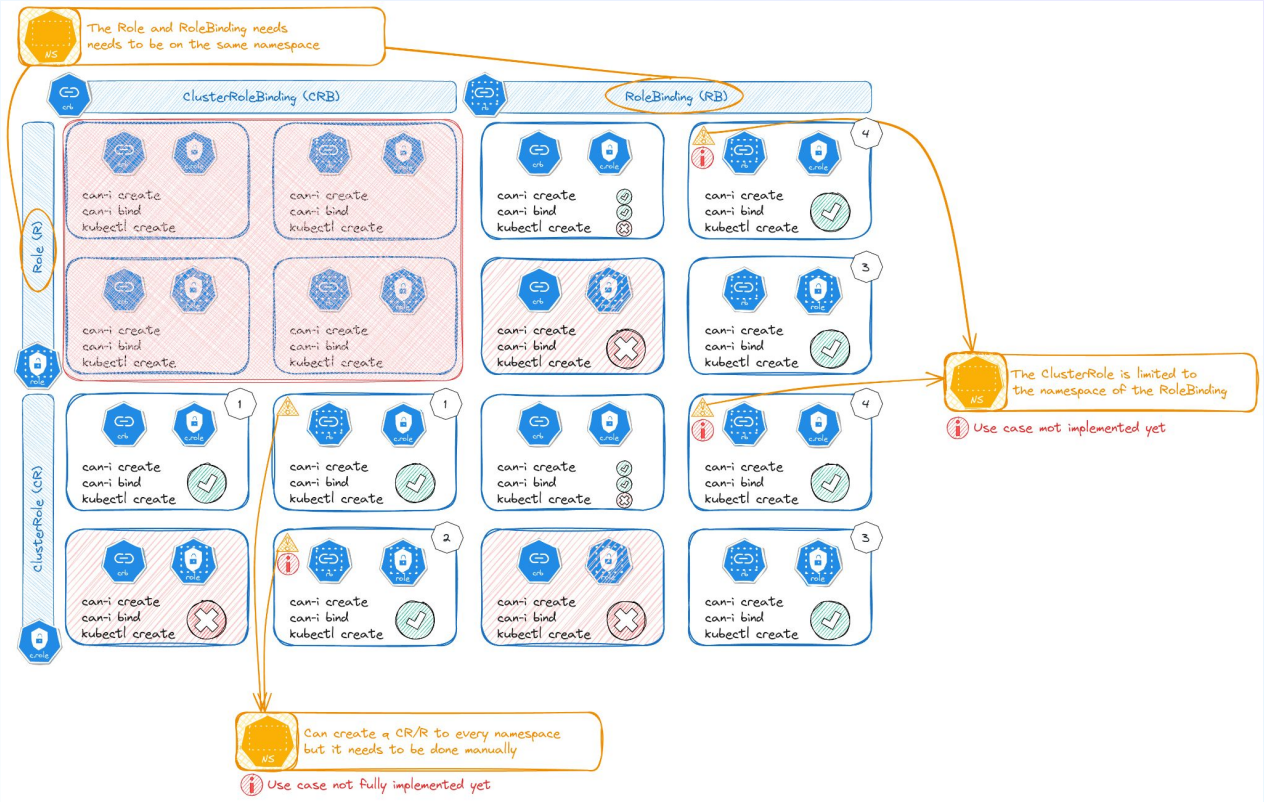
Port to graph database


Attack 1 - https://www.youtube.com/watch?v=1LMo0CftVC4
bootstrap-signer service-account
Identity
name: create-pod-role
resources: [pods]
verbs: [create]
attack-pod
IdentityAssume
RoleGrant
Role
PodCreate
Pod
name: secrets-role
resources: [secrets]
verbs: [get]
Role
RoleGrant
SecretRead
create-pod-role-token
cluster-admin-token
Secret
Secret
VolumeRead


attack-pod
POD_CREATE
create-p...
IDENTITY_ASSUME
boostrap...
ROLE_GRANT
ROLE_GRANT
ROLE_GRANT
service-...
secrets-r...
READ_SECRET
cluster-a...
CONTAINS
host-root...
VOLUME_WRITE
VOLUME_READ
bad-pod

# Role bind attacks

Who does love RBAC stuff ?



# How to simulate those attacks ?

Kind cluster to the rescue

**Easy to setup and lightweight**

Kind cluster is an easy and lightweight cluster to deploy locally that runs into Docker. Can replicate a full Kubernetes with multiple nodes on your laptop.

**End-to-end testing for each attacks.**

For each attack studied an associated vulnerable pod/container/roles/endpoints/... was created. Even fake users were provisioned to test the attack from end-to-end.

**... but some limitations**

Even if kind cluster is not an exact replica of a Kubernetes cluster (some edge cases or limitation can be faced on some attacks that involve the kernel like CE_UMH_CORE_PATTERN), it is **sufficient for most of our needs**.

# kubehound.io

The reference table for all Kubernetes Attacks implemented in KubeHound

### Prerequisites

Usually it is a k8s description (for instance pods helm shart). What is needed from a configuration point of view.

- SHARE_PS_NAMESPACE: `shareProcessNamespace: true`

### Exploitation

Full description step by step to exploit the attacks. The content should be sufficient for red or blue team.

- SHARE_PS_NAMESPACE: `/proc/$pid/root`

### Checks

How can I do a live check when I am on a vulnerable container, pod or user ?

- SHARE_PS_NAMESPACE: `ps ax` to find a root process.

### Defences

Lead to mitigate or detect the attacks. Example for least privileges or security policies are also listed.

- SHARE_PS_NAMESPACE: Prevent the use of shared namespaces in pods.

# kubehound.io

26 attacks listed so far, more in the pipe

## Attack Reference

| ID | Name | MITRE ATT&CK Technique | MITRE ATT&CK Tactic |
|---|---|---|---|
| CE_MODULE_LOAD | Container escape: Load kernel module | Escape to host | Privilege escalation |
| CE_NSENTER | Container escape: nsenter | Escape to host | Privilege escalation |
| CE_PRIV_MOUNT | Container escape: Mount host filesystem | Escape to host | Privilege escalation |
| CE_SYS_PTRACE | Container escape: Attach to host process via SYS_PTRACE | Escape to host | Privilege escalation |
| CE_UMH_CORE_PATTERN | Container escape: through core_pattern usermode_helper | Escape to host | Privilege escalation |
| CONTAINER_ATTACH | Attach to running container | N/A | Lateral Movement |
| ENDPOINT_EXPLOIT | Exploit exposed endpoint | Exploitation of Remote Services | Lateral Movement |
| EXPLOIT_CONTAINERD_SOCK | Container escape: Through mounted container runtime socket | N/A | Lateral Movement |
| EXPLOIT_HOST_READ | Read file from sensitive host mount | Escape to host | Privilege escalation |

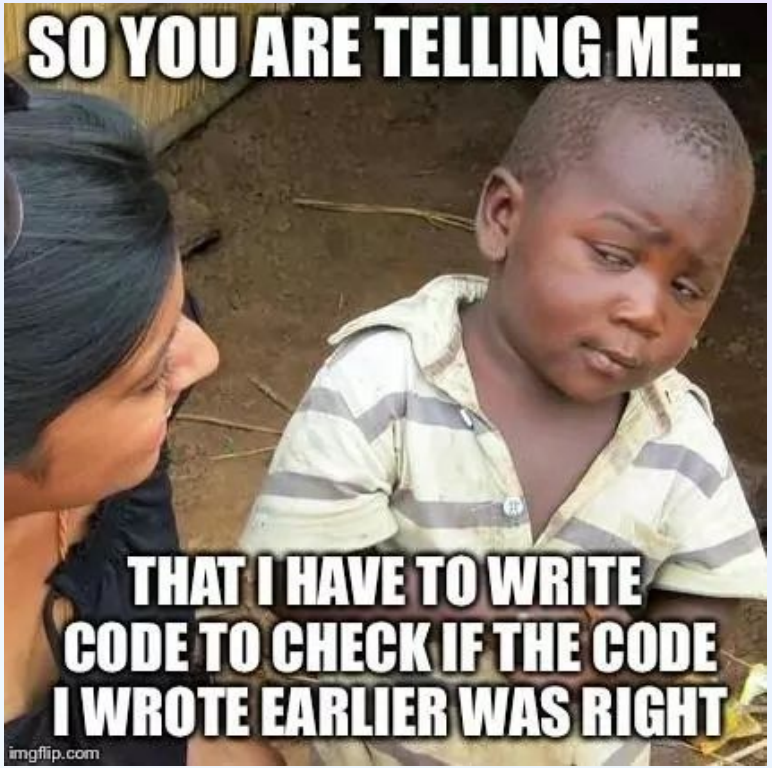
# How can we prevent any regression in our model ?

# Unit tests for the win

Something rare ~~in offsec world~~

**46%**

Coverage in KubeHound core



# Systems tests for the win

The reference table for all Kubernetes Attacks implemented in KubeHound

### Vulnerable kind cluster

Luckily, we can spawn a vulnerable kind cluster with all our attacks listed in kubehound.io reference table.

- **In Github action generated in every PR**.
- Locally for some automated tests.

### Generated code

From the vulnerable kind cluster configuration helm configuration files, we convert them into Golang resources to have **an exhaustive list** of pods, roles, endpoints, ...

### Automated ingestion

Ingest the vulnerable kind cluster like a regular cluster. Building a real graph referencing all k8s objects and associated attack paths.

### End-to-end tests

Run KubeHound/Gremlin queries to check if we have the expected results:

- Vertice: How many attack paths CE_NSENTER ?
- Edges: Do we have all the expected volumes ?
- DSL: Testing our custom queries.

# Fun Fact

## *When your CTO join the party*

PoC

# v0.1

Neo4J based

10 hours to ingest 25k pods

1 hour to dump all objects using a bash script

Ultimate goal set by the team

# v1.0

Full OSS stack

1 hour to ingest 25k pods

10 minutes to dump all k8s objects using only API endpoints

# Set a new standard :)

But …

ALQ / Datadog CTO

“Are we sure about the orders of magnitude? Let's say you have 1,000 nodes in a cluster, each connected to every other node, thus O(10^6) edges. An iPhone runs 6 cores at 2GHz, getting data to and from memory takes O(100) cycles so we should get O(10^7) edges processed by second. There are gross oversimplifications in all this, but the napkin math says that it should be measured in seconds, not hours or days.”

# Performance improvements

There is always a but …

use in memory graph backend

+

tune graph to better optimize for writes

+

optimize queries used to generate edges

+

optimize K8s API querying

**30 sec building graph** (from 35 min)

# Take away

Some insights gleaned on **the power of automated attack graphs** through developing and using KubeHound:

- Provide the ability to quantify security posture and risk
- Scale horizontally to handle any environment
- Act as a force multiplier by sharing the mindset of the best offensive practitioners with defenders

79

TLDR: **Attack graphs change the game and will be the natural evolution of security tooling**

# Future Vision

KubeHound Next Generation

# Customization

Fine tune the model

**Enable tailoring KubeHound to your own environment**

- Custom rules to define critical assets
- Custom inputs to exploitable conditions e.g EXPLOIT_HOST_READ file path
- Custom filters to discard extra data

# Refining the Graph

Compare the different attack paths

**Embed extra information within edges using a weight which defines**

- How easy is the attack to execute?
- How easy is the attack to detect?
- Does the attack require time to execute? (bruteforcing for instance)

# Refining the Graph

Leadership loves KPI

**Enable automated reporting of key metrics and risks**

- Calculate security posture metrics

- Heatmap of critical attack paths

- Consolidate processing errors

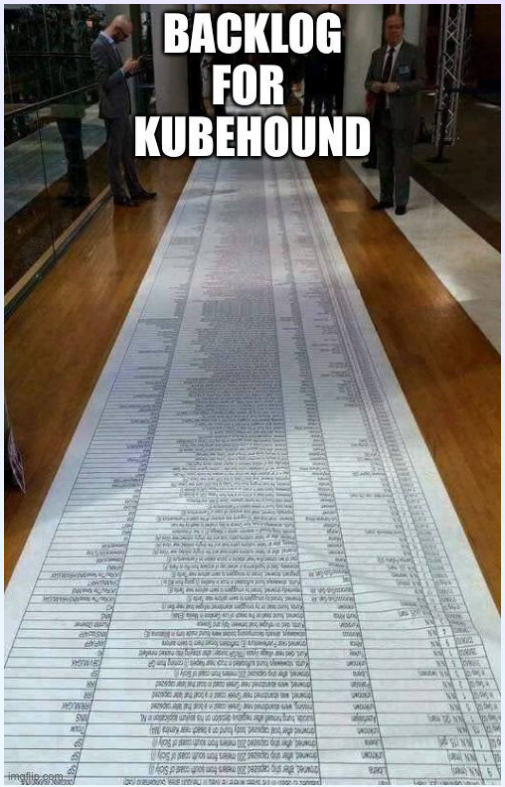
# We have a dream

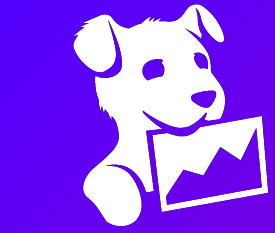
KubeHound roadmap

Create a proper UI to navigate across the results

Diff checker to identify progress between 2 snapshots

Live mode to enable CICD integration (stream data)






# Thank you


kubehound.io

## We are recruiting for the team :)

**Senior Security Engineer - Adversary Simulation Engineering**

Engineering

Paris, France

Join the team!