KubeHound & Beyond:
Evolving security through graphs & automation

Jeremy Fox

March 2024

Jeremy Fox

Jeremy Fox is a cybersecurity specialist with 10 years experience across government and private sector. Following a career change from the finance industry, he developed a wide range of skills in offensive security from reverse engineering and exploit development to red team operations and cloud security.

He is an engineer at heart, having programmed in everything from C/C++ and ASM, through Python and .NET, to Golang. Although his first love was, and always will be, low-level Windows internals, he now works as a Senior Security Engineer at Datadog developing automated offensive security tooling to detect vulnerabilities in large scale cloud environments. His most recent project is KubeHound, an automated Kubernetes attack path calculator.

SPECTEROPS SO-CON 2024

Agenda

Introduction

K8s 101

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 Security 101

Container escape

Exploit a **container misconfiguration** to gain node access

- Multiple methods of attack
- Very **powerful** - grants access to all node resources

Kubernetes Identity

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

- Service accounts define pod identity

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 contain service **account tokens**

The Problem Space

Scale, complexity and quantifying security

Vulnerability Context

FINDING: Overprivileged Container

Web application exposed to the internet running inside a container configured with ***privileged=true***

- Internet facing
- Privilege not required
- Limited auditing

FINDING: Overprivileged Container

Control place DNS container running with ***CAP_SYS_MODULE*** enabled

- Internal service
- Restricted, audited access
- Privilege required for functionality

How secure is your cluster?

On a scale of 1-10

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?

Democratising Offensive Security

Attack is hard!

Table stakes for modern, cloud offensive security assessment:

- In depth K8s security knowledge
- In depth OS security knowledge
- The *“evil bit”*
- Sufficient time

Democratising Offensive Security

Attacker mindset helps defenders

Generally accepted that some form of offensive mindset helps defenders.
BUT:

- Attack is hard!
- Defenders have a day job
- Offense evolves fast

Scale

Datadog case study

The Datadog environment is **VAST** - exactly how vast is confidential 🙈

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

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

The Solution

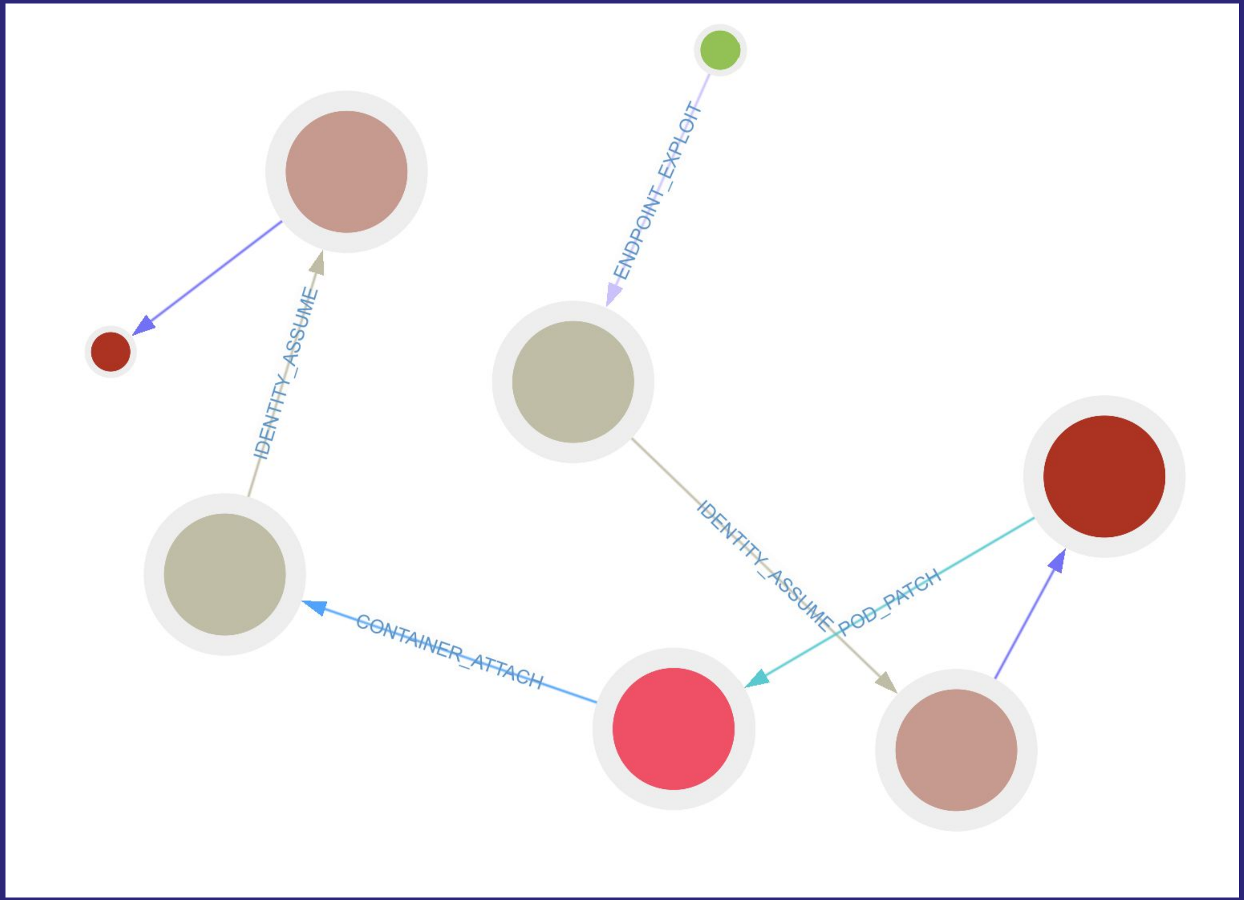
Graph theory X offensive security

Graph Theory

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

Graph Theory

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



Graph Theory 101

Graph

A data type to represent complex, non-linear relationships between objects

- In KubeHound: a Kubernetes cluster

Vertex

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

- In KubeHound: containers, pods, endpoints

Edge

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

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

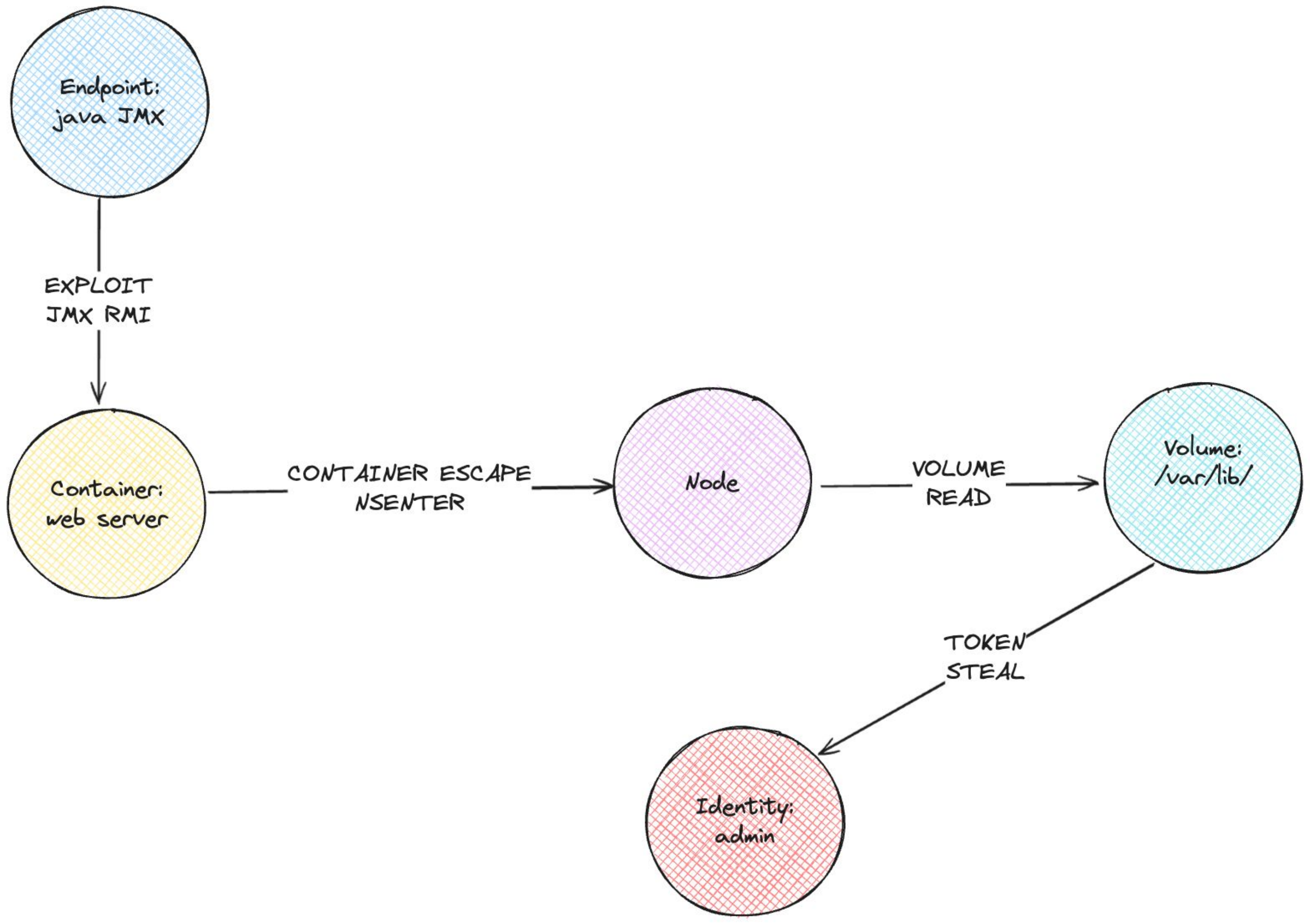
Path

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

Attack Graphs

Attacker TTPs as a data structure



KubeHound 101

Entity

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

Attacks

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

Critical Asset

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

Critical Path

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

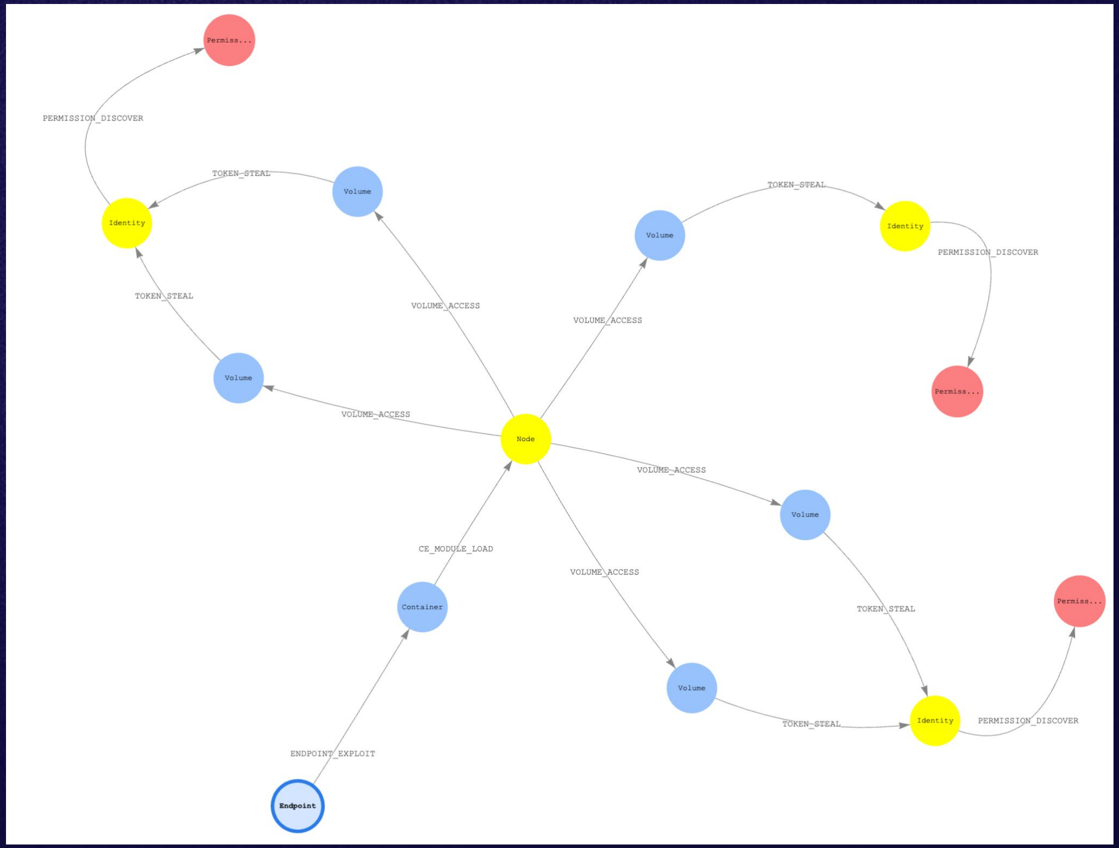
KubeHound

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



KubeHound in Action

Case study and capability showcase

Case Study I

Modelling a complex attack chain

Can we use KubeHound to find a very complex path (10 hops) from an exposed service to a critical asset?

Demo

10 hops to cluster admin

Case Study I

Modelling a complex attack chain

✅ Vulnerability context

✅ Scale

✅ Democratising offense

Case Study II

Quantitative analysis of security posture

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

Demo

Security metrics calculation

Case Study II

Quantitative analysis of security posture

- ✅ Quantifying security posture
- ✅ Scale
- ✅ Democratising offense

Development Process

Research, abstract, design, iterate

Showcase development process

Powerful approach - can be generalised

The approach outlined could be applied to create attack graphs in any number of systems

- Workstations / endpoints
- SaaS identity
- HashiCorp Vault
- AWS

Research

Domain-specific graph model

Research I

Gr0k K8s security

Collate, ingest and categorize all the Kubernetes security research of the past 5 years:

- Conference talks
- Blogs
- Tooling
- Videos

Research II

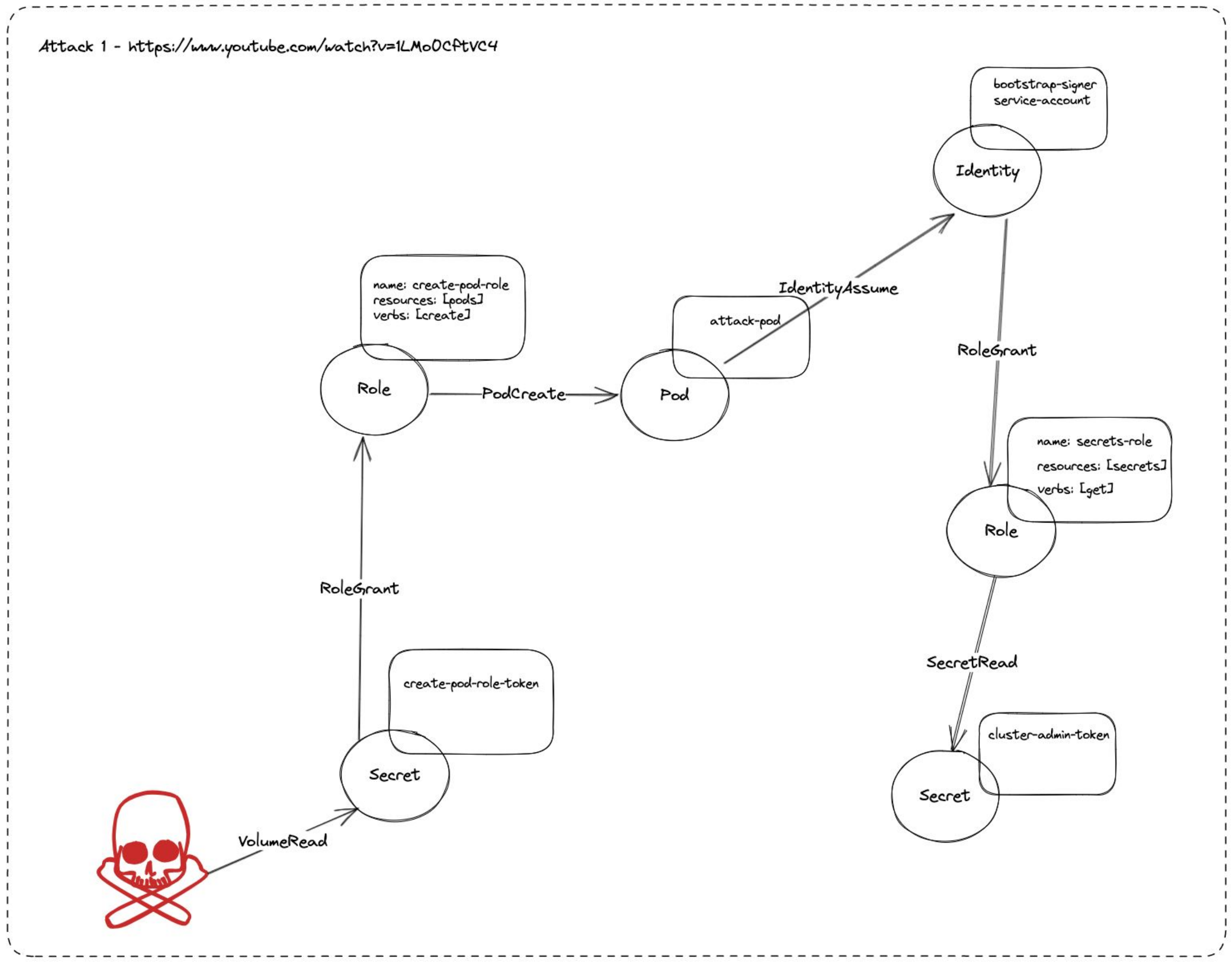
Sketch attack components

Example attack: *"Compromising Kubernetes Cluster by Exploiting RBAC Permissions"* - CyberArk @ RSAC 2020

- Attacker compromises a token with ***pod/create*** permission
- Attacker creates a custom container with an entrypoint script
- Script requests all cluster secrets from K8s API and POSTs to attacker controlled endpoint
- Attack creates a pod with a privileged identity (bootstrap-signer) to run the container image
- *"All your secrets are belong to us"*

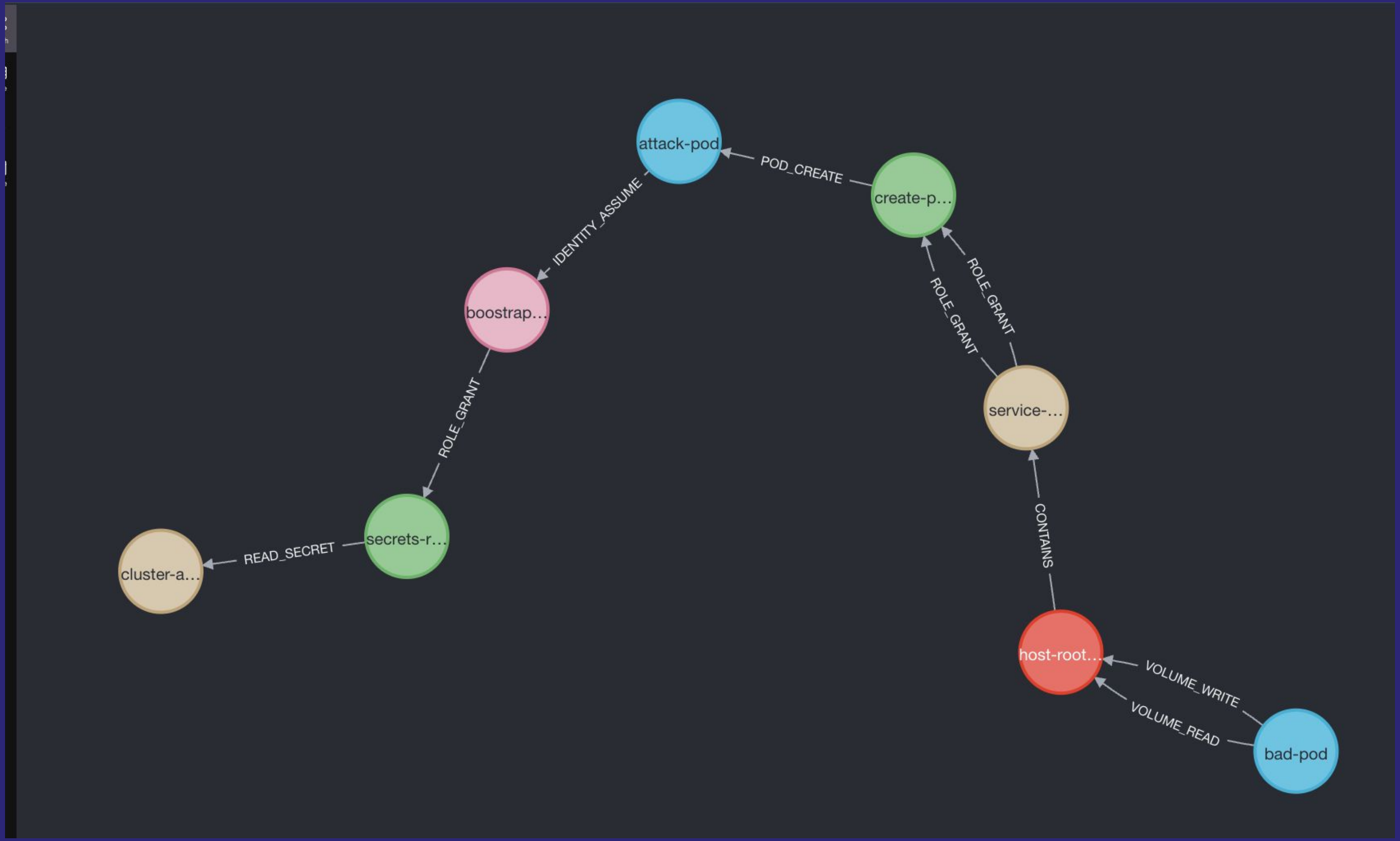
Research II

Sketch attack components



Research III

Port to graph database



Research IV

Repeat for all attacks

CE_MODULE_LOAD
CE_NSENTER
CE_PRIV_MOUNT
CE_SYS_PTRACE
CE UMH CORE PATTERN
CE_VAR_LOG_SYMLINK
CONTAINER_ATTACH
ENDPOINT_EXPLOIT
SHARE_PS_NAMESPACE

EXPLOIT_CONTAINERD
EXPLOIT_HOST_READ
EXPLOIT_HOST_TRAV
EXPLOIT_HOST_WRITE
TOKEN_BRUTEFORCE
TOKEN_LIST
TOKEN_STEAL
VOLUME_ACCESS
VOLUME_DISCOVER

IDENTITY_ASSUME
IDENTITY IMPERSONATE
PERMISSION_DISCOVER
POD_ATTACH
POD_CREATE
POD_EXEC
POD_PATCH
…

Research Phase Outcomes

Confidence in a notional graph model

Mapping known attacks to a notional model gives us confidence that:

- Our model is complete
- We are not processing unnecessary data

Doing this early, and **thoroughly** ensures we have the correct abstraction

Design

Efficient translation of source data to graph

Design I

Formalise graph model

Vertices

- Label (i.e entity type)
- Properties (name, namespace, etc…)

Edges

- Label (i.e attack type)
- Properties (weight)

Calculation methodology

- Source data
- Algorithm

Design II

Data pipeline

Source ingest

- Now we know what data we need - how do we get it?

Vertex creation

- What Kubernetes resources do we need to ingest (*pods, nodes, endpoints*, etc)?

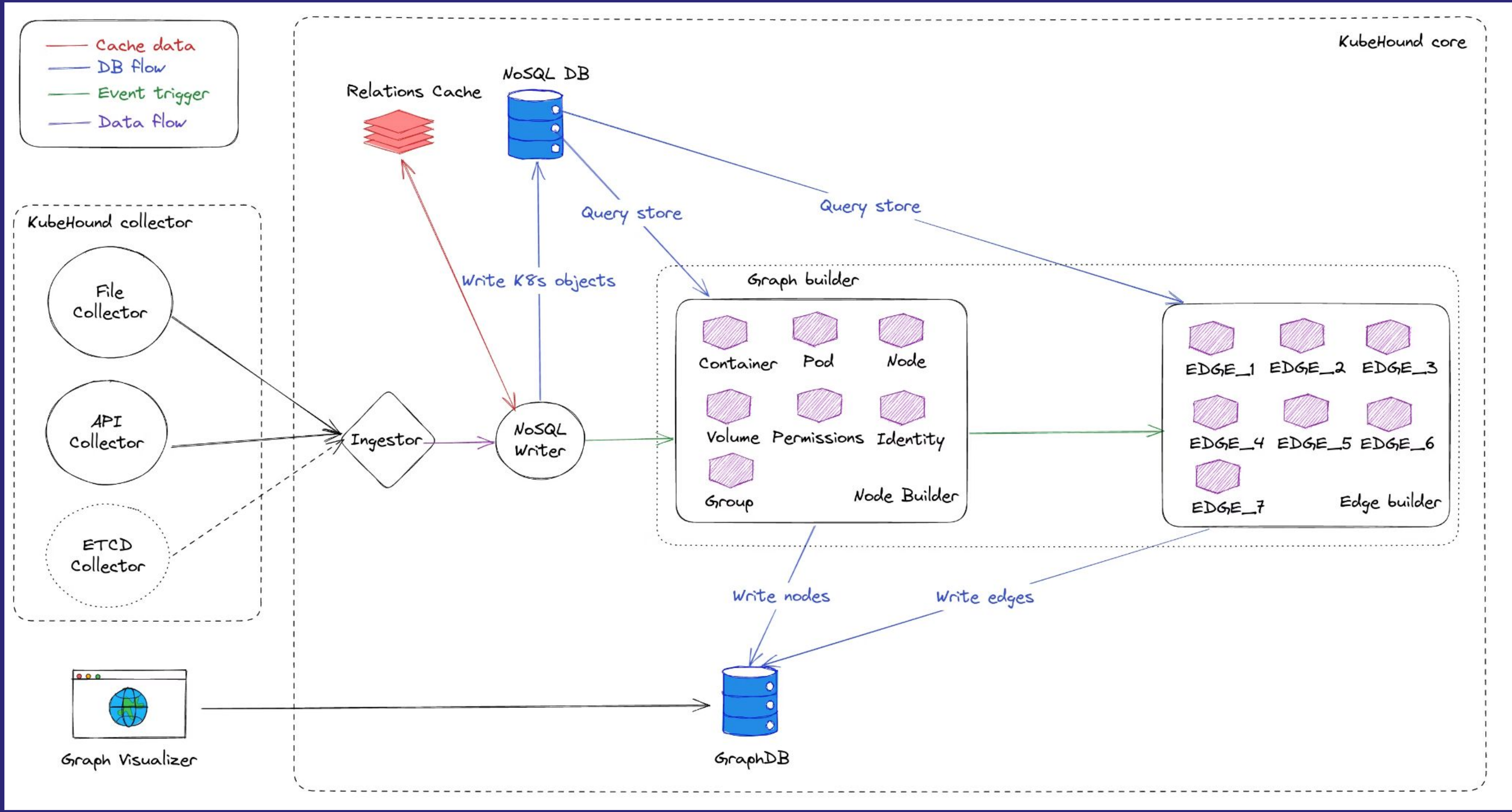
Edge creation

- Do we have the required data to support our edge algorithms?

Dependencies

- What depends on what?
- What can be done in parallel?
- What must be done in sequence?

Final Architecture



Implementation

Guiding principles and outcomes

KubeHound Development

Key principles

- Build for scale
- Focus on performance
- Everything will change
- Build for OSS release
- Optimize for ease of contribution of new attacks

KubeHound Development

Iterative performance improvement

Notional cluster of 25,000 pods:

- v0.1: 10 hours
- v1.0: 1 hour
- v1.1: **5 minutes***

***2 minutes without API rate limiting**

Conclusion

Graphs change the game

Conclusion

Automated attack graphs change the game

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

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

SPECTEROPS

Thank you

Visit kubehound.io to try for yourselves!

Jeremy Fox | @0xff6a