StackRox

*CNCF Webinar Series*

# Kubernetes Security Controls and Enforcement: Applying Lessons from the K8s Security Audit

Connor Gilbert

12 November 2019

CLOUD NATIVE
COMPUTING FOUNDATION

# Who's this?

**Connor Gilbert**
Senior Product Manager, StackRox
Used to be an engineer

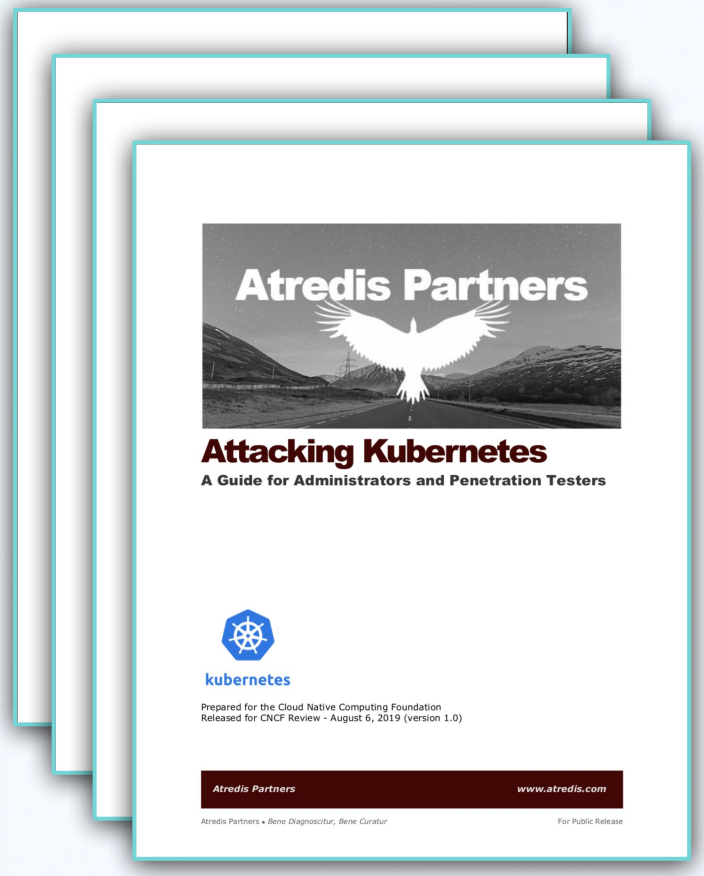
- Stumbled into Kubernetes in 2015
- Introduced it to one company
- Started building container and Kubernetes security products after that

# Coming up

- What happened in the security audit?
  - The process
  - Key takeaways
  - Selected results
- Native K8s controls you can leverage
  - Shared responsibilities
  - What's new? What's not?
  - Default configs vs. ideal configs
- How to design a K8s-native security strategy
  - Enforce security controls and practices…
  - …without sacrificing velocity

# What happened in the security audit?

# Security audit reports




- **Security report:** focuses on K8s internal product security
- **Threat model:** describes key K8s components and how effectively they are secured
- **Whitepaper:** explains important aspects of K8s' design, recommends actions for ops and dev
- **Threat guide:** how to secure or attack a K8s cluster

**Total:** 241 pages

# Security audit logistics

- Security Audit Working Group solicited proposals and selected a winning bid
- Assessment conducted March–May 2019, based on K8s 1.13.4
- Results released in August 2019

# Security audit context

- CNCF has sponsored similar audits of other projects, including:
  - CoreDNS
  - Envoy
  - Prometheus
- Unusually open compared to typical audits of commercial products
- All security findings released publicly (see issue #81146 on GitHub)

# Key takeaways from the security audit

- The Kubernetes project and CNCF are investing in product security
- The security audit identified a number of security issues in specific K8s components
  - Severities range from Informational to High
  - Improvements recommended in various areas
  - GitHub issues are filed for each issue
  - Some issues have been fixed
- Configuring K8s can be complex
  - To be secure, you need to take steps to protect your infrastructure and applications

# Product security findings: overall

Overall, Kubernetes is a **large system** with **significant operational complexity**.

The assessment team found **configuration and deployment** of Kubernetes to be **non-trivial,** with certain components having **confusing default settings, missing operational controls,** and **implicitly defined security controls**.

Also, the state of the Kubernetes **codebase has significant room for improvement**. The codebase is large and **complex,** with large sections of code containing **minimal documentation** and **numerous dependencies,** including systems external to Kubernetes.

*Kubernetes Security Assessment*, Trail of Bits, p. 5

# Product security findings: themes

**Parsing problems**
User input handlers could overflow or run out of memory (TOB-K8S-015, TOB-K8S-019, TOB-K8S-020)

**Insecure deprecated features**
- Insecure SSH tunnels (TOB-K8S-012)
- Passwords in clear text (ATR-K8S-002)

**Dangerous designs**
Features are working as expected, but can be abused, e.g. readiness and liveness probes. (TOB-K8S-024)

**Documentation**
Recommended edits:
- Encryption settings
- PVCs don't enforce PSP host-path limits (TOB-K8S-038)

**Info leaks**
- Verbose logs (TOB-K8S-001)
- CoreDNS zone transfer (TOB-K8S-032)
- Sensitive host env vars (TOB-K8S-005)

**Feature requests**
Areas for improvement:
- Seccomp (TOB-K8S-002)
- TLS cert revocation (TOB-K8S-028)

*For current status, see issue #81146 on GitHub.*

# Native K8s controls you can leverage

# Background: shared responsibility

Application Development

Application Operations

Infrastructure

Managed Service

Risk

Security

Compliance

# Background: security surface

- Application security
- Deployment configs
- Microservice interactions
- Kubernetes API access
- Node configs
- Security configs
- Monitoring

# What's different? What's the same?

**Threats**

- **Apps:** On the Internet, nobody knows you're ~~a dog~~ running in a container
- **Infrastructure:** You have a new, powerful API surface to protect

**Security workflows**

- **Apps:**
  - Deployed using immutable SHA256 identifiers
  - Configured with declarative specifications
  - Not messed with manually—much less "I'll just SSH in"
  - Built to be failure-tolerant
- **Infrastructure:**
  - Central place to configure many important security configurations
  - Software-defined everything™
  - A critical foundation, but now more strictly separated from apps

# API and infrastructure security

> While Kubernetes facilitates high-availability workload deployments, the underlying hosts, components, and environment of a Kubernetes cluster must be **configured and managed**. This management has a **direct impact on the capabilities** of the cluster, and **affects the behavior** of an operator's composed objects.
> *Kubernetes Security Whitepaper,* Trail of Bits, p. 5

- Network access to API server
  - cf. Billion Laughs (CVE-2019-11253)
- RBAC *(covered later)*
- Access control for nodes
- Standard Linux node hardening
  - Advantage: limited-purposes, cookie-cutter nodes

# Pods

Often, compromising a Kubernetes cluster begins with **first compromising a lower privileged Pod**. The secure configuration of Pods is an **often-overlooked** aspect of the system.

*Attacking Kubernetes,* Atredis Partners, p. 46

**Job #1**
Keep adversaries out.

**Job #2**
Keep them in one place.

# Defaults aren't enough

Kubernetes contains **many default settings** which **negatively impact the security posture** of a cluster.

These settings also have **conflicting usage semantics,** where some use either **opt-in or opt-out** specifications. The conflicting usage generally boils down to the preservation of **backwards compatibility** for both workload and component configurations.

Ensuring appropriate configuration of all options **requires significant attention** by cluster administrators and operators.

*Kubernetes Security Whitepaper*, Trail of Bits, p. 17

# Image security

File systems also often contain **bash** or **package managers** that further **enable an attacker** to gain a shell and install additional tools. An ideal installation should **remove all non-essential binaries** and **prevent modification** to the binaries that are required.

For cluster administrators, care should be taken that **vulnerable applications and Pods** are **patched as soon as possible,** so that Internal Attackers may not gain an **initial foothold** within the cluster.

*Attacking Kubernetes,* Atredis Partners, p. 46

# Deploy-phase controls: RBAC consequences

All containers in a Pod run with a **service account**. … **Attack scenarios** have been documented against third-party services which will orchestrate Pod deployment using overly permissive service accounts. In these instances, a compromise of a Pod container is **catastrophic**.

…however, they **may not yield much access when using role-based access control** (RBAC) authorization controls.

At the time of this report, Kubernetes mounted **default credentials in every Pod**; an Internal Attacker could use these credentials to access other resources within the cluster, such as the kublet *(sic)*. From there, the Internal Attacker may be able to **move laterally** throughout the cluster to wider access.

*Attacking Kubernetes*, Atredis Partners, pp. 14-15, 23

# Deploy-phase controls: RBAC difficulties

> [Objects] can be composed by **referencing objects** that may not yet exist. Additionally, objects can be created even if the component using the object does not exist. This functionality can be **very dangerous when constructing RBAC policies**, since functionality must be tested to ensure the configuration works in the expected manner. This could lead an administrator to **believe that policies are in effect, when in fact they are not**.

*Kubernetes Security Whitepaper,* Trail of Bits, p. 16

# Deploy-phase controls: RBAC background

## Role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: use-psp
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - policy1
```

+

## Role Binding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: <binding name>
roleRef:
  kind: ClusterRole
  name: use-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: <sa>
  name: <sa namespace>
```

**= Access**

# Deploy-phase controls: RBAC best practices

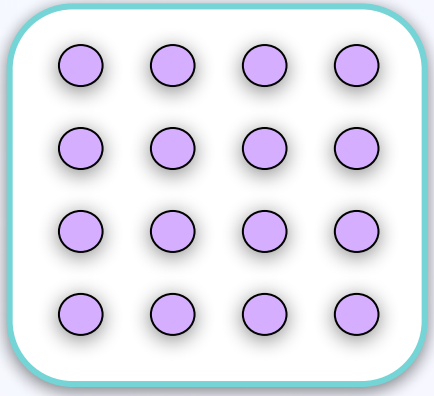
- *cluster-admin* considered harmful
- Use role aggregation carefully
- Grant each role with exactly one binding
- Clean up unused roles and bindings
- Avoid "dangling bindings" to deleted roles

⚠️ *None of this matters* if the legacy Attribute-Based Access Control (ABAC) controller is still enabled.

# Deploy-phase controls: Namespaces

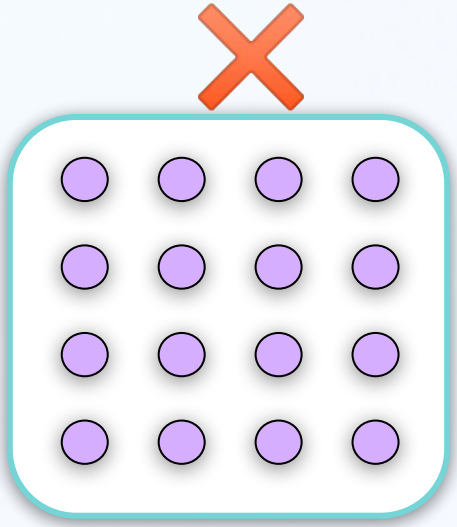
**Kubernetes namespaces** were developed as a method to **help provide workload isolation**. Running multiple, potentially multi-tenant, workloads in the same namespace **sidesteps the protections of namespaces**, resulting in a single large and flat namespace.

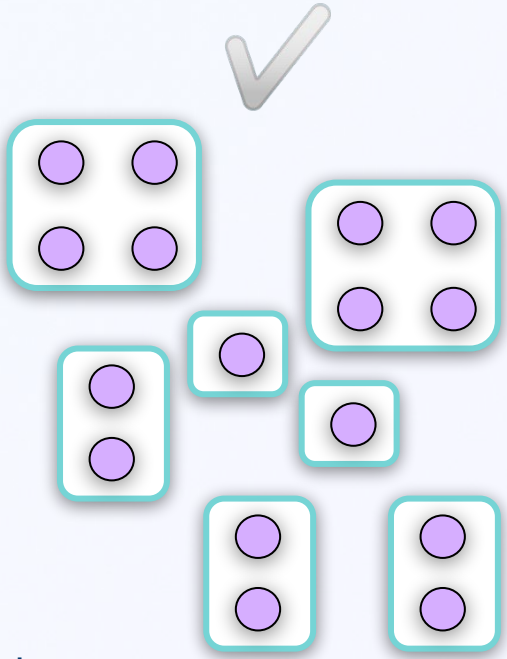
*Kubernetes Security Whitepaper,* Trail of Bits, p. 9



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    tier: backend
  namespace: my-team-ns
spec:
...
```

# Deploy-phase controls: Namespaces





- Network policies
- RBAC
- Ability to mount secrets
- Use of image pull secrets
- *And other boundaries*

# Deploy-phase controls: Read-only root file system

> [The] root file system is not commonly **read-only**, allowing for **additional tools to be installed**.
> *Attacking Kubernetes*, Atredis Partners, p. 46

*Need a writable path?*

- Add a `VOLUME` Dockerfile instruction
- Mount a K8s `emptyDir` volume (CRI-O doesn't make `VOLUME`s writable)

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    ...
    spec:
      containers:
      - name: my-container
        ...
        securityContext:
          capabilities:
            drop: ["NET_RAW"]
          readOnlyRootFilesystem: true
```

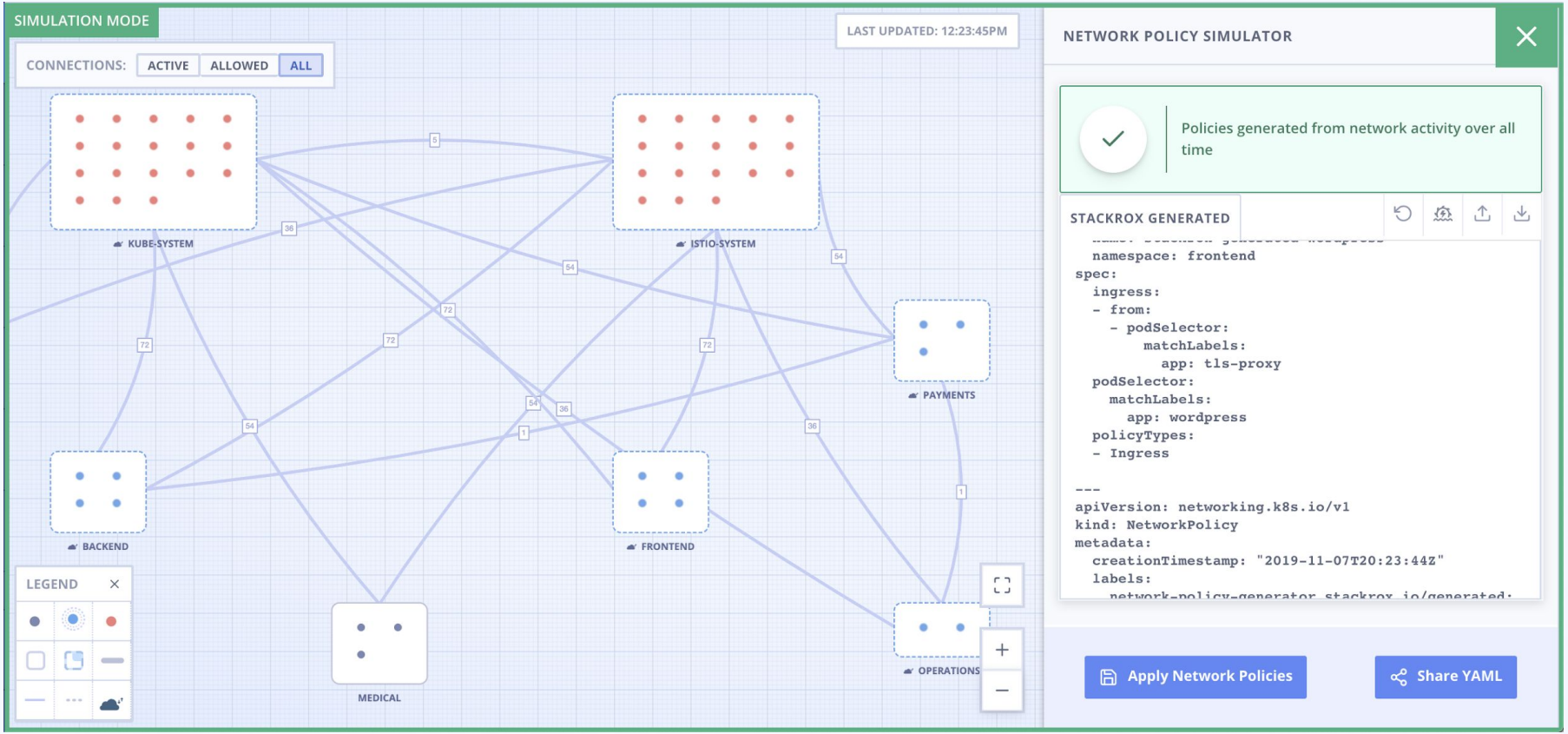
# Deploy-phase controls: Network policies

Finally, ensure the container network interface is **as restrictive as possible** through the definition of **cluster network policies**.
*Kubernetes Security Whitepaper,* Trail of Bits, p. 17

By default, every pod can talk to every other pod.

To change this, apply a network policy. Network restrictions are enforced for a given pod only when that pod has a policy applied to it.

# Deploy-phase controls: Network policies



# How to design a K8s-native security strategy

# The opportunity we have

Despite the results of the assessment and the operational complexity of the underlying cluster components, **Kubernetes streamlines difficult tasks** related to maintaining and operating cluster workloads such as deployments, replication, and storage management.

Additionally, **Kubernetes takes steps to help cluster administrators harden and secure their clusters** through features such as Role Based Access Controls (RBAC) and various policies which extend the RBAC controls.

Continued development of these security features, and further refinement of best practices and sane defaults will **lead the Kubernetes project towards a secure-by-default configuration**.

*Kubernetes Security Assessment,* Trail of Bits, p. 6

# The opportunity we have

# Enforcement options

**Pod Security Policy**
- Native up-front config
- Uses RBAC identities
- Controls privileges, host mounts, and other settings

**Dynamic admission controller**
- Requires deployment
- Can consider any data—the sky (and API timeout) is the limit!

**Ongoing monitoring and analysis**
- Gives the benefit of hindsight
- Doesn't block progress
- But, it's reactive

⚠️ Remember the user experience when choosing:
- what to enforce, and
- where to enforce it.

Don't be a bottleneck and don't cry "wolf".

# How to get started

- Start with the easy stuff that helps everyone
  - Annotate and label deployments consistently
  - Use concrete image tags (not `latest`)
  - Start scanning images for low-hanging fruit
- Continue with other self-contained changes
  - Limit network access to the K8s API server
  - Start disabling automatic service account mount
  - Replace your cluster admins with scoped access
- Work on cross-functional changes app-by-app
  - Try a read-only root file system for stateless services
  - Make sure resource requirements are specified
  - Add ingress network policies to sensitive deployments (then all)
- Keep going!

# Conclusion

# Recap

- The Kubernetes security audit identified a number of improvements for Kubernetes itself.
- You can also apply a number of specific controls to your clusters to improve your security posture.
- You can use Kubernetes to collaboratively improve security without impeding development velocity.
  - Use declarative, immutable configs to your advantage!
  - Pick from the enforcement menu as you up your security game.

# What's next?

**Have a question now?**
Ask in Zoom!

**Think of one later?**
✉c@stackrox.com
@connorgilbert

**Want to learn more?**
https://stackrox.com/cncf/