sysdig SECURE EVERY SECOND.

Why Kubernetes is Insecure by Default **And What You Can Do About It**

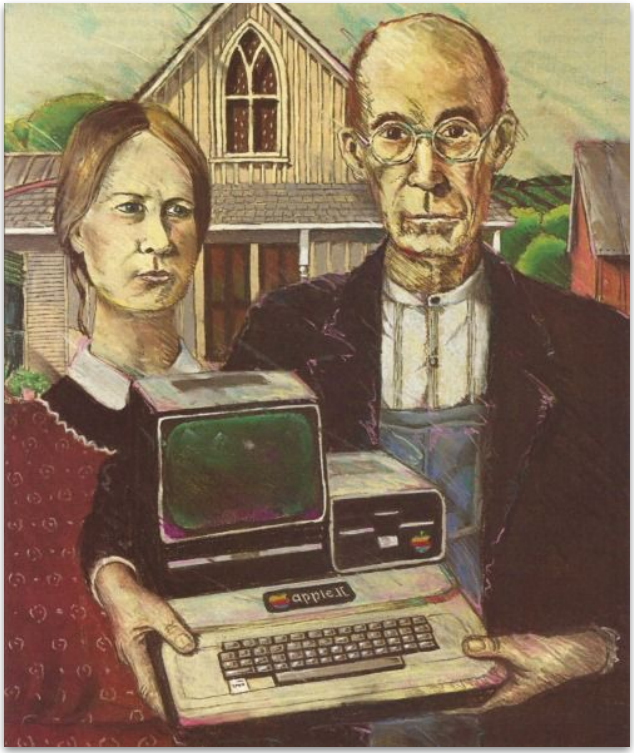
Curtis Collicutt

Solutions Engineer @Sysdig

About Me

- Work at Sysdig helping customers determine security outcomes & achieve them
- I try to focus on the socio-technical aspects of computing

Email curtis.collicutt@sysdig.com for comments/criticisms/critiques



"We build our computers the way we build our cities—over time, without a plan, on top of ruins"

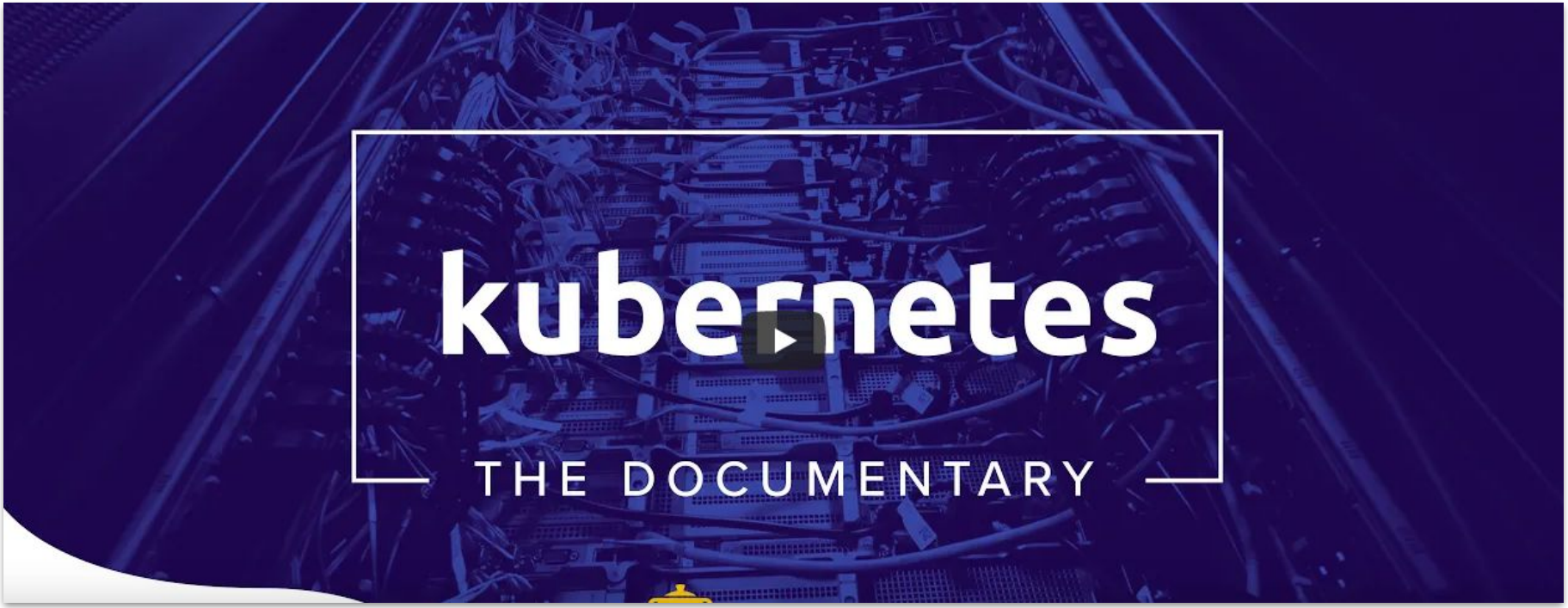
Ellen Ullman
Computer Programmer
& Author

“Everything fails, all the time”

Werner Vogels
AWS CTO

Why Is Kubernetes Insecure by Default?

The History of Kubernetes



https://www.youtube.com/watch?v=BE77h7dmoQU

The Borg (Paper)

Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes

Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

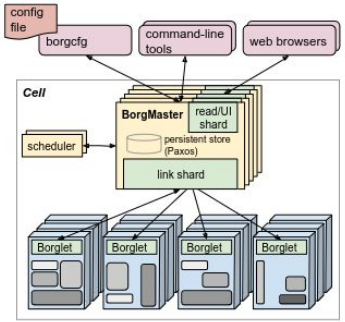
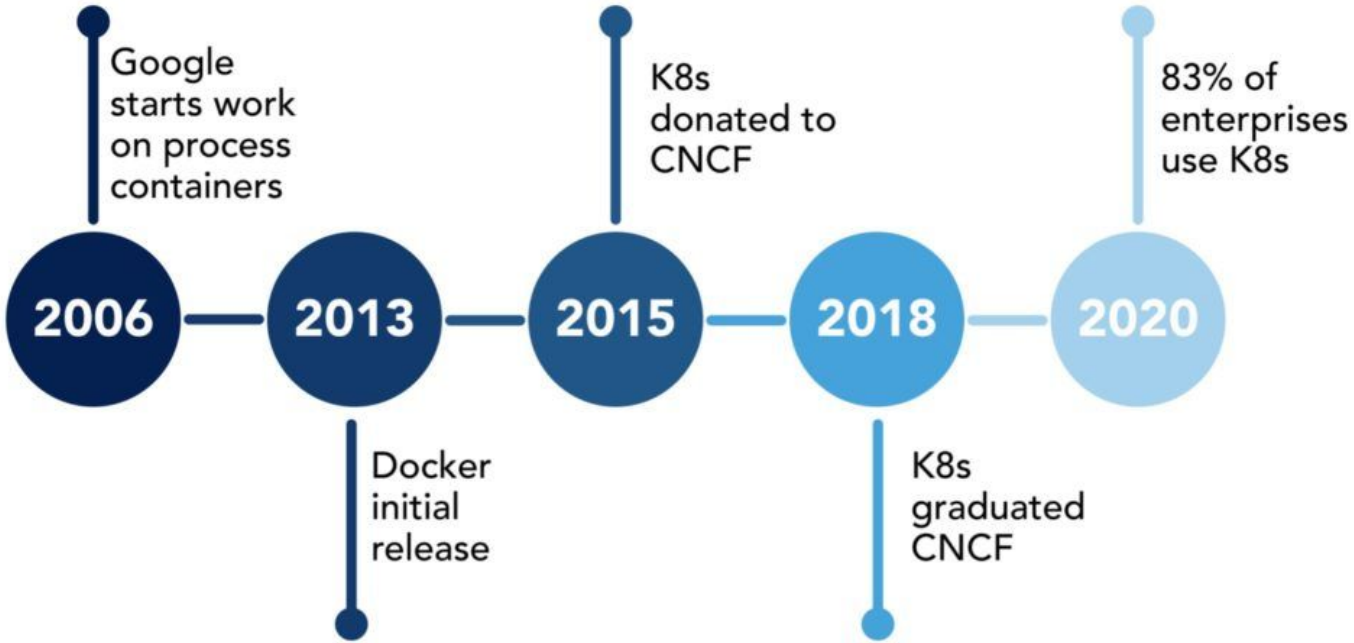


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

cluding with a set of qualitative observations we have made from operating Borg in production for more than a decade.

1 Introduction



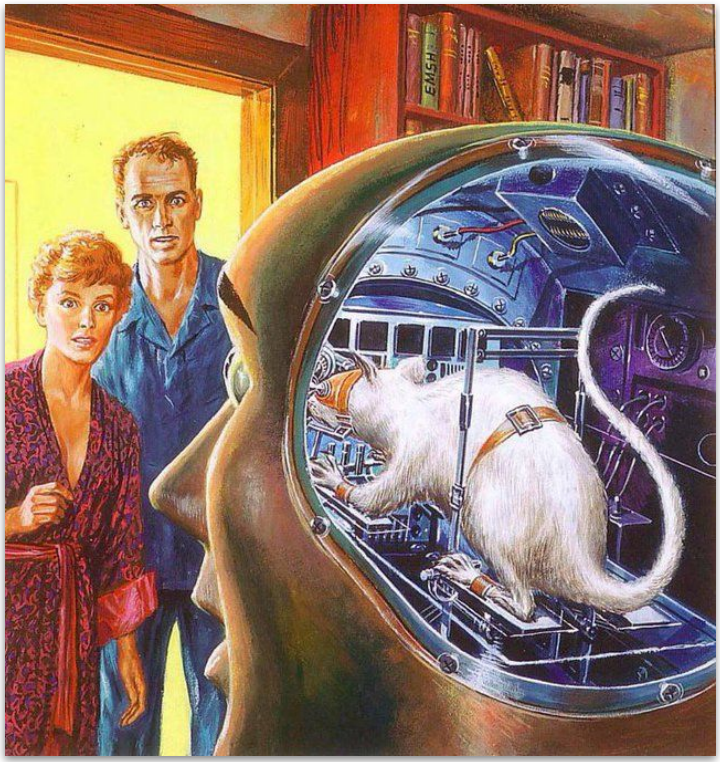
https://www.opsramp.com/guides/why-kubernetes/who-made-kubernetes/

Design Decisions

- I want to make it clear that the open source teams working on Kubernetes have done an incalculable amount of work that I'm thankful for.
- During that work, they made design decisions; decisions that allowed Kubernetes to be extremely successful.
- Design decisions are not good or bad, just a choice, often based around differentiation of the project/product.

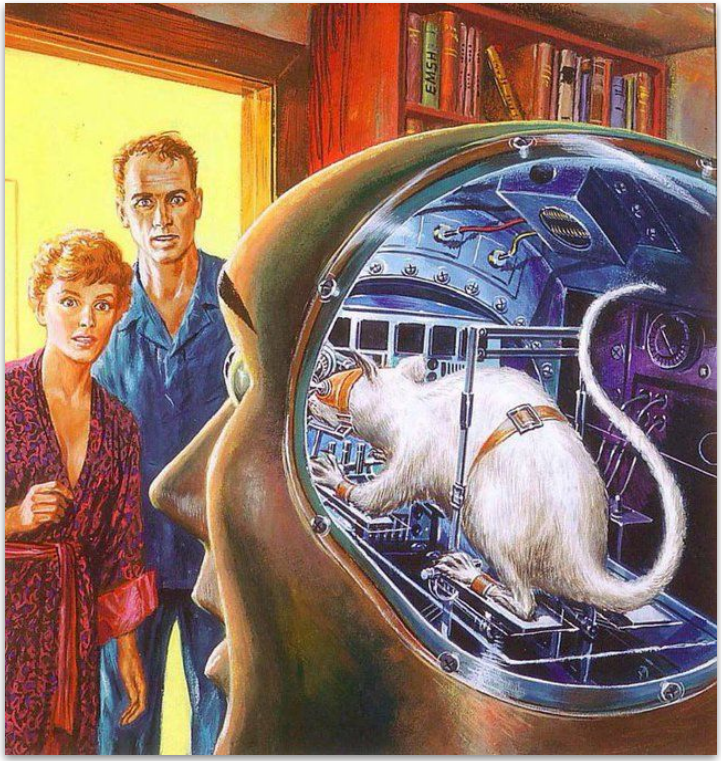
Run as Root

```
$ k run nginx --image=nginx
$ k exec -it nginx -- cat \ /proc/1/status | grep "Uid\|Gid"
Uid:    0    0    0    0
Gid:    0    0    0    0
```



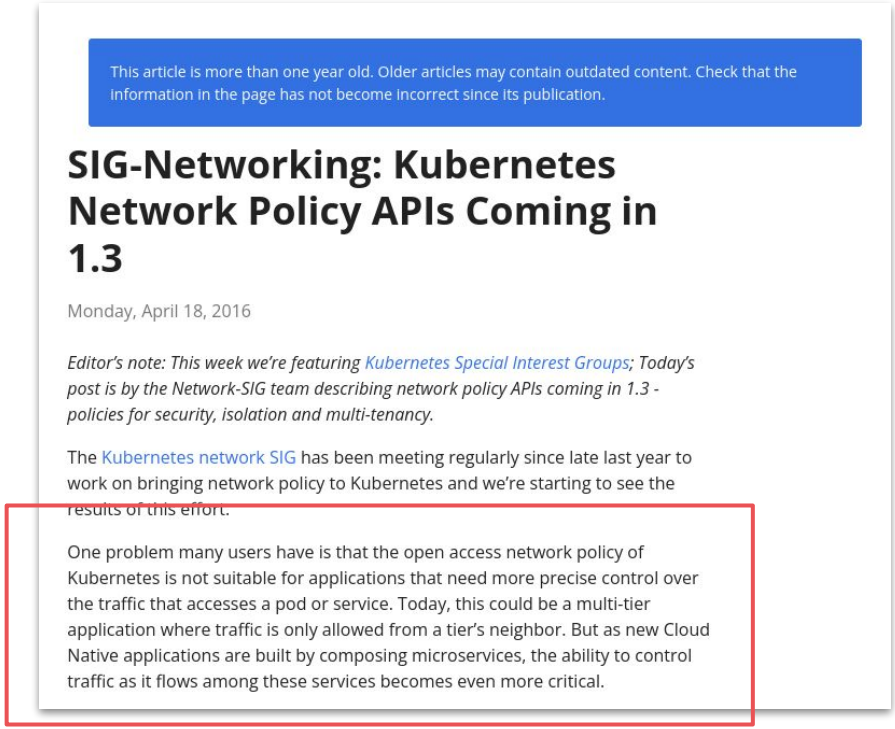
Run as Root...Why?

- Seems to come from Docker's original stance...?
- Ease of use?
- Legacy applications might have needed it at the time?
- Linux heritage?
- Containers not considered a security boundary?
- Unaware of Implications?
- Overall flexibility?



The Network is Wide Open (Design Decision)

- Kubernetes originated without Network Policies, and **the point** was for every pod to be able to talk to every other pod on a layer 3 network
- This kind of layer 3 level access provided great scalability (no network broadcasting)
- Network Policies added in **1.3**



This article is more than one year old. Older articles may contain outdated content. Check that the information in the page has not become incorrect since its publication.

SIG-Networking: Kubernetes Network Policy APIs Coming in 1.3

Monday, April 18, 2016

Editor's note: This week we're featuring Kubernetes Special Interest Groups; Today's post is by the Network-SIG team describing network policy APIs coming in 1.3 - policies for security, isolation and multi-tenancy.

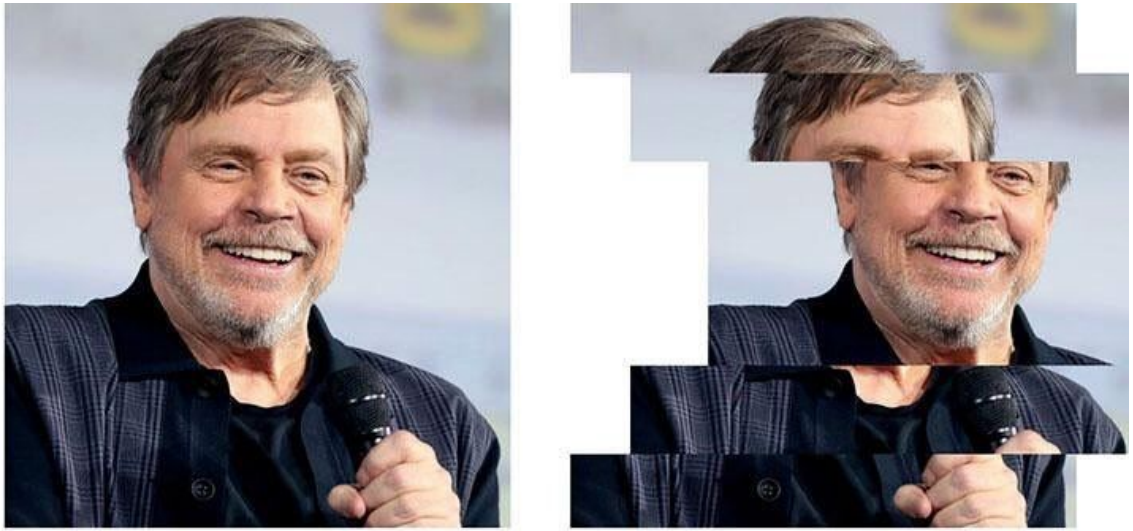
The Kubernetes network SIG has been meeting regularly since late last year to work on bringing network policy to Kubernetes and we're starting to see the results of this effort.

One problem many users have is that the open access network policy of Kubernetes is not suitable for applications that need more precise control over the traffic that accesses a pod or service. Today, this could be a multi-tier application where traffic is only allowed from a tier's neighbor. But as new Cloud Native applications are built by composing microservices, the ability to control traffic as it flows among these services becomes even more critical.

https://kubernetes.io/blog/2016/04/kubernetes-network-policy-apis/?m=0

The Security Onus is Put on the User

If you want a more secure container deployment, you add the “the YAML”



Things We Can Do to Make **Kubernetes Secure by Default**

What Would Make Sense in Almost Any Kubernetes Environment?

1. Secure Baseline
    a. Don’t allow root containers, etc
    b. Don’t allow pods to talk to pods in other namespaces (lateral movement)
2. Automate the baseline
3. Build a last line of defense with Falco

Pod Security Standards (PSS)

```
kubectl label namespace <NAMESPACE> pod-security.kubernetes.io/enforce=restricted
```

- Limit types of volumes
- Don't run as root user
- No privilege escalation
- Seccomp profile set to `"RuntimeDefault"` or `"Localhost"`
- Drop all capabilities except perhaps add `NET_BIND_SERVICE`

PSPs are replaced with Pod Security Admission (PSA), a built-in admission controller that implements the security controls outlined in the Pod Security Standards (PSS)

Kubernetes v1.25: Pod Security Admission Controller in Stable

Thursday, August 25, 2022

Authors: Tim Allclair (Google), Sam Stoelinga (Google)

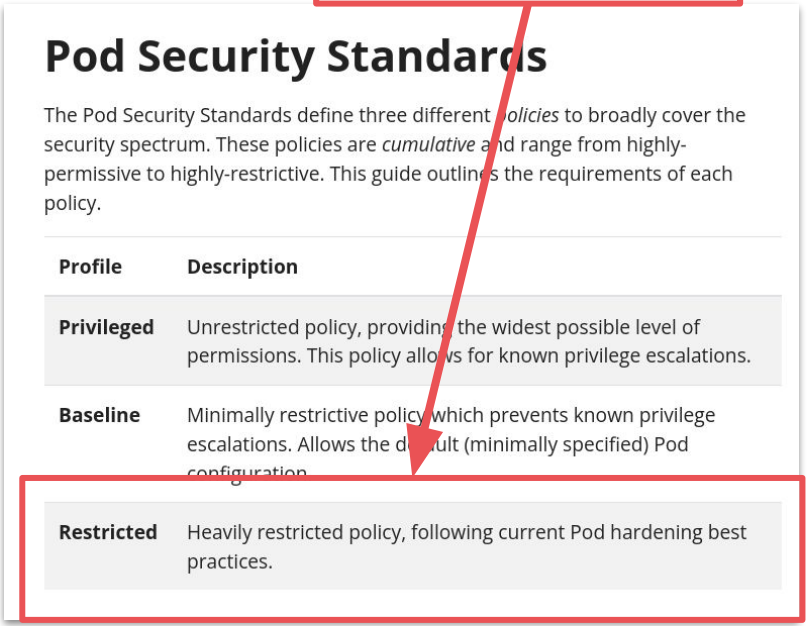
The release of Kubernetes v1.25 marks a major milestone for Kubernetes out-of-the-box pod security controls: Pod Security admission (PSA) graduated to stable, and Pod Security Policy (PSP) has been removed. PSP was deprecated in Kubernetes v1.21, and no longer functions in Kubernetes v1.25 and later.

https://kubernetes.io/docs/concepts/security/pod-security-standards/

Pod Security Standards (PSS)

```
kubectl label namespace <NAMESPACE> pod-security.kubernetes.io/enforce=restricted
```

- Limit types of volumes
- Don't run as root user
- No privilege escalation
- Seccomp profile set to "RuntimeDefault" or "Localhost"
- Drop all capabilities except perhaps add NET_BIND_SERVICE



Pod Security Standards

The Pod Security Standards define three different *policies* to broadly cover the security spectrum. These policies are *cumulative* and range from highly-permissive to highly-restrictive. This guide outlines the requirements of each policy.

Profile	Description
Privileged	Unrestricted policy, providing the widest possible level of permissions. This policy allows for known privilege escalations.
Baseline	Minimally restrictive policy which prevents known privilege escalations. Allows the default (minimally specified) Pod configuration.
Restricted	Heavily restricted policy, following current Pod hardening best practices.

https://kubernetes.io/docs/concepts/security/pod-security-standards/

Network Policy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: namespace-only
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector: {}
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
```

- **Don't** allow pods to talk to pods in other namespaces
- **Do** allow pods to talk to one another in the same namespace
- **Do** allow connectivity to Kubernetes service objects
- **Do** allow egress to 0.0.0.0/0
- Meant to work most anywhere (some won't like default egress though)

https://github.com/ccollicutt/kubernetes-is-insecure-by-default

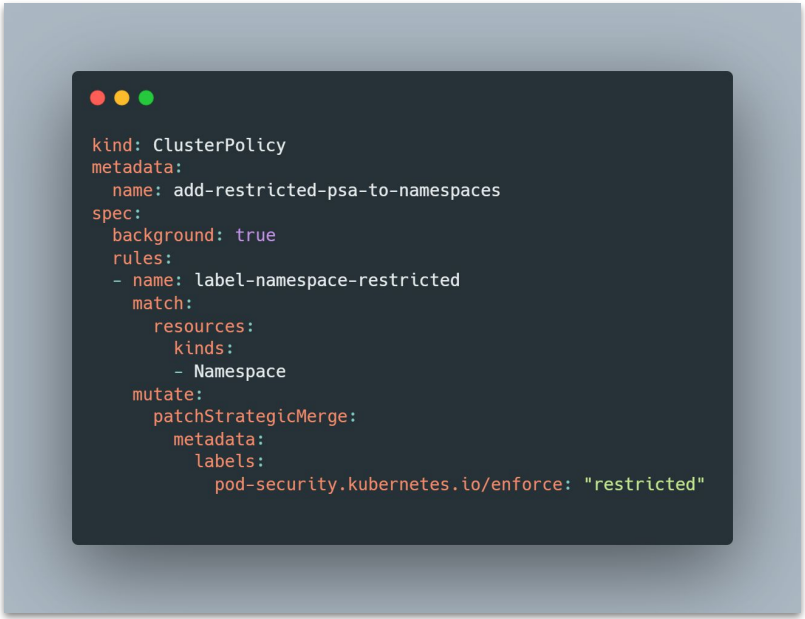
Automate Those New Defaults

- One way is with Kyverno ClusterPolicy
- Basically apply a template to new namespaces, new pods, etc
- E.g. here we merge in:
  - Disallows privilege escalation (`allowPrivilegeEscalation: false`)
  - Requires the container to run as a non-root user (runAsNonRoot: true)
  - Drops all additional Linux capabilities (`capabilities: drop: - ALL`)
  - It sets a cluster-wide securityContext that mandates the use of a `RuntimeDefault Seccomp` profile for the whole Pod.

https://github.com/ccollicutt/kubernetes-is-insecure-by-default

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: security-context
spec:
  background: false
  validationFailureAction: Enforce
  rules:
  - name: securitycontext-allowPrivilegeEscalation
    match:
      resources:
        kinds:
        - Pod
    mutate:
      foreach:
      - list: "request.object.spec.containers"
        patchStrategicMerge:
          spec:
            containers:
            - name: "{{ element.name }}"
              securityContext:
                allowPrivilegeEscalation: false
                runAsNonRoot: true
                capabilities:
                  drop:
                  - ALL
  - name: securitycontext-seccompProfile
    match:
      resources:
        kinds:
        - Pod
    mutate:
      patchStrategicMerge:
        spec:
          securityContext:
            seccompProfile:
              type: RuntimeDefault
```

Add Enforce Restricted to All Namespaces



```yaml
kind: ClusterPolicy
metadata:
  name: add-restricted-psa-to-namespaces
spec:
  background: true
  rules:
  - name: label-namespace-restricted
    match:
      resources:
        kinds:
        - Namespace
    mutate:
      patchStrategicMerge:
        metadata:
          labels:
            pod-security.kubernetes.io/enforce: "restricted"
```

- Any new namespace will have the enforce: restricted label applied automatically
- *NOTE: Would want to restrict some namespaces from having this applied*

https://github.com/ccollicutt/kubernetes-is-insecure-by-default

Automate Those New Defaults

- This Kyverno policy includes a generate rule that will create a `NetworkPolicy` resource with specified settings in every new namespace
- `kind: NetworkPolicy` specifies the type of resource to generate.
- `name: namespace-only` is the name of the NetworkPolicy.
- namespace: `"{{request.object.metadata.name}}"` sets the namespace to the name of the newly created Namespace.
- Data contains the specification for the NetworkPolicy

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-custom-network-policy
spec:
  rules:
  - name: generate-network-policy-for-namespace
    match:
      resources:
        kinds:
        - Namespace
    generate:
      kind: NetworkPolicy
      name: namespace-only
      namespace: "{{request.object.metadata.name}}"
      apiVersion: networking.k8s.io/v1
      data:
        spec:
          podSelector: {}
          policyTypes:
          - Ingress
          - Egress
          ingress:
          - from:
            - podSelector: {}
          egress:
          - to:
            - ipBlock:
                cidr: 0.0.0.0/0
```

https://github.com/ccollicutt/kubernetes-is-insecure-by-default

A Note on runAsNonRoot

```
$ k describe pod plain-nginx  | grep Error
    Reason:     CreateContainerConfigError
  Warning  Failed   98s (x8 over 2m50s)  kubelet            Error: container has runAsNonRoot and image will run as root (pod: "plain-nginx_enforcing(c9c06da8-3ce7-404c-89c5-0feb218b8b02)", container: plain-nginx)
$ k get pod plain-nginx -oyaml | grep "image:"
  - image: nginx
```


https://github.com/ccollicutt/kubernetes-is-insecure-by-default

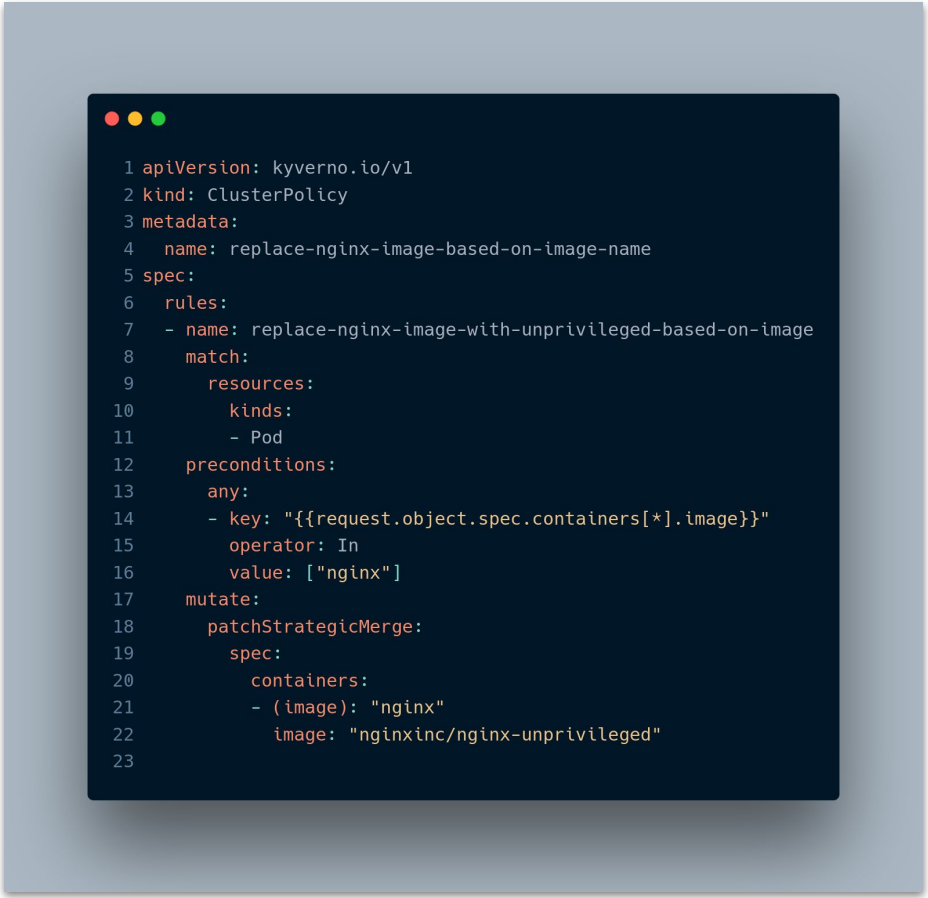
Automate Those New Defaults

Replace any image using plain nginx

with

nginxinc/nginx-unprivileged

*NOTE: You wouldn't do this in the real world, the point is that you **can** mutate/alter the resource into almost anything you want.*



```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: replace-nginx-image-based-on-image-name
spec:
  rules:
  - name: replace-nginx-image-with-unprivileged-based-on-image
    match:
      resources:
        kinds:
        - Pod
    preconditions:
      any:
      - key: "{{request.object.spec.containers[*].image}}"
        operator: In
        value: ["nginx"]
    mutate:
      patchStrategicMerge:
        spec:
          containers:
          - (image): "nginx"
            image: "nginxinc/nginx-unprivileged"

```

https://github.com/ccollicutt/kubernetes-is-insecure-by-default

Secure Defaults

Demo

Or With a Mutating Webhook

- I wrote a Mutating webhook in Python that does the same thing
- Just a demo/toy

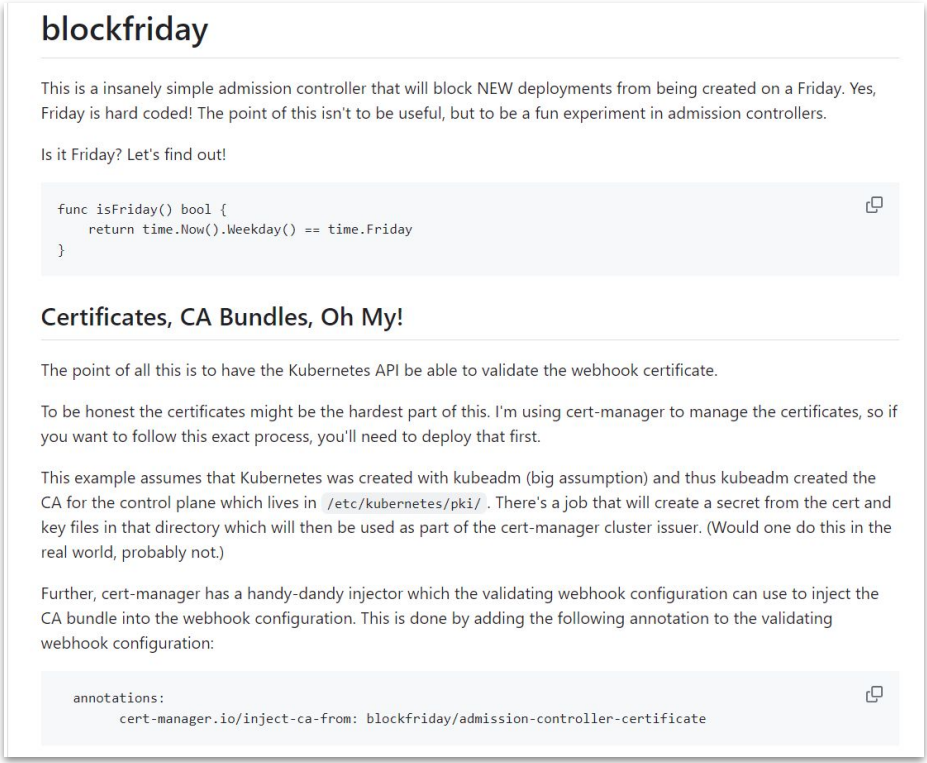
Code Blame 74 lines (61 loc) · 2.22 KB

```python
from flask import Flask, request, jsonify
import json
import base64
import logging
import os

app = Flask(__name__)

SWAP_IMAGE = "nginxinc/nginx-unprivileged:latest"


def create_patches(containers):
    patches = []
    for i, container in enumerate(containers):
        patches.append(
            {
                "op": "replace",
                "path": f"/spec/containers/{i}/image",
                "value": SWAP_IMAGE,
            }
        )
    return patches

```

https://github.com/ccollicutt/swap-image-mutating-webhook

Or You Can Write an Admission Controller

- Here's one I wrote that blocks deployments on Friday
- Written in Go
- Just a demo/toy



blockfriday

This is a insanely simple admission controller that will block NEW deployments from being created on a Friday. Yes, Friday is hard coded! The point of this isn't to be useful, but to be a fun experiment in admission controllers.

Is it Friday? Let's find out!

```
func isFriday() bool {
    return time.Now().Weekday() == time.Friday
}
```

Certificates, CA Bundles, Oh My!

The point of all this is to have the Kubernetes API be able to validate the webhook certificate.

To be honest the certificates might be the hardest part of this. I'm using cert-manager to manage the certificates, so if you want to follow this exact process, you'll need to deploy that first.

This example assumes that Kubernetes was created with kubeadm (big assumption) and thus kubeadm created the CA for the control plane which lives in `/etc/kubernetes/pki/`. There's a job that will create a secret from the cert and key files in that directory which will then be used as part of the cert-manager cluster issuer. (Would one do this in the real world, probably not.)

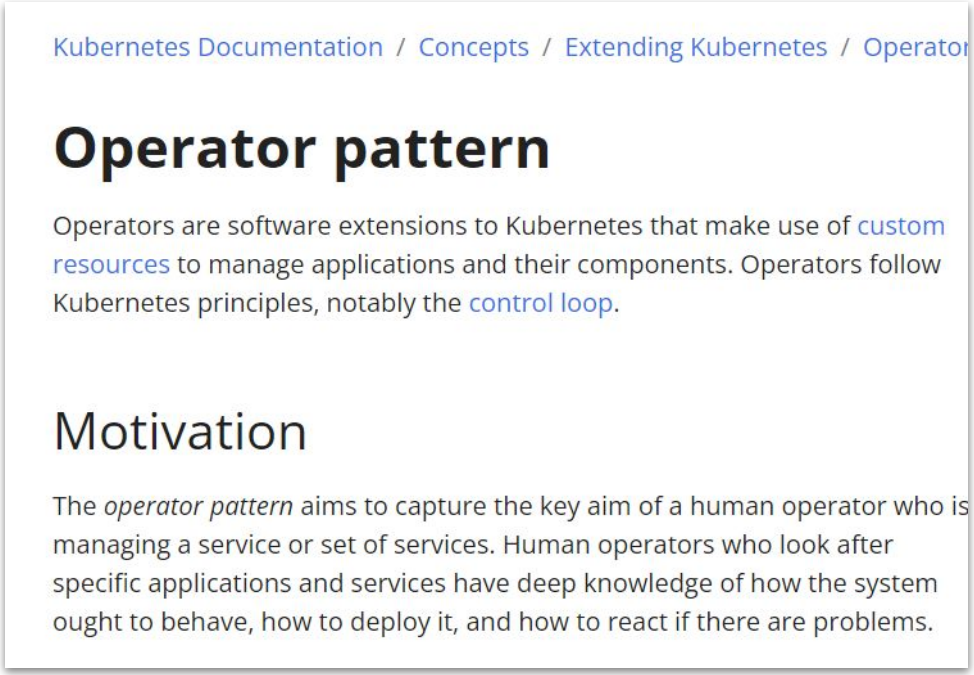
Further, cert-manager has a handy-dandy injector which the validating webhook configuration can use to inject the CA bundle into the webhook configuration. This is done by adding the following annotation to the validating webhook configuration:

```
  annotations:
       cert-manager.io/inject-ca-from: blockfriday/admission-controller-certificate
```

https://github.com/ccollicutt/swap-image-mutating-webhook

Or You Can Write an Operator

- Operators allow for considerably more flexibility and capability, but are more complex to create and maintain



https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

Or You Can Use ValidatingAdmissonCpolicy

- Allow you to use the CEL expression language

Policy

validatingadmissionpolicy/basic-e

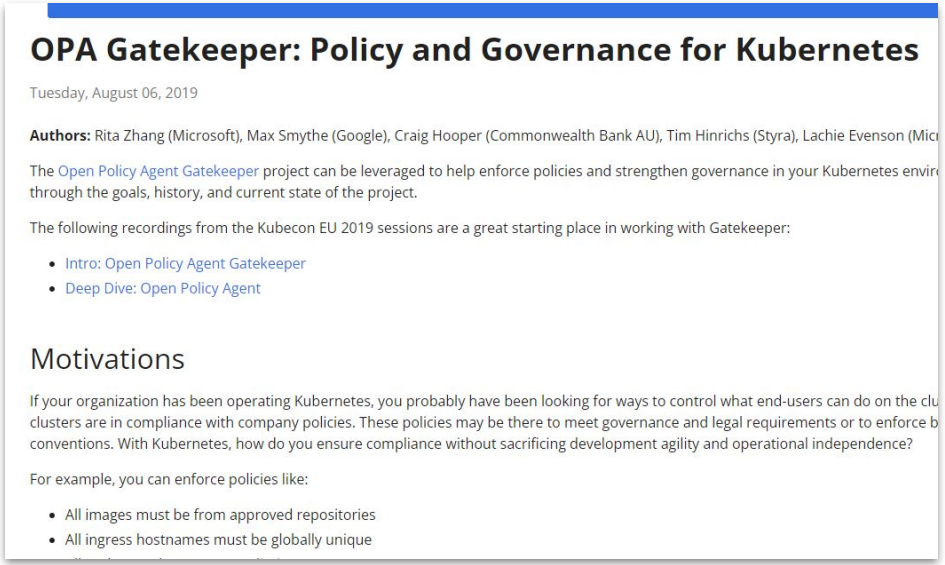
```yaml
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingAdmissionPolicy
metadata:
  name: "demo-policy.example.com"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
    - apiGroups:   ["apps"]
      apiVersions: ["v1"]
      operations:  ["CREATE", "UPDATE"]
      resources:   ["deployments"]
  validations:
    - expression: "object.spec.replicas <= 5"
```

https://kubernetes.io/docs/reference/access-authn-authz/validating-admission-policy/

Or OPA Gatekeeper

- A generic policy engine
- Sysdig uses OPA as part of its Cloud Security Posture Management offering

Policy



OPA Gatekeeper: Policy and Governance for Kubernetes

Tuesday, August 06, 2019

Authors: Rita Zhang (Microsoft), Max Smythe (Google), Craig Hooper (Commonwealth Bank AU), Tim Hinrichs (Styra), Lachie Evenson (Micr

The Open Policy Agent Gatekeeper project can be leveraged to help enforce policies and strengthen governance in your Kubernetes envir through the goals, history, and current state of the project.

The following recordings from the Kubecon EU 2019 sessions are a great starting place in working with Gatekeeper:

- Intro: Open Policy Agent Gatekeeper
- Deep Dive: Open Policy Agent

Motivations

If your organization has been operating Kubernetes, you probably have been looking for ways to control what end-users can do on the clu clusters are in compliance with company policies. These policies may be there to meet governance and legal requirements or to enforce b conventions. With Kubernetes, how do you ensure compliance without sacrificing development agility and operational independence?

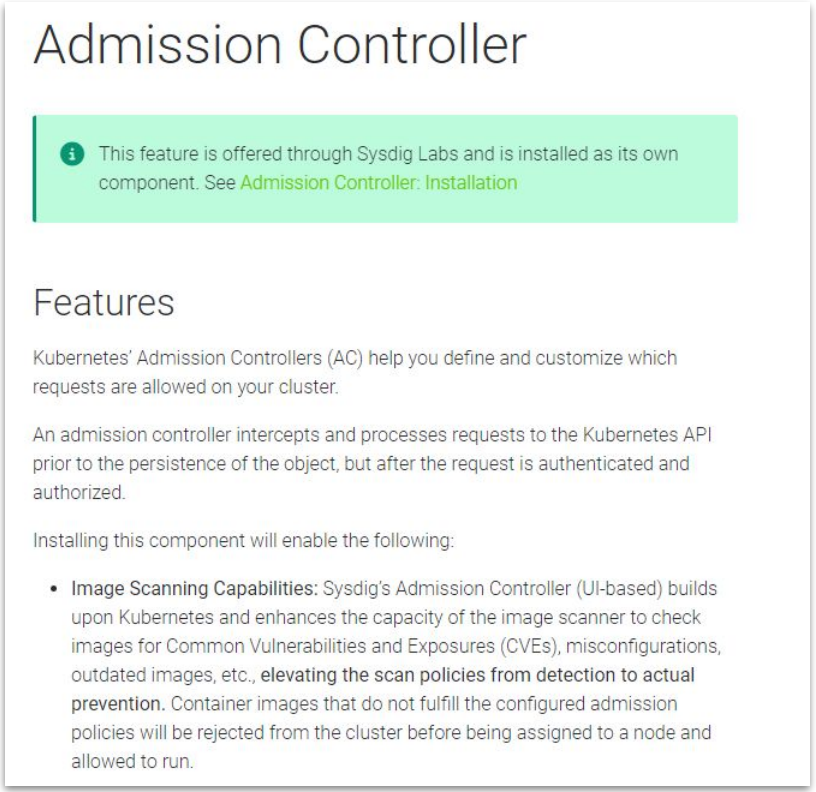
For example, you can enforce policies like:

- All images must be from approved repositories
- All ingress hostnames must be globally unique

Sysdig Also Has An Admission Controller

- Create policies in Sysdig {C,K}SPM and have them applied/enforced at runtime

Policy



Admission Controller

This feature is offered through Sysdig Labs and is installed as its own component. See Admission Controller: Installation

Features

Kubernetes' Admission Controllers (AC) help you define and customize which requests are allowed on your cluster.

An admission controller intercepts and processes requests to the Kubernetes API prior to the persistence of the object, but after the request is authenticated and authorized.

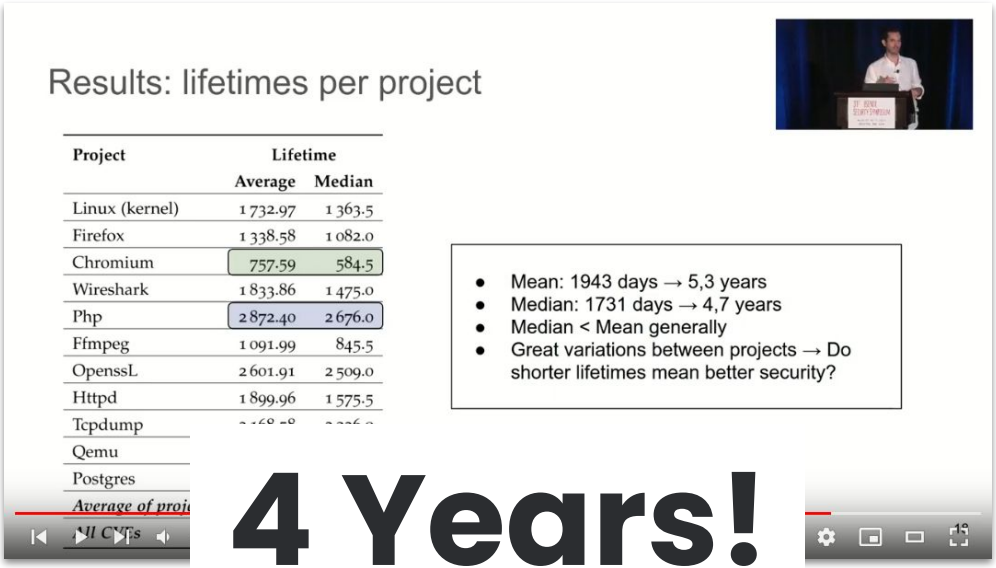
Installing this component will enable the following:

- **Image Scanning Capabilities:** Sysdig's Admission Controller (UI-based) builds upon Kubernetes and enhances the capacity of the image scanner to check images for Common Vulnerabilities and Exposures (CVEs), misconfigurations, outdated images, etc., **elevating the scan policies from detection to actual prevention.** Container images that do not fulfill the configured admission policies will be rejected from the cluster before being assigned to a node and allowed to run.

The Last Line of Defense

How Long Do Vulnerabilities Live in the Code?

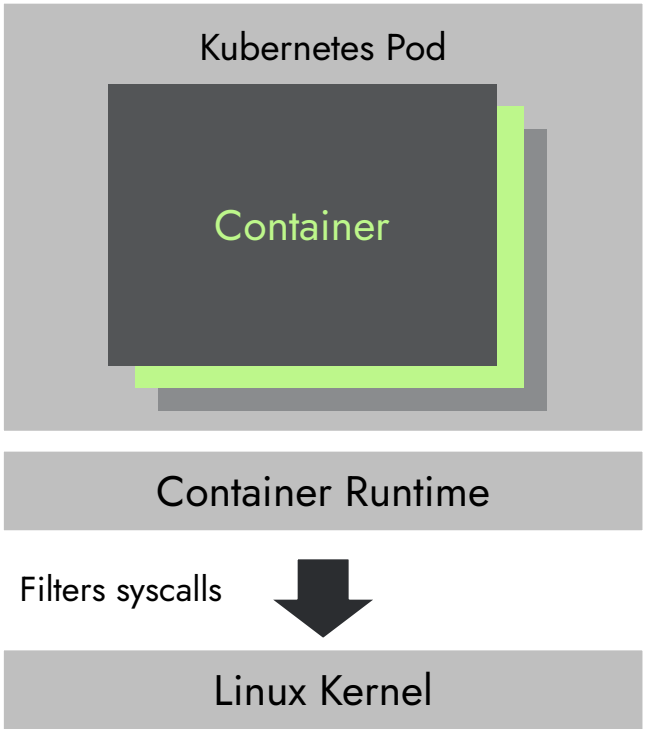
How Long Was the Bug There Before It Was Found?



"...we perform the first large-scale measurement of Free and Open Source Software vulnerability lifetimes...
We find that the average lifetime of a vulnerability is around 4 years"

https://www.usenix.org/system/files/sec22summer_alexopoulos.pdf

Do Containers Contain?



Containers interact directly with the host kernel. Linux has a large attack surface

- 2000+ CVEs since 1999
- 250+ privilege escalations

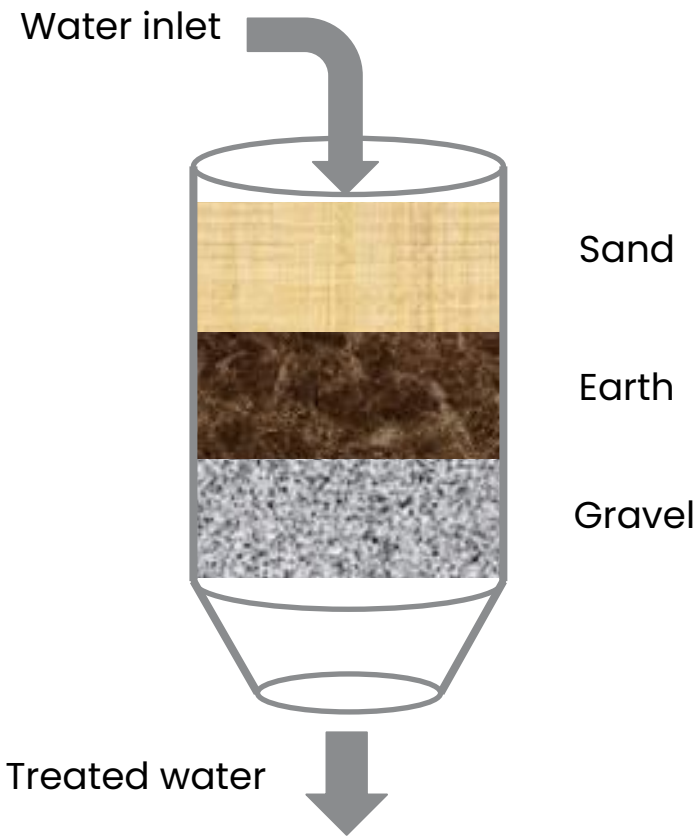
One major Linux or container runtime bug could mean a container escape

Workloads gain access to host, can attack infrastructure, other containers

Based on https://youtu.be/QM-7bAaUiTk

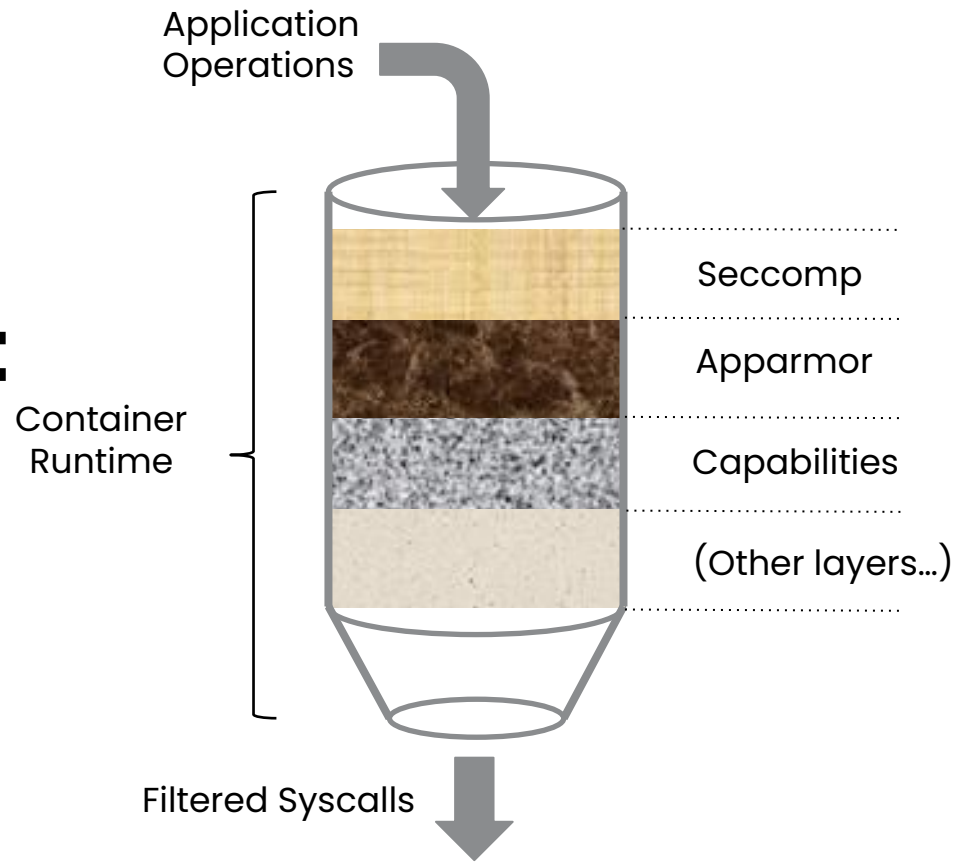
Do Containers Contain?

Is That the Right Question?



Do Containers Contain?

Is That the Right Question?



Open Standard for Cloud-Native Threat Detection

50M+
Downloads

100+
Contributing Companies

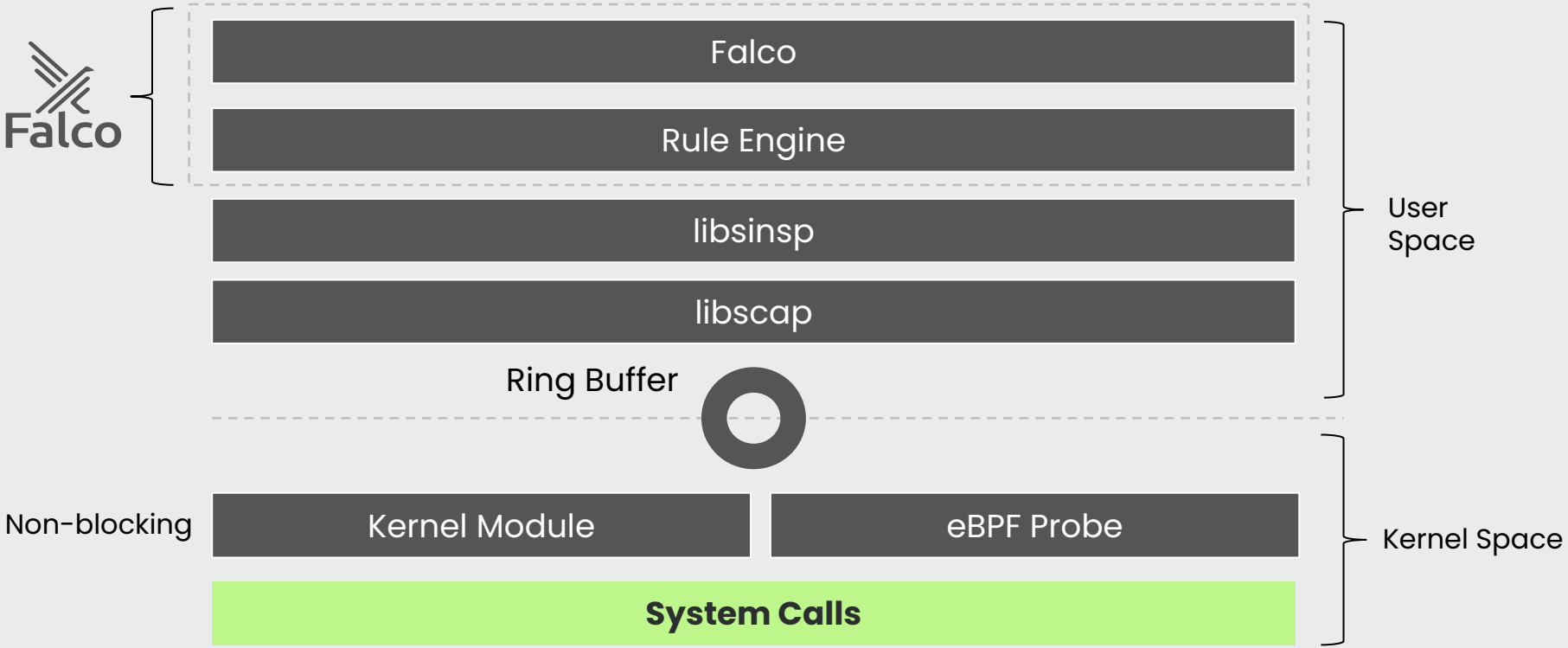
Falco End Users

- Apple
- at&t
- BOSE
- HSBC
- MUFG
- SAMSUNG
- shopify
- Snapchat
- StateFarm
- Uber
- Walmart
- VOLVO

Broad Industry Adoption

aws	Fargate runtime security
Google	gVisor serverless security
IBM	Sysflow telemetry and security
Microsoft	Defender data collection
Red Hat	Data collection
sumo logic	Anomaly detection

Falco Kernel Module Architecture

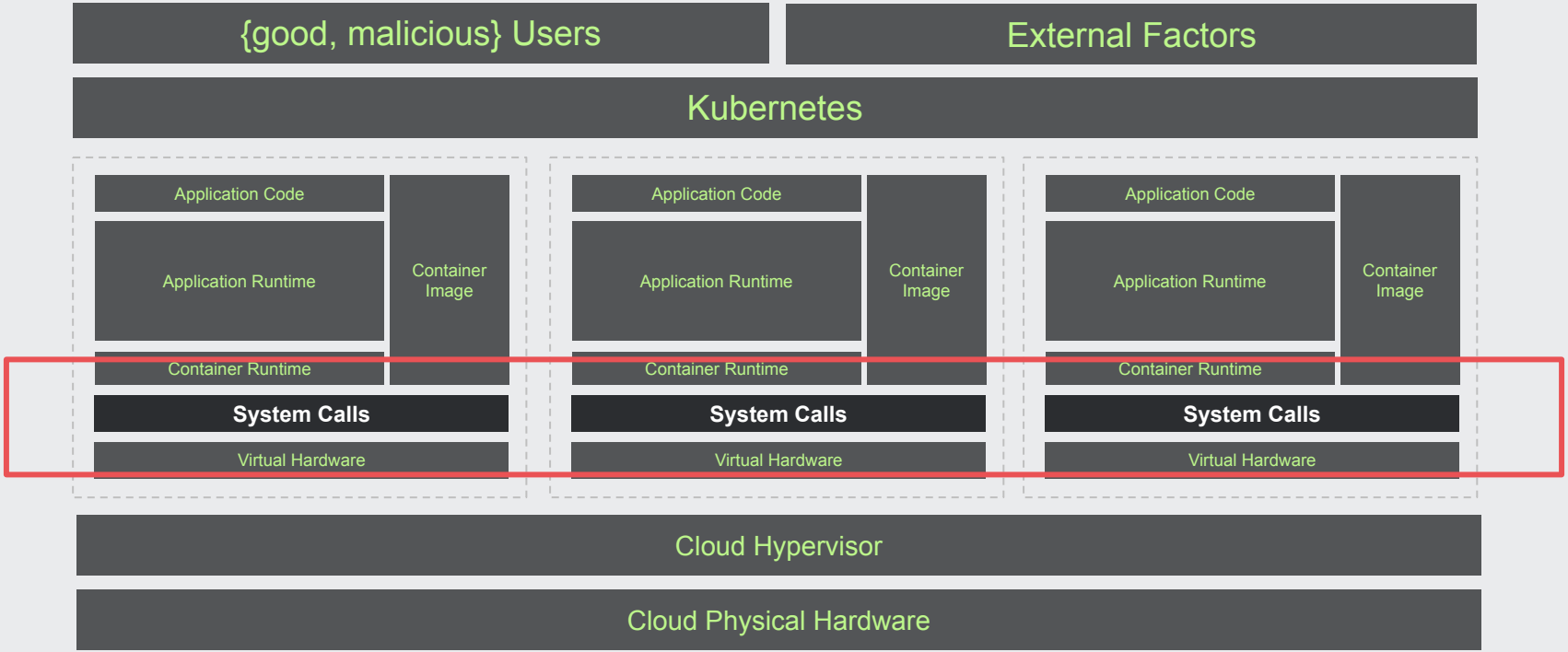


Falco Rules

```
- rule: Fileless execution via memfd_create
  desc: >
    Detect if a binary is executed from memory using the memfd_create technique. This is a well-known defense evasion
    technique for executing malware on a victim machine without storing the payload on disk and to avoid leaving traces
    about what has been executed. Adopters can whitelist processes that may use fileless execution for benign purposes
    by adding items to the list known_memfd_execution_processes.
  condition: >
    spawned_process
    and proc.is_exe_from_memfd=true
    and not known_memfd_execution_processes
  output: Fileless execution via memfd_create (container_start_ts=%container.start_ts proc_cwd=%proc.cwd evt_res=%evt.res proc
  priority: CRITICAL
  tags: [maturity_stable, host, container, process, mitre_defense_evasion, T1620]
```

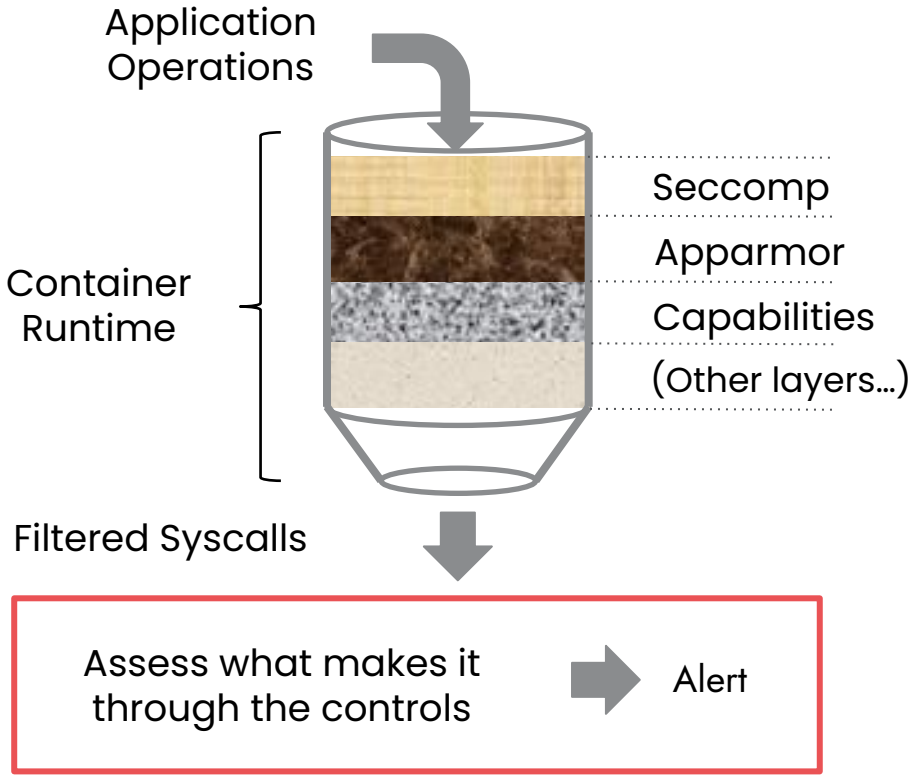
Falco Demo

System Calls: A Good Place to Live



Syscall Quality

With Falco we can apply **rules** to understand the quality of the system calls that are making it through our runtime controls (aka barriers)

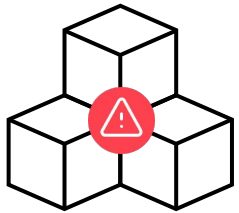


Falco Plugins

...More Than System Calls

Workload System Calls + Software as a Service Audit Logs

Kubernetes and Host System Calls + **Software as a Service and Audit Logs**



Container with critical vulnerability exposed to Internet

Install Crypto Miner

Steal User Credentials

Move Laterally

okta

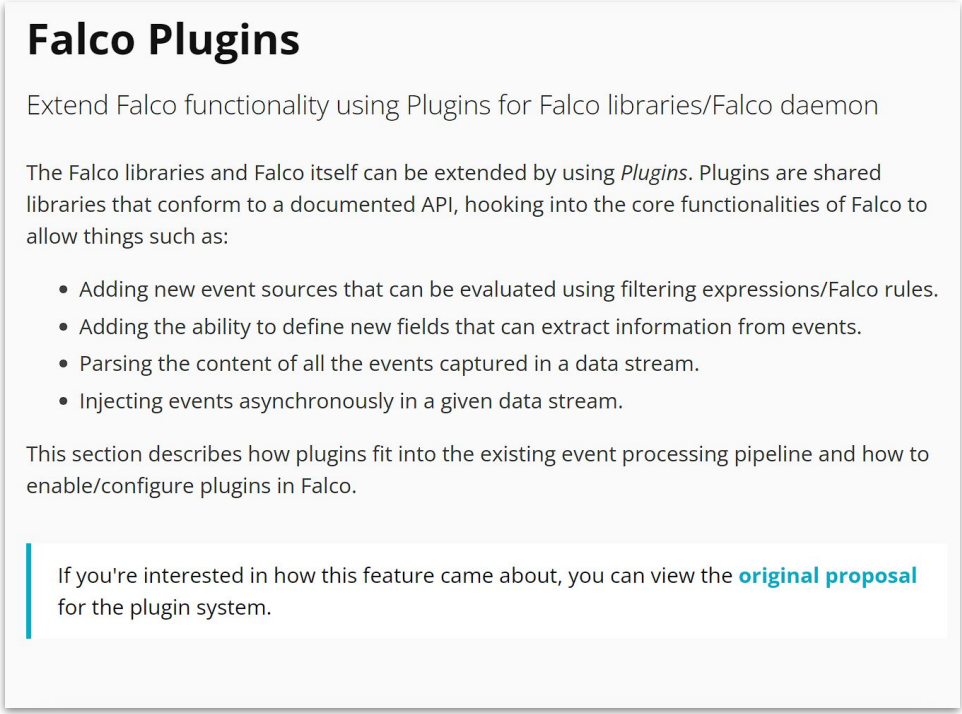
GitHub

Steal Proprietary Data

https://github.com/falcosecurity/plugins/tree/master/plugins

Falco Plugins

With Falco Plugins, Falco can perform threat detection and apply Falco rules on **any streaming data or log file**, simply through building a plugin.



Falco Plugins

Extend Falco functionality using Plugins for Falco libraries/Falco daemon

The Falco libraries and Falco itself can be extended by using *Plugins*. Plugins are shared libraries that conform to a documented API, hooking into the core functionalities of Falco to allow things such as:

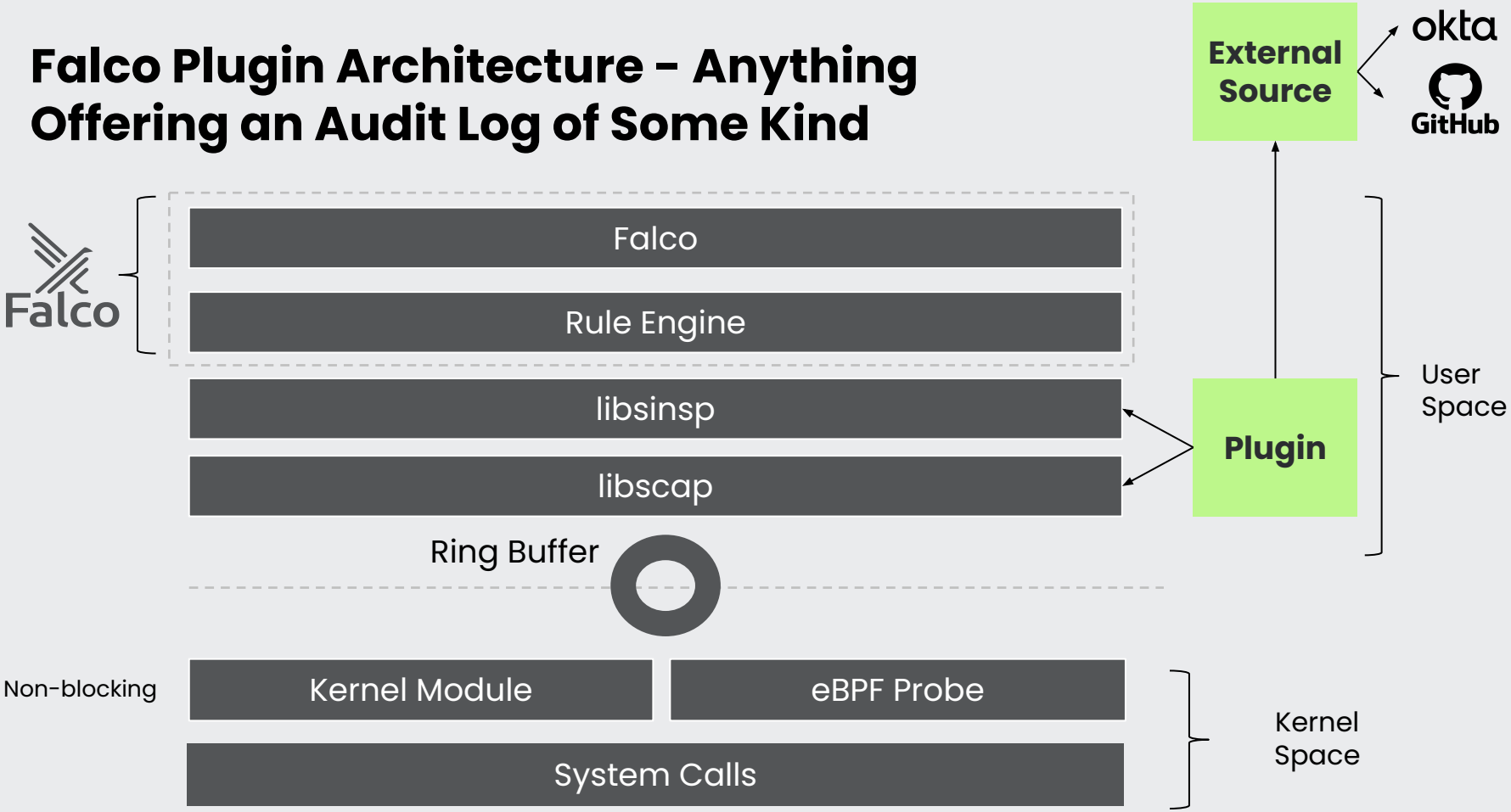
- Adding new event sources that can be evaluated using filtering expressions/Falco rules.
- Adding the ability to define new fields that can extract information from events.
- Parsing the content of all the events captured in a data stream.
- Injecting events asynchronously in a given data stream.

This section describes how plugins fit into the existing event processing pipeline and how to enable/configure plugins in Falco.

> If you're interested in how this feature came about, you can view the **original proposal** for the plugin system.

https://github.com/ccollicutt/swap-image-mutating-webhook

Falco Plugin Architecture - Anything Offering an Audit Log of Some Kind



Open Source Mission

Kubernetes

- Kubernetes mission is to be the best individual Kubernetes cluster possible, and meet all the technical needs of the ecosystem…
- Not, necessarily, to be “enterprisey” and manage many clusters
- Hence, lots of adoption of distributions like EKS, AKS, GKE…

Falco

- Falco is designed to be the best individual sensor possible
- Not necessarily to be the potentially thousands of sensors needed across an entire organization
- Hence, part of why Sysdig exists…to wrap all the “enterprisey” needs around Falco and manage thousands of sensors


sysdig
SECURE
EVERY
SECOND