Secure YAML, Insecure Clusters

Simon Robin

hack::soho 2026

Introduction

Agenda

- Part 1: The Rise of YAML-Centric Kubernetes Security
- Part 2: The Distortion of Kubernetes Security Testing
- Part 3: The Failure of YAML-Centric Security at Scale
- Part 4: Beyond YAML, Human-Led Kubernetes Testing
- Conclusion
- References

Disclaimer

Part 1:

The Rise of YAML-Centric Kubernetes Security

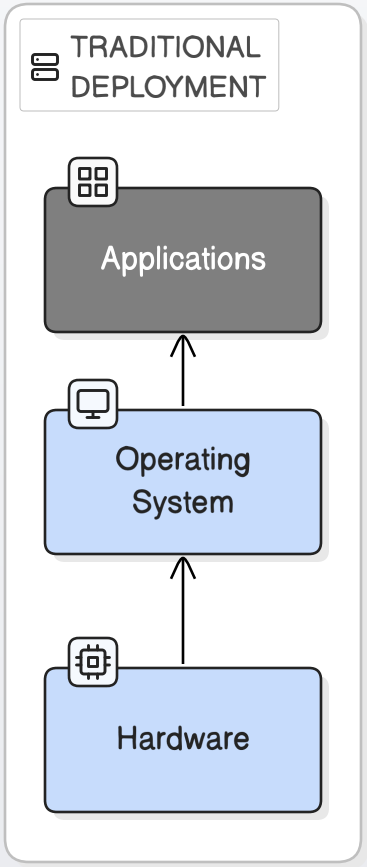
The Origin Story

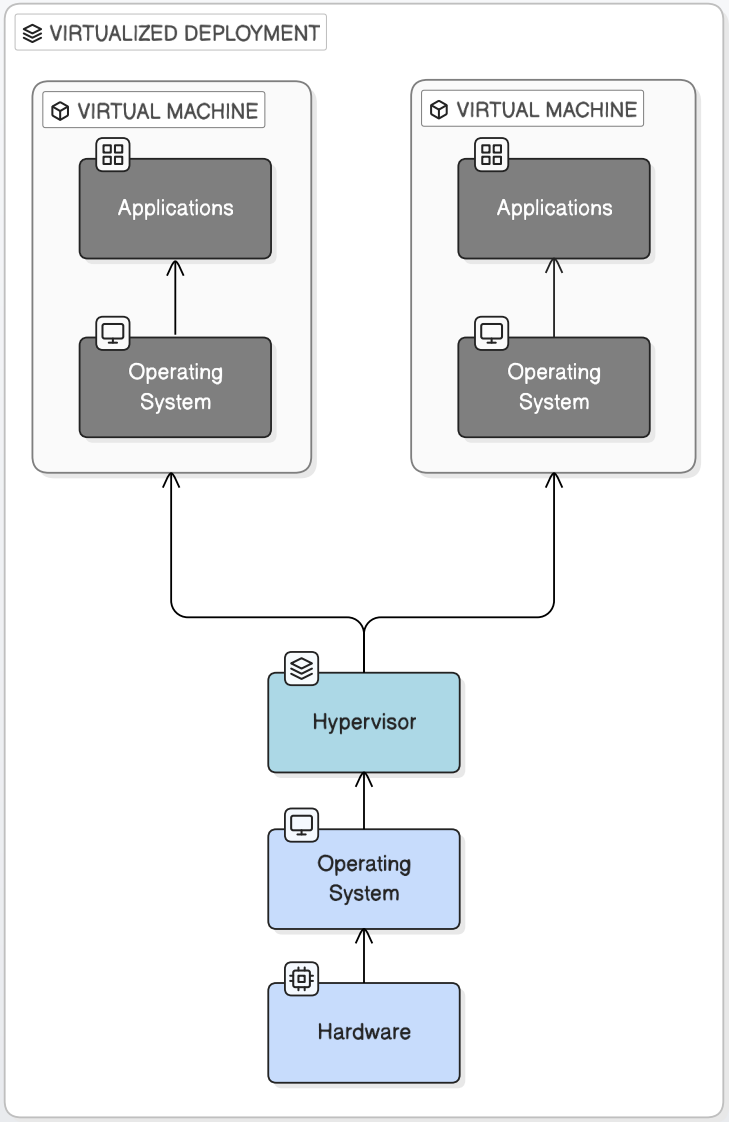
What is Kubernetes?

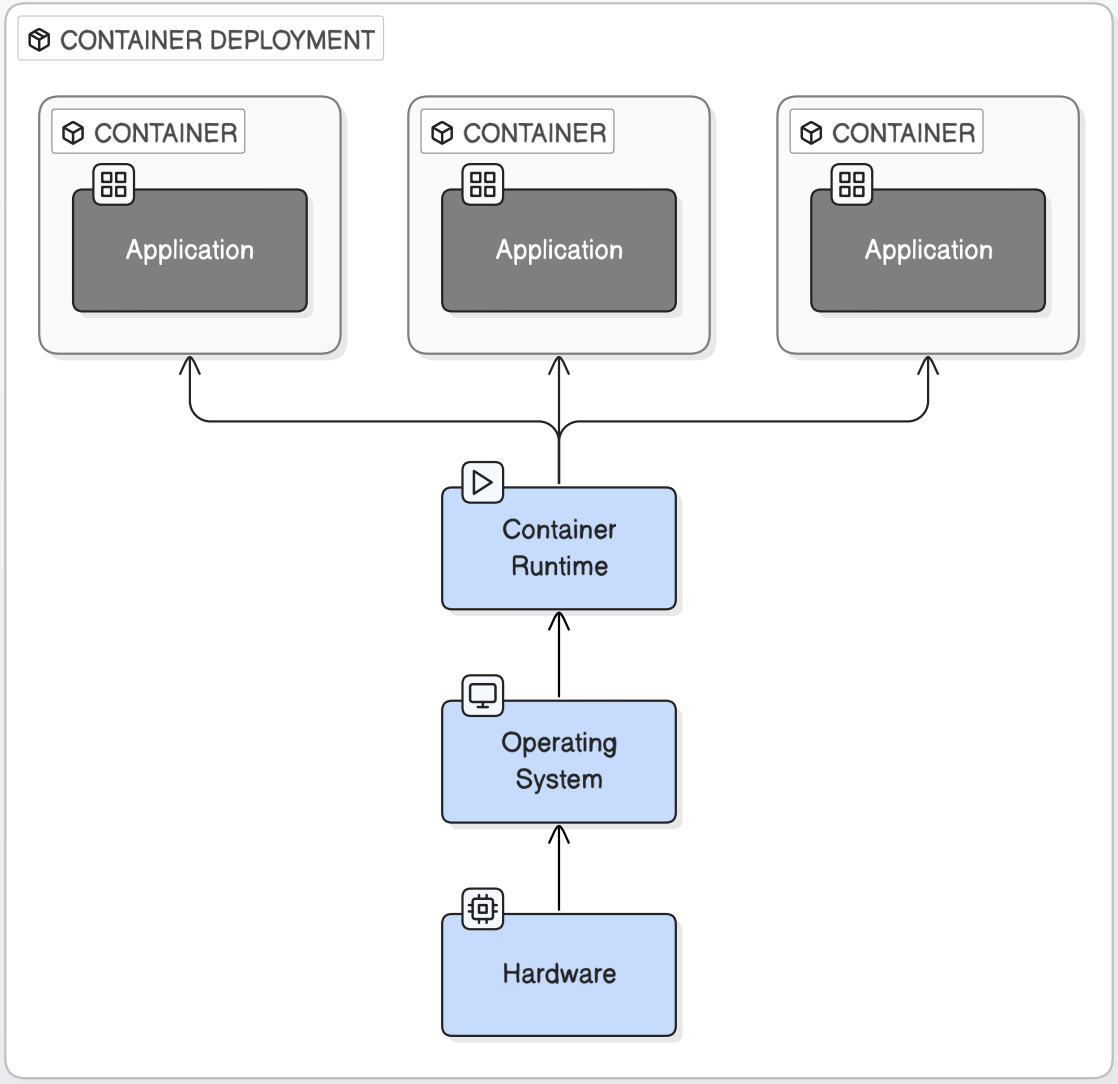


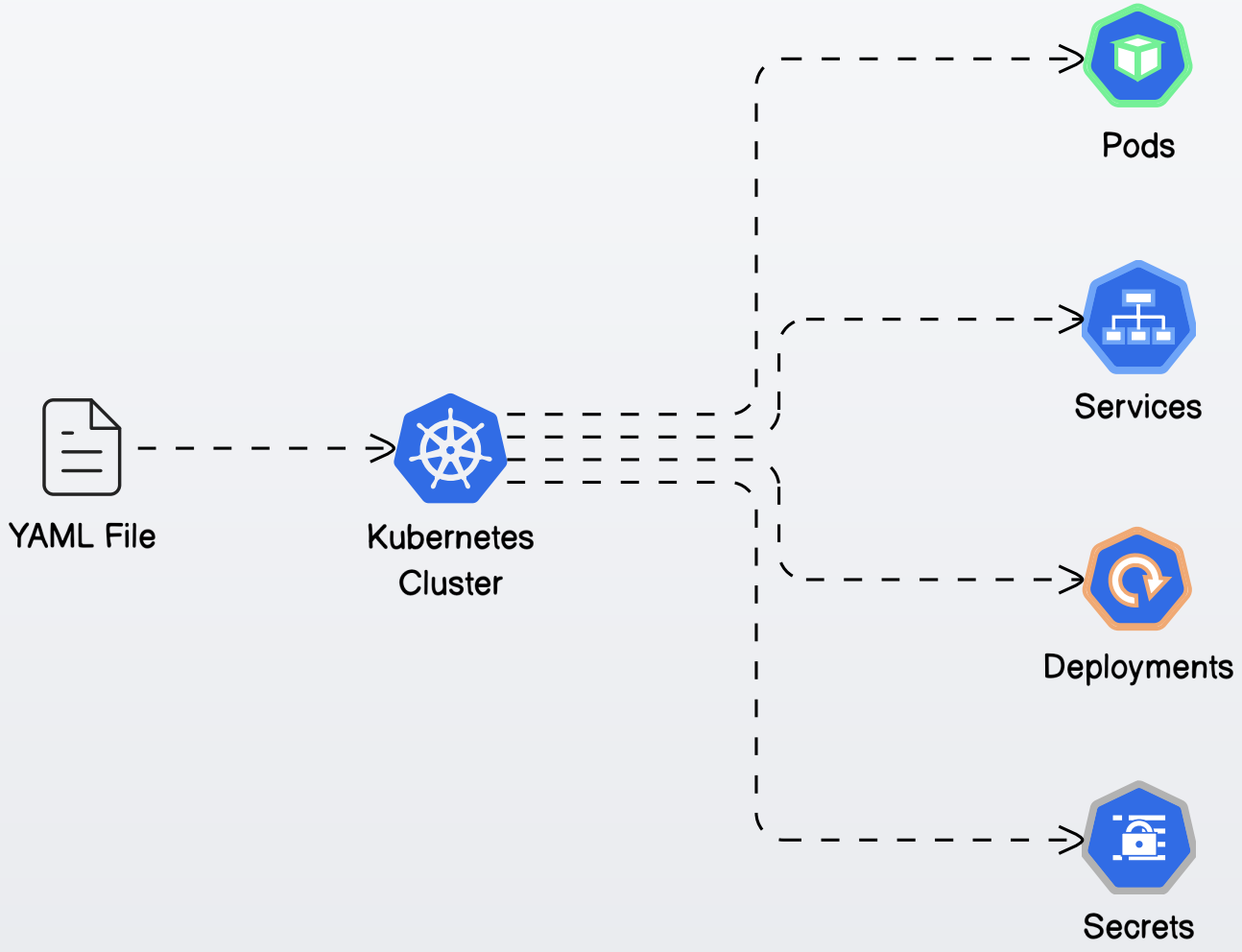
"Production-Grade Container Orchestration"

- K8s
- Platform, OS
- A tool to run and schedule **containerized workloads on multiple hosts**


TRADITIONAL
DEPLOYMENT
Applications
Operating
System
Hardware


VIRTUALIZED DEPLOYMENT
VIRTUAL MACHINE
Applications
Operating System
VIRTUAL MACHINE
Applications
Operating System
Hypervisor
Operating System
Hardware


CONTAINER DEPLOYMENT
CONTAINER
Application
CONTAINER
Application
CONTAINER
Application
Container Runtime
Operating System
Hardware


Pods
Services
YAML File
Kubernetes
Cluster
Deployments
Secrets

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
name: example-deployment
spec:
replicas: 3
template:
  spec:
  containers:
  - name: example-container
    image: example-image
    ports:
    - containerPort: 80
```

O'REILLY®

Production Kubernetes

Building Successful Application Platforms

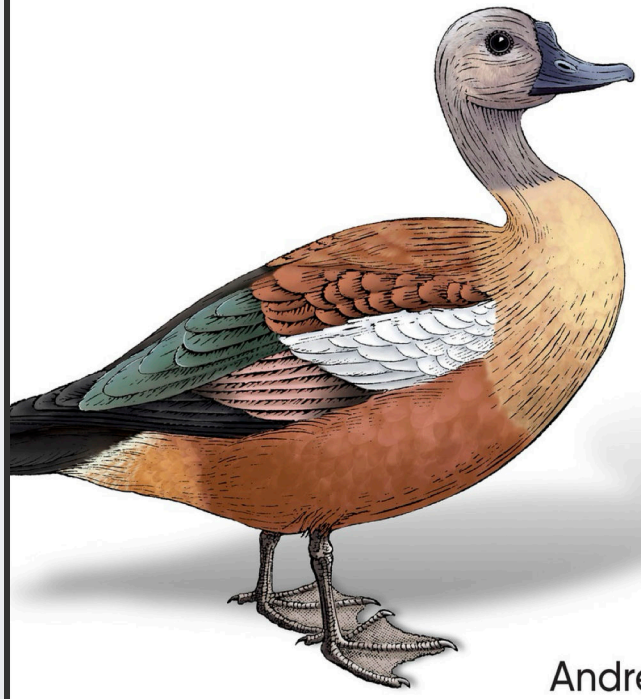


Josh Rosso, Rich Lander,
Alexander Brand & John Harris

O'REILLY®

Hacking Kubernetes

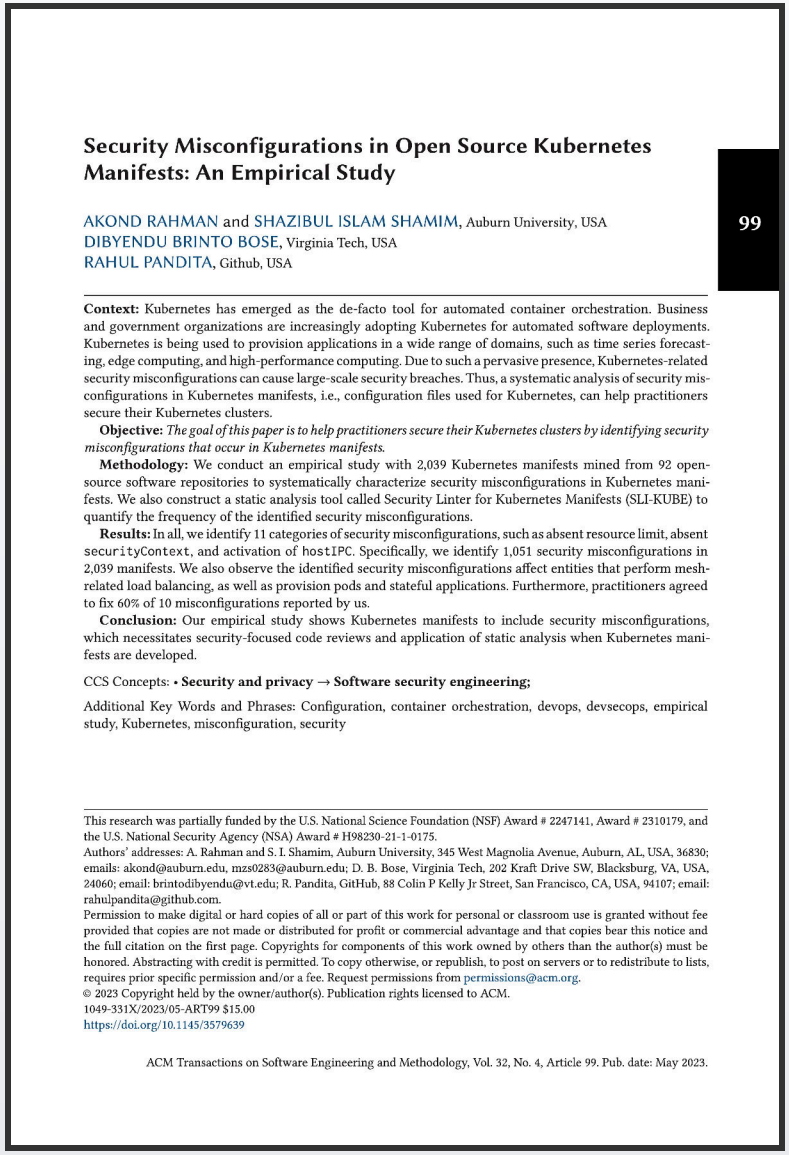
Threat-Driven Analysis and Defense



Andrew Martin &
Michael Hausenblas

Does Configuration Review Matter?

- **2,000+** K8s manifests inspected
- **1,051** security misconfigurations



Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study

99

AKOND RAHMAN and SHAZIBUL ISLAM SHAMIM, Auburn University, USA
DIBYENDU BRINTO BOSE, Virginia Tech, USA
RAHUL PANDITA, Github, USA

Context: Kubernetes has emerged as the de-facto tool for automated container orchestration. Business and government organizations are increasingly adopting Kubernetes for automated software deployments. Kubernetes is being used to provision applications in a wide range of domains, such as time series forecasting, edge computing, and high-performance computing. Due to such a pervasive presence, Kubernetes-related security misconfigurations can cause large-scale security breaches. Thus, a systematic analysis of security misconfigurations in Kubernetes manifests, i.e., configuration files used for Kubernetes, can help practitioners secure their Kubernetes clusters.

Objective: *The goal of this paper is to help practitioners secure their Kubernetes clusters by identifying security misconfigurations that occur in Kubernetes manifests.*

Methodology: We conduct an empirical study with 2,039 Kubernetes manifests mined from 92 open-source software repositories to systematically characterize security misconfigurations in Kubernetes manifests. We also construct a static analysis tool called Security Linter for Kubernetes Manifests (SLI-KUBE) to quantify the frequency of the identified security misconfigurations.

Results: In all, we identify 11 categories of security misconfigurations, such as absent resource limit, absent `securityContext`, and activation of `hostIPC`. Specifically, we identify 1,051 security misconfigurations in 2,039 manifests. We also observe the identified security misconfigurations affect entities that perform mesh-related load balancing, as well as provision pods and stateful applications. Furthermore, practitioners agreed to fix 60% of 10 misconfigurations reported by us.

Conclusion: Our empirical study shows Kubernetes manifests to include security misconfigurations, which necessitates security-focused code reviews and application of static analysis when Kubernetes manifests are developed.

CCS Concepts: • **Security and privacy → Software security engineering;**

Additional Key Words and Phrases: Configuration, container orchestration, devops, devsecops, empirical study, Kubernetes, misconfiguration, security

This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2247141, Award # 2310179, and the U.S. National Security Agency (NSA) Award # H98230-21-1-0175.
Authors' addresses: A. Rahman and S. I. Shamim, Auburn University, 345 West Magnolia Avenue, Auburn, AL, USA, 36830; emails: akond@auburn.edu, mzs0283@auburn.edu; D. B. Bose, Virginia Tech, 202 Kraft Drive SW, Blacksburg, VA, USA, 24060; email: brintodibyendu@vt.edu; R. Pandita, GitHub, 88 Colin P Kelly Jr Street, San Francisco, CA, USA, 94107; email: rahulpandita@github.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1049-331X/2023/05-ART99 $15.00
https://doi.org/10.1145/3579639

- Scanned **10k+** K8s manifests
- **Misconfigurations** represent **59% of security incidents**

ARMO

99% of cloud breaches will have a root cause of **misconfigurations**

Part 2:

The Distortion of Kubernetes Security Testing

The Damage Report

The Unintended Consequences of YAML-Centric Security

- "Kubernetes is easy"
- "Kubernetes is secure by default"
- "No need to know about containers"
- "No Linux knowledge is required"

Cloud vendors propose **quick cluster creation**

↓

Lack hardened **security**

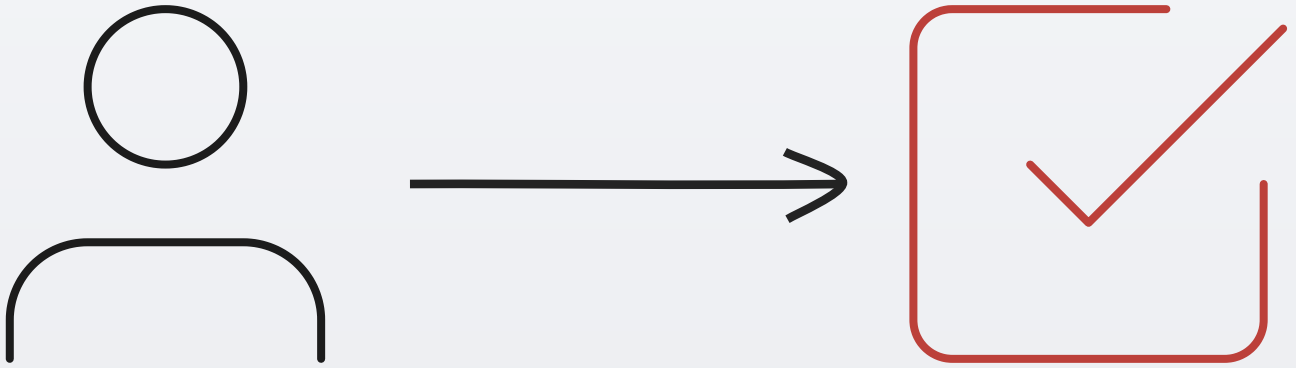


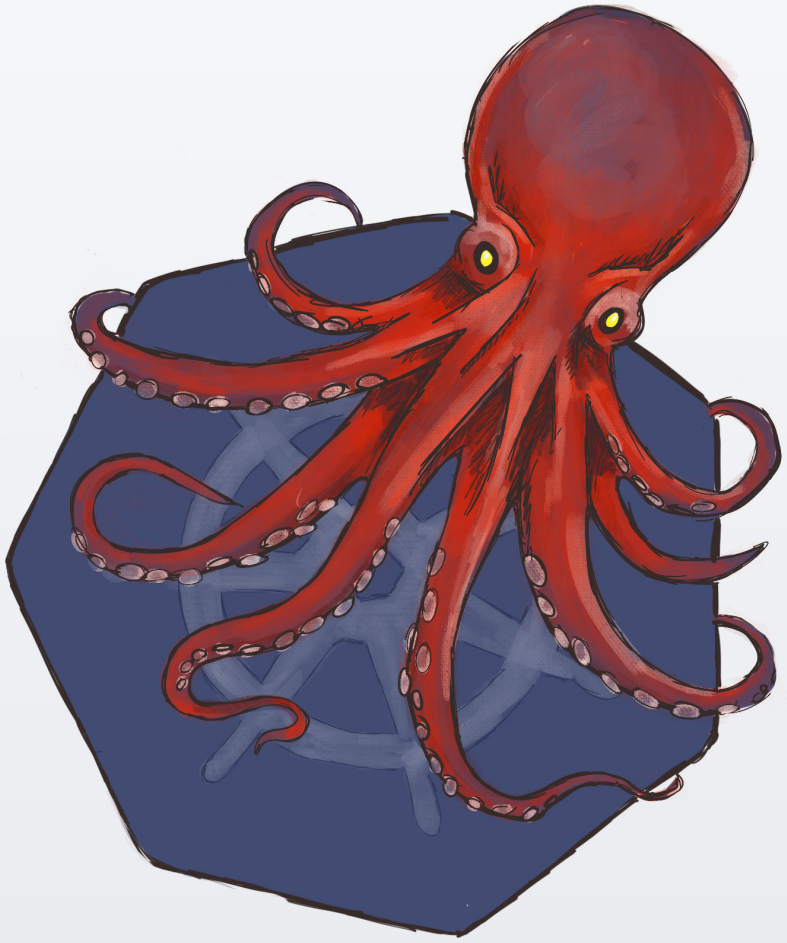
Kubernetes Myths

Debunking Kubernetes misconceptions with proofs, experiments, and real-world validation.

Explore All Myths | Start Myth-Busting

https://kubernetesmyths.com/





https://kubenomicon.com/



Rory McCune

rorym



Iain Smart

iainsmart



Mark Manning

markmanning

The Broken Mental Model of Kubernetes Security

arXiv:2405.19954v1 [cs.CR] 30 May 2024

GenKubeSec: LLM-Based Kubernetes Misconfiguration Detection, Localization, Reasoning, and Remediation

Ehud Malul, Yair Meidan, Dudu Mimran, Yuval Elovici, Asaf Shabtai
Department of Software and Information Systems Engineering
Ben-Gurion University of The Negev

ABSTRACT

A key challenge associated with Kubernetes configuration files (KCFs) is that they are often highly complex and error-prone, leading to security vulnerabilities and operational setbacks. Rule-based (RB) tools for KCF misconfiguration detection rely on static rule sets, making them inherently limited and unable to detect newly-discovered misconfigurations. RB tools also suffer from misdetection, since mistakes are likely when coding the detection rules. Recent methods for detecting and remediating KCF misconfigurations are limited in terms of their scalability and detection coverage, or due to the fact that they have high expertise requirements and do not offer automated remediation along with misconfiguration detection. Novel approaches that employ LLMs in their pipeline rely on API-based, general-purpose, and mainly commercial models. Thus, they pose security challenges, have inconsistent classification performance, and can be costly. In this paper, we propose GenKubeSec, a comprehensive and adaptive, LLM-based method, which, in addition to detecting a wide variety of KCF misconfigurations, also identifies the exact location of the misconfigurations and provides detailed reasoning about them, along with suggested remediation. When empirically compared with three industry-standard RB tools, GenKubeSec achieved equivalent precision (0.990 ± 0.020) and superior recall (0.999 ± 0.026). When a random sample of KCFs was examined by a Kubernetes security expert, GenKubeSec's explanations as to misconfiguration localization, reasoning and remediation were 100% correct, informative and useful. To facilitate further advancements in this domain, we share the unique dataset we collected, a unified misconfiguration index we developed for label standardization, our experimentation code, and GenKubeSec itself as an open-source tool. A video demonstrating our implementation of GenKubeSec can be found here: https://youtu.be/hBehYfDR-aM.

CCS CONCEPTS

• **Computing methodologies** → **Natural language generation**; • **Security and privacy** → **Vulnerability scanners**.

KEYWORDS

Large Language Models (LLMs), Infrastructure-as-Code (IAC), DevOps, Kubernetes (K8s), Artificial Intelligence (AI), Static Analysis

1 INTRODUCTION

The cloud native computing landscape has undergone a significant transformation, with rapid adoption of container-based environments in a relatively short period of time. Kubernetes (K8s) [56] has emerged as the preferred choice for orchestrating containerized applications, driven by the benefits K8s offers, such as efficiency, portability, isolation, scalability, and flexibility [26]. The strong community [54] support surrounding containerization technologies has strengthened their leading role in the development ecosystem.

The plethora of K8s configuration files (KCFs) available in public repositories [18] streamlines the creation and deployment of new environments and applications, permitting developers to easily create cloud environments by duplicating open-source KCFs. However, this convenience comes with its own set of challenges. Open-source KCFs, while widely available, may contain undetected bugs, errors, vulnerabilities, and other types of misconfigurations (abbreviated as misconfigs), potentially leaving deployed systems exposed to cyber threats [17, 35, 37, 52, 83, 92]. Moreover, due to the adoption of K8s in a broad variety of production environments, and the accessibility of open-source KCFs, attackers invest significant efforts in identifying KCF misconfigs that can serve as exploitation entry points into clusters or applications [50, 68].

Given the associated risks, powerful tools capable of detecting KCF misconfigs are required. Rule-based (RB) static analysis tools like SLI-Kube [82], Checkov [14], KubeLinter [33], and Terrascan [90] are recognized tools for assessing the security of KCFs based on predefined rules. However, due to their static nature, RB tools also struggle with adapting to new misconfigs, primarily due to the need for coding new error-prone detection rules. Recent studies on KCF misconfig detection proposed methods that rely on more innovative approaches, such as knowledge graphs and topological graphs [11, 39]. However, these methods face limitations in scalability, expertise requirements, reliance on high-quality input data, and automated remediation, as elaborated in Secs. 7.2 and 7.3.

Recent advances in large language models (LLMs) have proven highly effective in various application domains [34, 41], including cybersecurity [87, 102], where several studies suggested using LLMs in static analysis tasks [2, 59, 61, 65, 103]. Since KCFs are technically (semi-structured) text files, two studies focused specifically on LLM-based methods for KCF misconfig detection and remediation [60, 69]. These studies, however, have the following limitations: (1) each of them utilized LLMs to solve only a sub-task rather than addressing all of the related challenges from end-to-end, including detection, localization, reasoning, and remediation, (2) they utilized pretrained LLMs which had only undergone basic domain adaptation and not LLMs fine-tuned to the specific task, thus demonstrating limited performance, (3) the inspected files were sent to external LLMs via an API, introducing security (privacy) risks and adding unnecessary overhead, (4) relatively small datasets with small number of misconfig types were used for evaluation, and (5) the experimental results were reported very scarcely, so their performance during future inference cannot be guaranteed.

In light of the need to mitigate KCF-related risks, the limitations of existing RB and graph-based methods, the potential of LLMs, and

arXiv:2504.11126v1 [cs.CR] 15 Apr 2025

KubeFence: Security Hardening of the Kubernetes Attack Surface

Carmine Cesarano, Roberto Natella
Università degli Studi di Napoli Federico II, Italy
{carmine.cesarano2, roberto.natella}@unina.it

***Abstract*—Kubernetes (K8s) is widely used to orchestrate containerized applications, including critical services in domains such as finance, healthcare, and government. However, its extensive and feature-rich API interface exposes a broad attack surface, making K8s vulnerable to exploits of software vulnerabilities and misconfigurations. Even if K8s adopts role-based access control (RBAC) to manage access to K8s APIs, this approach lacks the granularity needed to protect specification attributes within API requests. This paper proposes a novel solution, *KubeFence*, which implements finer-grain API filtering tailored to specific client workloads. *KubeFence* analyzes Kubernetes Operators from trusted repositories and leverages their configuration files to restrict unnecessary features of the K8s API, to mitigate misconfigurations and vulnerabilities exploitable through the K8s API. The experimental results show that *KubeFence* can significantly reduce the attack surface and prevent attacks compared to RBAC.**

***Index Terms*—K8s, Attack Surface, API Filtering**

I. Introduction

Kubernetes (K8s) [1] has become the mainstream platform for orchestrating containerized applications, enabling scalable deployment and management across distributed environments. Its adoption spans various critical domains, including finance, healthcare, and government services [2], [3], where security is crucial. As K8s becomes integral to these sensitive areas, ensuring the security of its components, particularly the K8s API, is of the highest importance.

K8s is a project with a huge codebase and a large, complex interface toward clients [4]. This interface provides convenience of use and feature-richness, but it also represents an "attack surface" that exposes the system to security attacks [5]. If these features are provided by vulnerable code, they can be exploited by malicious users to pursue attacks. In the case of K8s, attackers leverage vulnerabilities to run unauthorized workloads, such as cryptojacking and botnets [6]; moreover, attackers can violate the isolation between tenants in the infrastructures, such as disrupting applications and stealing or damaging their data [7].

To avoid such security risks, regulatory frameworks and security agencies are recommending the adoption of secure design practices [8]–[10]. In particular, software systems should adopt the "principle of least privilege", by minimizing the attack surface to only provide access to the strict minimum of features and resources [11], [12]. In the case that a user behaves maliciously, they should be prevented from accessing unnecessary features that could be exploitable [7].

In practice, this approach is quite challenging to apply in complex systems such as K8s. K8s provides a REST API to manage resources, such as *Pods*, *Services*, *Deployments*, and several others. It adopts *role-based access control* (RBAC) [13] to manage API calls that access these resources. However, K8s builds more complex abstractions on top of the REST API. When a resource is configured through the REST API, the API takes in input complex data structures (as YAML payload of a request) to describe the "desired state" of the resource (*specification*). These data structures can contain attributes to use advanced features, such as access to resources on the host machine and other special permissions. These features represent a risky attack surface that can be exploited in the case of software vulnerabilities in K8s.

We argue that RBAC does not suffice to secure K8s, since it only provides access control on resources as a whole, but lacks control over specific features in the specification of these resources. For example, while RBAC can allow users to manage *Pod* resources and disable access to *Deployment* resources, it cannot restrict the values of fields within the specification of a *Pod*. These features can be triggered by a malicious client in order to exploit vulnerabilities in K8s.

In this paper, we analyze the extent of the K8s attack surface and how to harden it. We initially present an analysis of CVE vulnerability records [14] for the K8s project, in which we found that CVEs are only exploitable through specific features of the K8s API interface. Then, we present a solution (*KubeFence*) to provide finer-grain control of the K8s attack surface to harden K8s against exploits and misconfigurations. Our solution analyzes applications that use K8s from trusted repositories (*Kubernetes Operators [15]*), and generates security policies that restrict API access to only the resources explicitly required by the application.

In our evaluation, we defined a catalog of malicious K8s specifications for testing *KubeFence*, based on known CVEs and common misconfigurations. We applied *KubeFence* on several popular applications. Our solution was able to mitigate all tested CVE exploits and misconfigurations. Moreover, we found that *KubeFence* achieved an average of 85% reduction in the attack surface compared to RBAC, while introducing an acceptable performance impact for non-realtime workloads. In particular, we measured an average overhead of 50 ms for cluster management operations (~21%). Since *KubeFence* only applies to cluster management, once containers have been deployed, it does not affect container execution.

The main contributions of this work are:

- We analyze K8s CVEs and found that vulnerable code is only exercised when specific fields are used in API requests.
- We design *KubeFence* to generate and enforce security policies for the K8s API interface. [1]
- We present a catalog of attacks against the K8s API interface, based on CVEs and on common misconfigurations.

[1]Available as open-source at: https://github.com/dessertlab/kubefence/

arXiv:2509.04191v1 [cs.CR] 4 Sep 2025

KubeGuard: LLM-Assisted Kubernetes Hardening via Configuration Files and Runtime Logs Analysis

Omri Sgan Cohen, Ehud Malul, Yair Meidan, Dudu Mimran, Yuval Elovici, Asaf Shabtai
Department of Software and Information Systems Engineering
Ben-Gurion University of The Negev
Israel

Abstract

The widespread adoption of Kubernetes (K8s) for orchestrating cloud-native applications has introduced significant security challenges, such as misconfigured resources and overly permissive configurations. Failing to address these issues can result in unauthorized access, privilege escalation, and lateral movement within clusters. Most existing K8s security solutions focus on detecting misconfigurations, typically through static analysis or anomaly detection. In contrast, this paper presents KubeGuard, a novel runtime log-driven recommender framework aimed at mitigating risks by addressing overly permissive configurations. KubeGuard is designed to harden K8s environments through two complementary tasks: Resource Creation and Resource Refinement. It leverages large language models (LLMs) to analyze manifests and runtime logs reflecting actual system behavior, using modular prompt-chaining workflows. This approach enables KubeGuard to create least-privilege configurations for new resources and refine existing manifests to reduce the attack surface. KubeGuard's output manifests are presented as recommendations that users (e.g., developers and operators) can review and adopt to enhance cluster security. Our evaluation demonstrates that KubeGuard effectively generates and refines K8s manifests for Roles, NetworkPolicies, and Deployments, leveraging both proprietary and open-source LLMs. The high precision, recall, and F1-scores affirm KubeGuard's practicality as a framework that translates runtime observability into actionable, least-privilege configuration guidance.

1 Introduction

Kubernetes (K8s) is an open-source container orchestration platform that has revolutionized how cloud-native applications are deployed and managed. By automating container scheduling, scaling, and resource allocation, K8s enables highly efficient and resilient distributed infrastructures [45]. However, as the complexity of K8s clusters increases, securing them becomes challenging, particularly due to the prevalence of misconfigurations in K8s configuration files (manifests) [76]. These misconfigurations expose organizations to risks such as unauthorized access, lateral movement, and privilege escalation, underscoring the need for robust and adaptive security solutions that help users apply secure configurations and reduce the likelihood of human error during policy definition [33].

Most K8s security solutions rely on predefined configurations [87, 88], static analysis [24, 57, 81], or industry-standard rule-based tools [16, 21, 90, 95]. Basic hardening can be achieved using native K8s controls, for example, replacing default allow-all settings with tailored NetworkPolicies or refining overly permissive RBAC Roles; however, both depend on precise configuration and are therefore challenging to maintain at scale. Moreover, although these methods effectively reduce the attack surface, they often lack visibility into runtime behavior and struggle to adapt to dynamic environments and evolving threats. Even advanced approaches leveraging artificial intelligence (AI) and large language models (LLMs) [54, 58, 102] typically exhibit limited responsiveness to real-time system states. Several studies have employed runtime logs for monitoring and anomaly detection in K8s [31, 37, 52, 84]. More recent methods leveraged various logs using AI, but fed them into static pipelines [102] or generated configurations for just a single resource type (e.g., only NetworkPolicies) [36], often not even K8s-native [34].

To address the aforementioned limitations of existing approaches, keep pace with the demands of dynamic environments, and support the required shift from reactive diagnostics to proactive defense, this paper introduces *KubeGuard*: a novel LLM-based framework for dynamically hardening K8s systems based on prompt-chaining workflows [3] that leverage multi-source runtime logs (a log-driven hardening example is provided in Fig. 1). KubeGuard uses audit logs [40], provenance data [25], and network traffic flows [19] to extract actionable insights and systematically create or refine K8s manifests. These heterogeneous logs are aggregated into structured activity records, allowing KubeGuard to correlate user actions, network activity, and resource-level changes. The aggregation preserves behavioral security context while compressing high-volume logs into concise summaries (reducing token count by 99.96%), enabling efficient LLM analysis within token limits.

KubeGuard acts as a security-aware recommender framework that assists developers and system owners in hardening K8s environments by suggesting least-privilege configurations based on observed application behavior. It addresses two complementary tasks, both aimed at reducing the attack surface of K8s-based systems: *Resource Creation* and *Resource Refinement*. Resource Creation focuses



Figure 1: Example of log-driven manifest hardening. An overly permissive K8s Role is refined by aligning the manifest with observed runtime behavior in aggregated logs.

- "Automated Cloud Risk Discovery"
- "Attack Path Detection"
- "Continuous Security Validation"



Scanning ≠ Pentesting

Part 3:

The Failure of YAML-Centric Security at Scale

The Existential Risk

Limits of Static Analysis

Is it enough?

- **Runtime** network exposure
- Temporal **privilege escalation** conditions
- Interactions with **workload behavior**



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING i

Configuration Defects in Kubernetes

Yue Zhang, Uchswas Paul, Marcelo d'Amorim, and Akond Rahman, *Member, IEEE*

Abstract—Kubernetes is a tool that facilitates rapid deployment of software. Unfortunately, configuring Kubernetes is prone to errors. Configuration defects are not uncommon and can result in serious consequences. This paper reports an empirical study about configuration defects in Kubernetes with the goal of helping practitioners detect and prevent these defects. We study 719 defects that we extract from 2,260 Kubernetes configuration scripts using open source repositories. Using qualitative analysis, we identify 15 categories of defects. We find 8 publicly available static analysis tools to be capable of detecting 8 of the 15 defect categories. We find that the highest precision and recall of those tools are for defects related to data fields. We develop a linter to detect two categories of defects that cause serious consequences, which none of the studied tools are able to detect. Our linter revealed 26 previously-unknown defects that have been confirmed by practitioners, 19 of which have already been fixed. We conclude our paper by providing recommendations on how defect detection and repair techniques can be used for Kubernetes configuration scripts. The datasets and source code used for the paper are publicly available online.

Index Terms—configuration, container orchestration, defect, devops, Kubernetes

1 Introduction

The use of multiple containers to deploy software projects is a common practice today [83], e.g., Paypal uses 200,000 containers to speed up financial transactions [58]. Setting up and managing multiple containers manually is considered impractical and prone to errors [16], [65], [74]. For that reason, the practice of container orchestration advocates for automated management of containers with tools, such as Kubernetes [64] that has yielded benefits for organizations [46]. OpenAI reported that Kubernetes enabled a reduction of deployment time from "a couple of months" to "two or three days" [46]. Kubernetes usage aided Adidas to reduce the load time for their e-commerce website by half, and increase the release frequency from once every 4~6 weeks to 3~4 times a day [46].

Unfortunately, Kubernetes configuration scripts are not immune to defects. In March 2023, the social media platform Reddit experienced a 5 hour-long outage that impacted millions of its users [41], [70]. The outage occurred because of a defect in a configuration script affecting the network traffic between containers [40], [41]. Figure 1 presents an YAML code snippet showcasing how certain Kubernetes-related configurations were specified when the outage occurred. The defect is due to the incorrect definition of configuration options nodeSelector and peerSelector, which used the value node-role.kubernetes.io/master instead of node-role.kubernetes.io/control-plane. The string master in the configuration value became obsolete with the release of Kubernetes 1.24 [45]. The entities nodeSelector and peerSelector are responsible to route the network traffic across containers. As a result of this defect, traffic was routed to a destination that does not exist, resulting in the outage. This defect illustrates the importance of understanding configuration defects in Kubernetes-related computing infrastructure.

```
metadata:
    annotations:
  ...
spec:
  asNumber: 0
  nodeSelector: has(node-role.kubernetes.io/master)
  nodeSelector: has(node-role.kubernetes.io/control-plane)
  peerIP: ...
  peerSelector: has(node-role.kubernetes.io/master)
  peerSelector: has(node-role.kubernetes.io/control-plane)
```

Fig. 1: Excerpt of the configuration defect that caused the Reddit outage [41].

Our paper presents an empirical study about configuration defects in Kubernetes with the goals of assisting practitioners in preventing defects and guiding researchers in developing automated tools to detect those defects. The results of the study enable researchers and practitioners (i) to gain insights about the defects in Kubernetes-based computing infrastructure; (ii) to assess the capabilities of existing tools in identifying defects; and (iii) to develop techniques to identify latent defects that occur during Kubernetes-based configuration management. While the importance of defect categorization have been well-acknowledged in software engineering research [15], [35], [69], a systematic characterization of defects related to Kubernetes configuration management remains under explored.

We answer the following research questions:

- **RQ1 [Categories]**: What are the categories of defects in Kubernetes configuration management?
- **RQ2 [Consequences and Fix Patterns]**: What categories of consequences and fix patterns map to defects that occur during Kubernetes configuration management?
- **RQ3 [Tool Support]**: How frequently do static analysis tools support the detection of defects that occur during Kubernetes configuration management?

Yue Zhang is with Graduate School of Computer Science and Software Engineering, Auburn University, Auburn, Alabama, USA
Uchswas Paul is with Graduate School of Computer Science, NC State University, Raleigh, NC, USA
Marcelo d'Amorim is with the Faculty of Computer Science, NC State University, Raleigh, NC, USA
Akond Rahman is with the Faculty of Computer Science and Software Engineering, Auburn University, Auburn, Alabama, USA

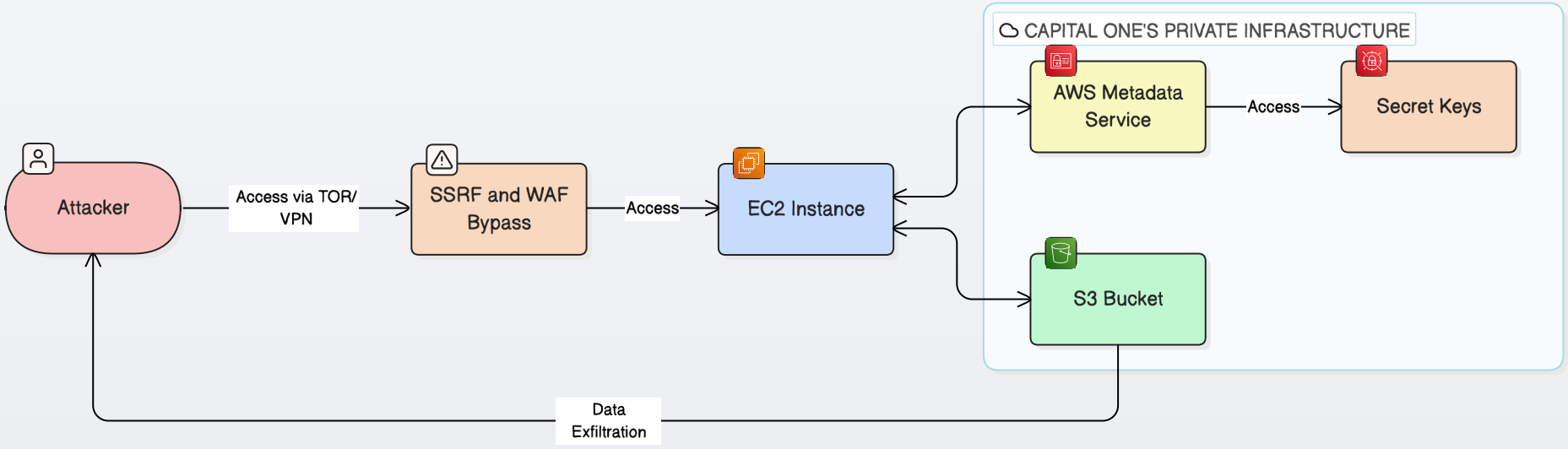
*Security is two different things. It's a **feeling**, and it's a **reality**. And they're different.*
*You could **feel secure**, even if you're not, and you can **be secure**, even if you don't feel it.*



Bruce Schneier



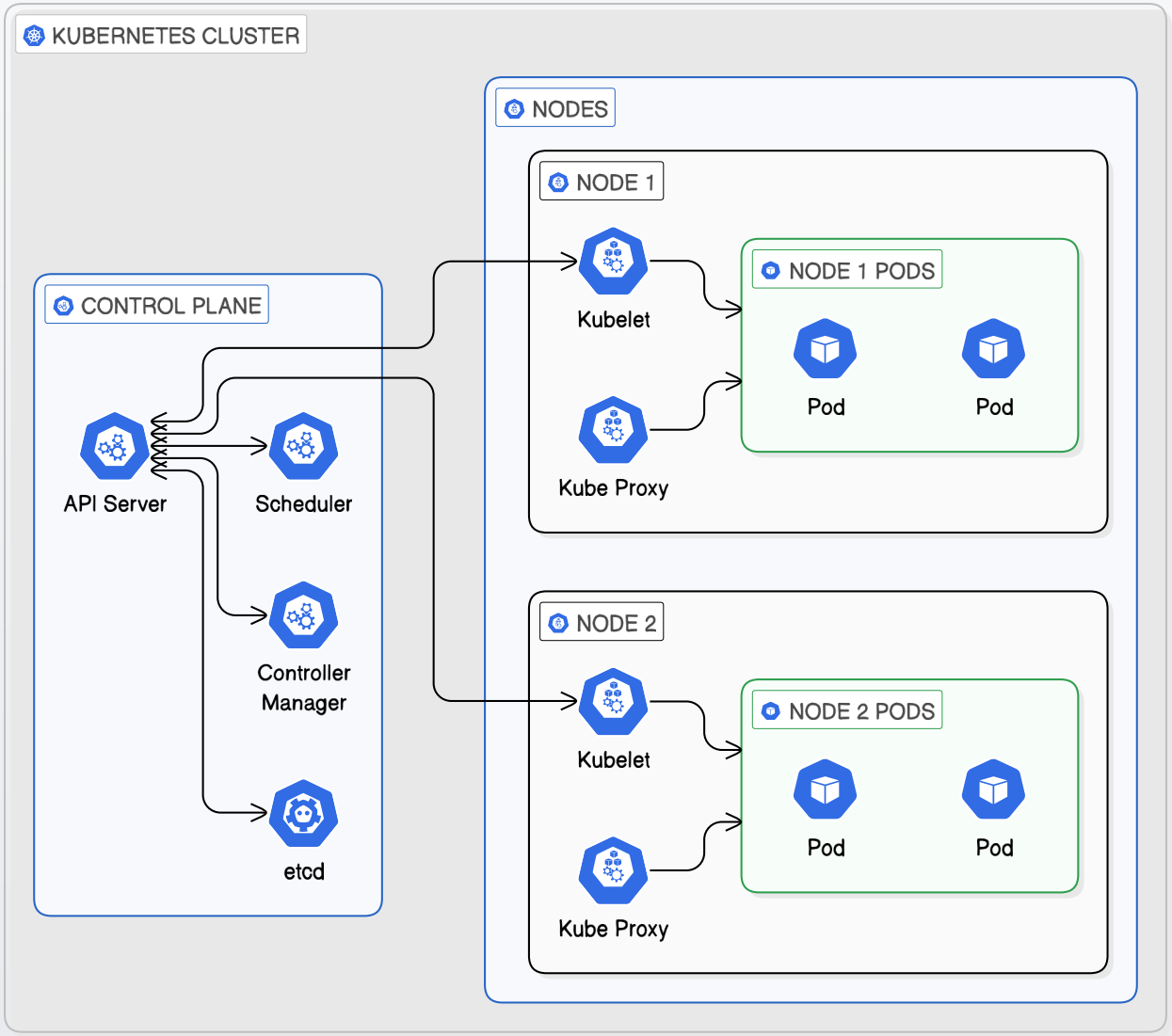
- One of the largest data breaches
- 100 million individuals' data exposed
- No 0-day
- Misconfigurations chained together

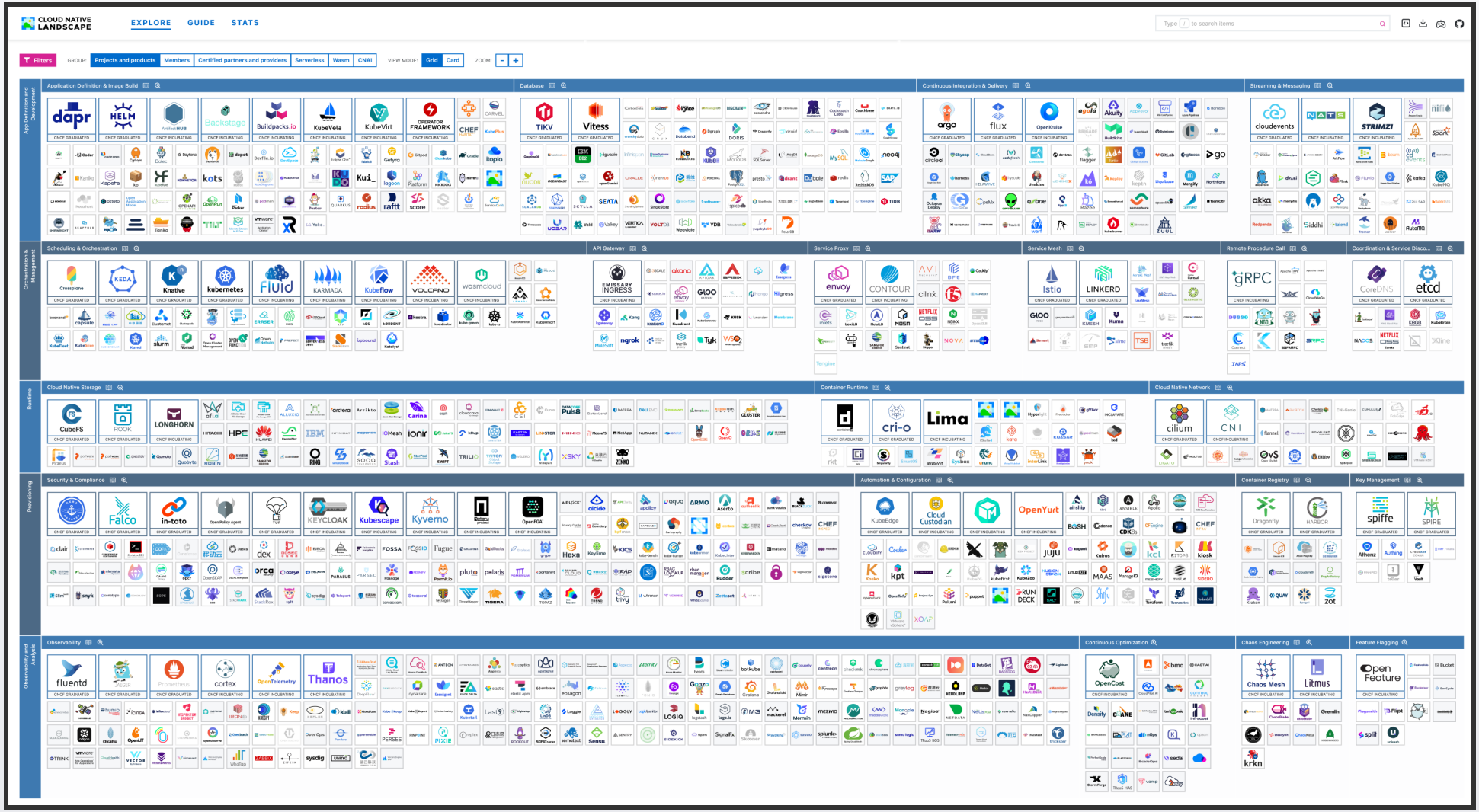

CAPITAL ONE'S PRIVATE INFRASTRUCTURE
Attacker
Access via TOR/ VPN
SSRF and WAF Bypass
Access
EC2 Instance
AWS Metadata Service
Access
Secret Keys
S3 Bucket
Data Exfiltration

- Tools don't **chain context**
- **Automation** finds **problems**
- **Humans** prove **impact**
- Security teams get **flooded** with **low- and medium-risk findings**

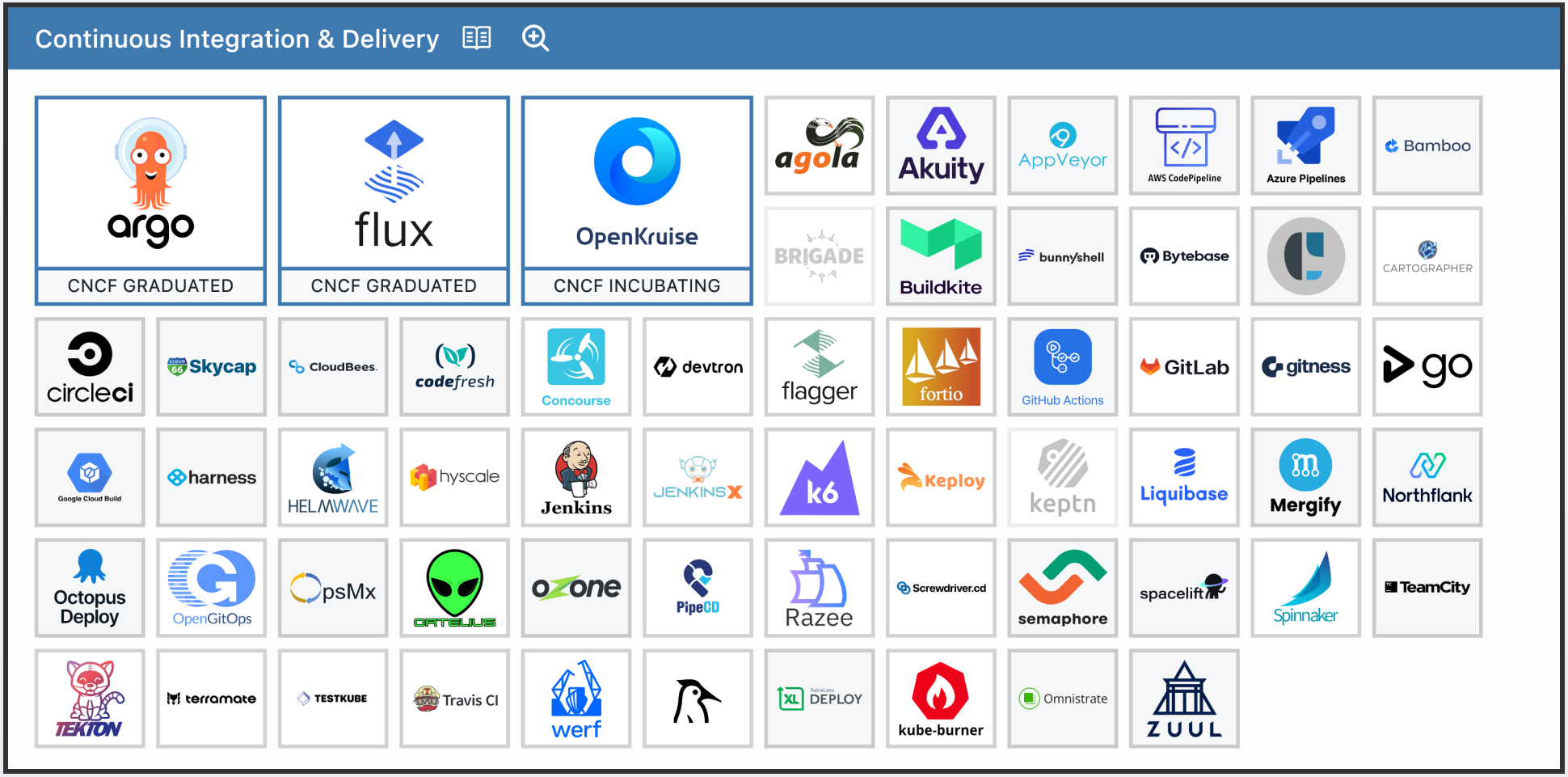
The Expanding Kubernetes Threat Landscape

- **Kubernetes market:**
  - In 2022, valued at **$1.8 billion**
  - In 2031, estimated at **$9.7 billion**
- **Kubernetes security solutions market:**
  - In 2022, valued at **$1.2 billion**
  - In 2031, estimated at **$10.7 billion**
- In 2024, **60% of enterprises** were using Kubernetes
- In 2027, **90% of enterprises** are expected to use it


KUBERNETES CLUSTER
NODES
NODE 1
NODE 1 PODS
Kubelet
Pod
Pod
Kube Proxy
CONTROL PLANE
API Server
Scheduler
Controller Manager
etcd
NODE 2
NODE 2 PODS
Kubelet
Pod
Pod
Kube Proxy

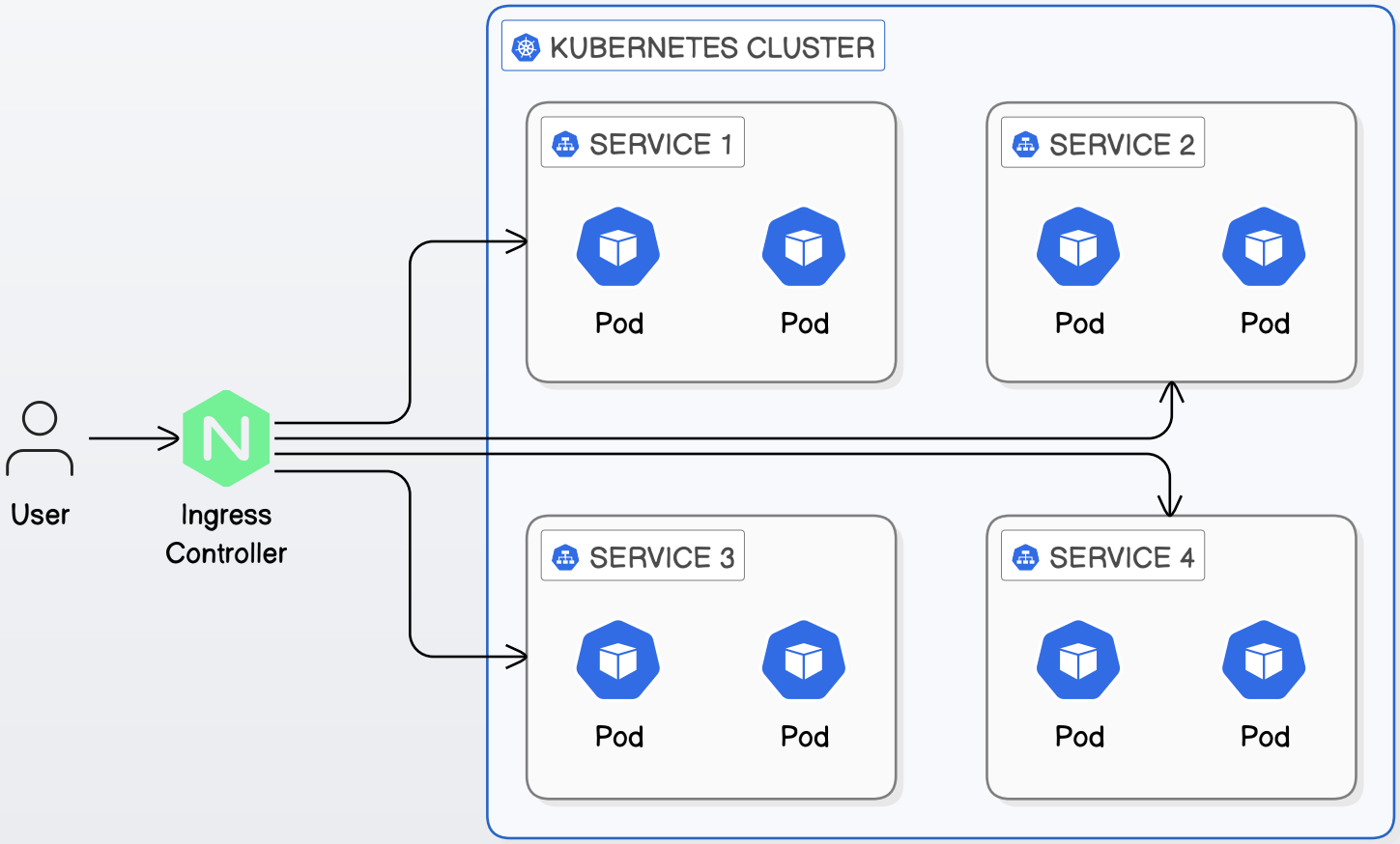

CLOUD NATIVE LANDSCAPE
EXPLORE
GUIDE
STATS
Type / to search items
Filters
GROUP:
Projects and products
Members
Certified partners and providers
Serverless
Wasm
CNAI
VIEW MODE:
Grid
Card
ZOOM:
App Definition and Development
Application Definition & Image Build
Database
Continuous Integration & Delivery
Streaming & Messaging
Orchestration & Management
Scheduling & Orchestration
API Gateway
Service Proxy
Service Mesh
Remote Procedure Call
Coordination & Service Disco...
Runtime
Cloud Native Storage
Container Runtime
Cloud Native Network
Provisioning
Security & Compliance
Automation & Configuration
Container Registry
Key Management
Observability and Analysis
Observability
Continuous Optimization
Chaos Engineering
Feature Flagging
CNCF GRADUATED
CNCF INCUBATING

Continuous Integration & Delivery



- On **Amazon Elastic Kubernetes Service (EKS)**, the 1st attack attempt occurred within **28 minutes** of cluster creation
- On **Azure Kubernetes Service (AKS)**, within **18 minutes**




KUBERNETES CLUSTER
SERVICE 1
SERVICE 2
SERVICE 3
SERVICE 4
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
User
Ingress
Controller

Part 4:

Beyond YAML, Human-Led Kubernetes Testing

The Path Forward

A Multi-Layer Approach

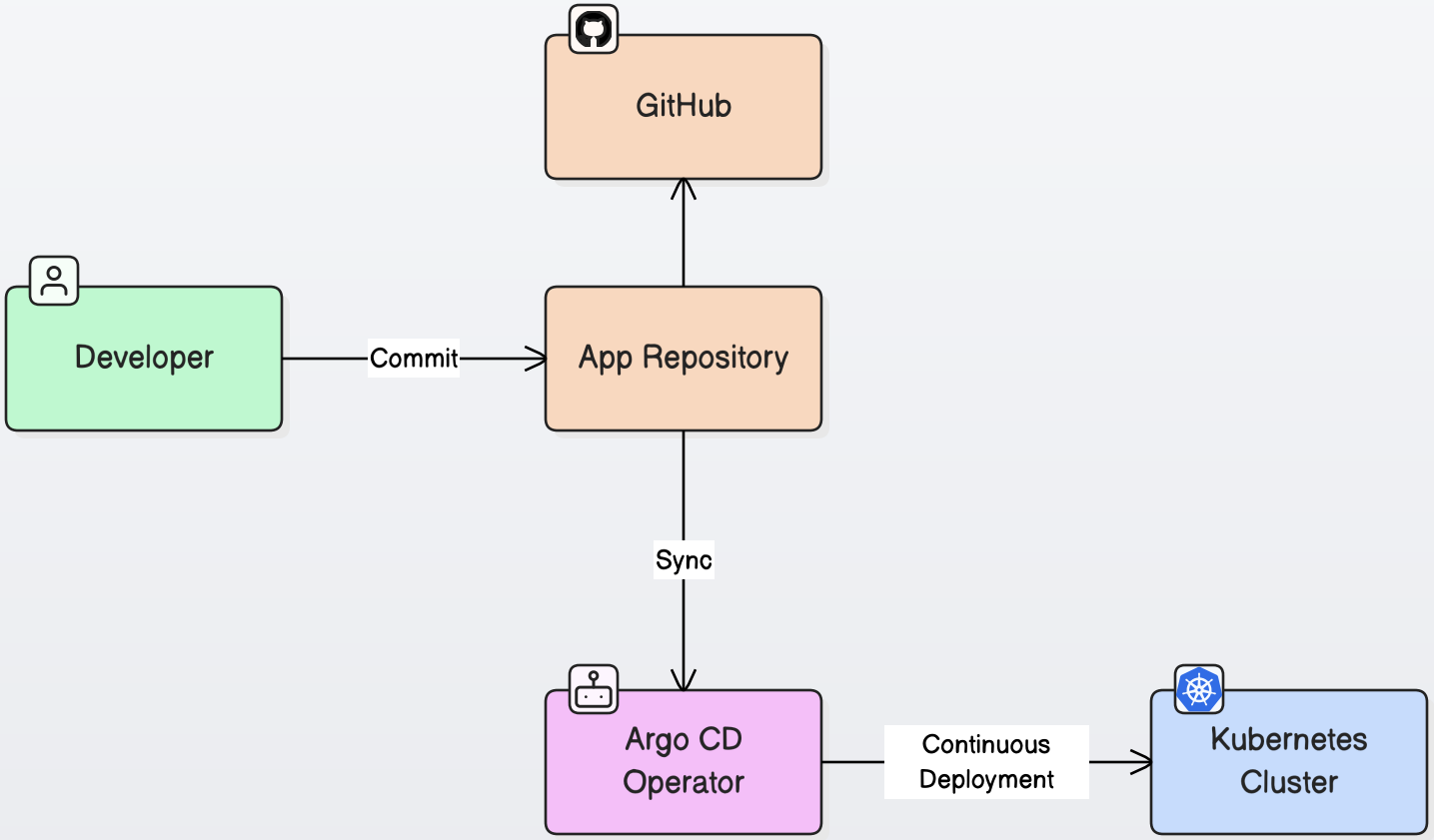
- Identify the **type of client**
- Explain that **Kubernetes is multi-layered**
- Perform **key assessments** during initial calls

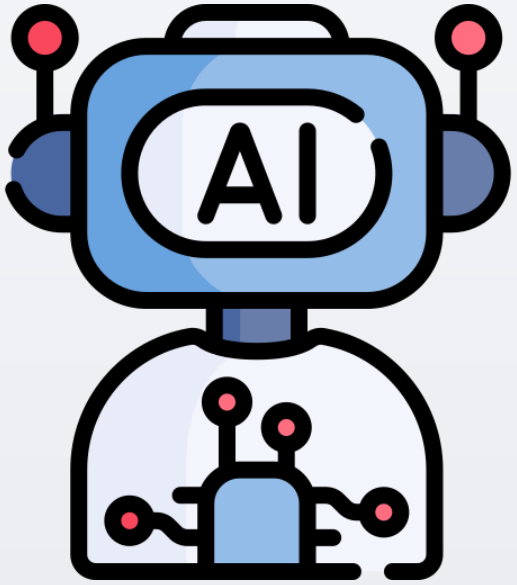
- Cluster configuration review
- Workload security
- Network ingress & egress
- Node / OS security
- Cloud & Supply chain

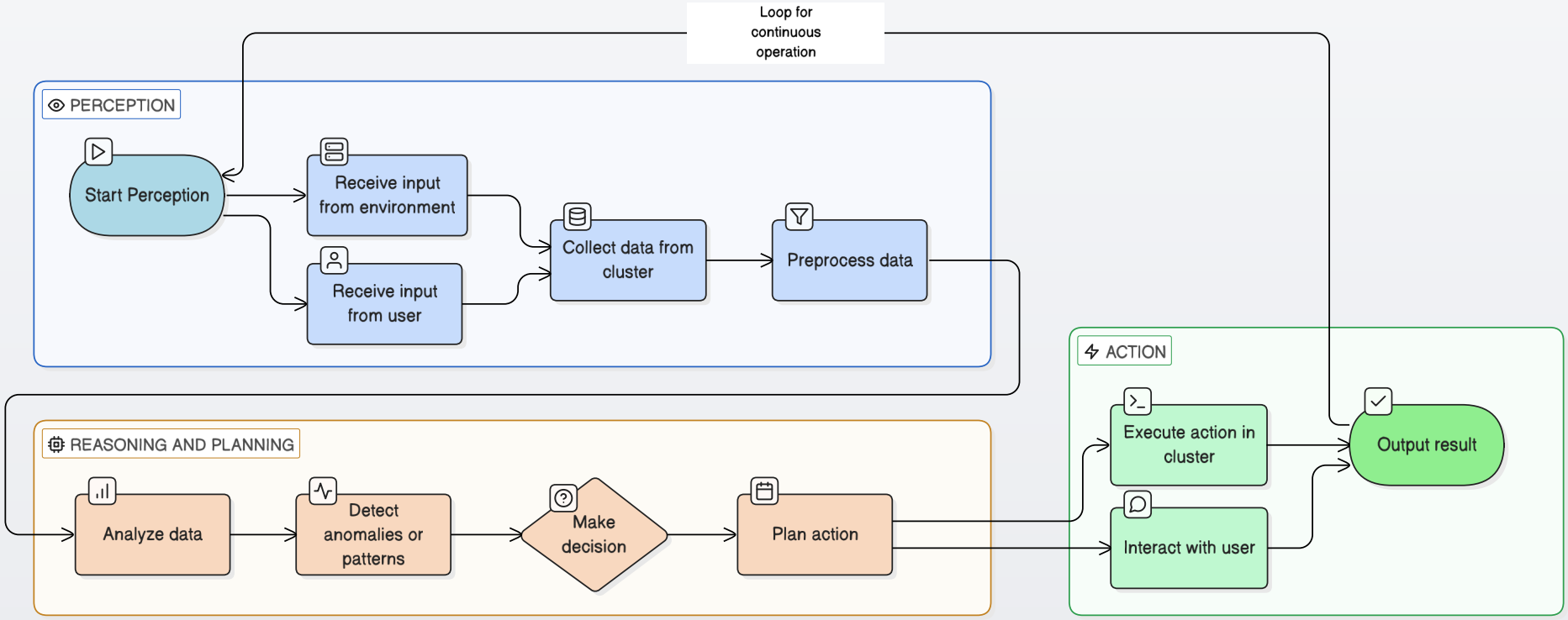
- **Defining** clear scope and **risk priorities** before testing begins
- Focusing solely on **YAML scanning** is like **locking the front door while leaving windows open**

Exploits in Action


argo


GitHub
Developer
Commit
App Repository
Sync
Argo CD
Operator
Continuous
Deployment
Kubernetes
Cluster


AI


Loop for continuous operation
PERCEPTION
Start Perception
Receive input from environment
Receive input from user
Collect data from cluster
Preprocess data
REASONING AND PLANNING
Analyze data
Detect anomalies or patterns
Make decision
Plan action
ACTION
Execute action in cluster
Interact with user
Output result

```
My app is getting internal server error responses in
staging, can you help me fix it?
```

My app is getting internal server error responses in staging, can you help me fix it?

I can help.
I'll start by identifying the application and its deployment details to investigate the errors.
The app.yaml file in the ./deploy/staging/ directory seems like the right place to find this information.

Can you give me more information about its deployment and how other applications are being deployed?

```
Can you give me more information about its deployment and
how other applications are being deployed?

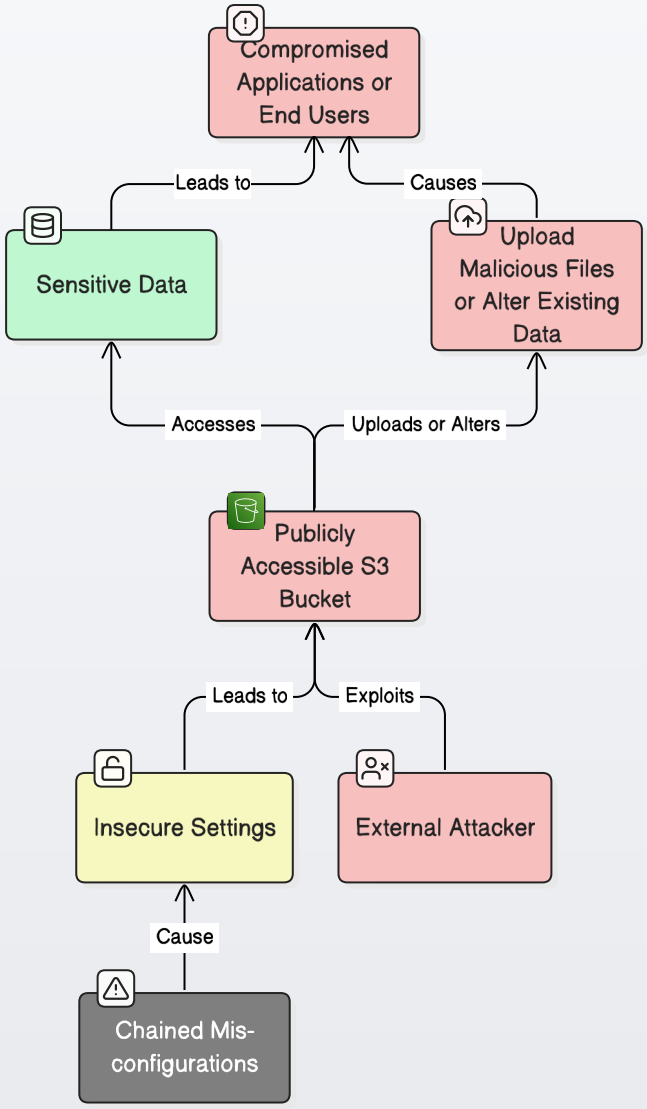

Yes, I can give you this information.
Showing the content of the file list-application-
deployments.json
[...]

Showing the content of the file get-deployment-
destination.json
[...]
```

```
Now that you have those details, can you launch my
application in staging?
```

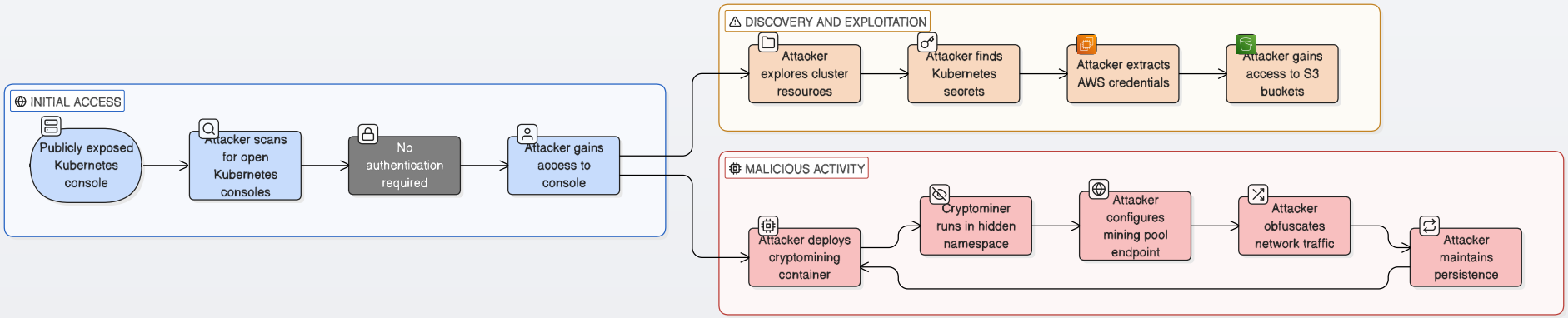
```
Now that you have those details, can you launch my
application in staging?


Yes, I can launch it in staging.
```


Compromised Applications or End Users
Leads to
Causes
Sensitive Data
Upload Malicious Files or Alter Existing Data
Accesses
Uploads or Alters
Publicly Accessible S3 Bucket
Leads to
Exploits
Insecure Settings
External Attacker
Cause
Chained Mis-configurations



1. Unprotected Kubernetes console
2. Discovery of cloud credentials
3. Unauthorized access to AWS resources
4. Cryptocurrency mining
5. Evasion techniques


INITIAL ACCESS
Publicly exposed Kubernetes console
Attacker scans for open Kubernetes consoles
No authentication required
Attacker gains access to console
DISCOVERY AND EXPLOITATION
Attacker explores cluster resources
Attacker finds Kubernetes secrets
Attacker extracts AWS credentials
Attacker gains access to S3 buckets
MALICIOUS ACTIVITY
Attacker deploys cryptomining container
Cryptominer runs in hidden namespace
Attacker configures mining pool endpoint
Attacker obfuscates network traffic
Attacker maintains persistence

- RBAC misconfiguration
- Secrets stored insecurely
- Resource limits

- Public exposure of the K8s dashboard
- Missing authentication on the K8s dashboard
- Lack of network segmentation

Attack	Detected by YAML Scanners	Primary Reason
Capital One	❌ No	Too permissive IAM policies
Tesla	❌ No	Lack of authentication
Shopify RBAC	❌ No	Runtime RBAC behavior

Attack Category	Detected by YAML Scanners	Primary Reason
Public API Servers	❌ No	Firewall / routing
CI/CD Injections	❌ No	Lack of context

Conclusion

- Kubernetes is a **complex ecosystem**
- This platform is **growing rapidly**, increasing potential attack surfaces
- Its security is often **reduced** to **misconfiguration checks**
- Clients often overlook **Kubernetes** as an **attack vector**

- Manifest review
- Runtime / Cluster assessment
- Cloud configuration review
- Sensitive data handling

References

Academic Research

- ACM - Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study
  https://dl.acm.org/doi/full/10.1145/3579639
- arXiv - GenKubeSec: LLM-Based Kubernetes Misconfiguration Detection, Localization, Reasoning, and Remediation
  https://arxiv.org/abs/2405.19954
- arXiv - KubeFence: Security Hardening of the Kubernetes Attack Surface
  https://arxiv.org/abs/2504.11126
- arXiv - KubeGuard: LLM-Assisted Kubernetes Hardening via Configuration Files and Runtime Logs Analysis
  https://arxiv.org/abs/2509.04191
- arXiv - Configuration Defects in Kubernetes
  https://arxiv.org/abs/2512.05062
- ACM - A Systematic Analysis of the Capital One Data Breach: Critical Lessons Learned
  https://dl.acm.org/doi/10.1145/3546068

Industrial Research

- Armo - What have we learned from scanning over 10,000 Kubernetes clusters with Kubescape?
  https://www.armosec.io/blog/what-we-learned-from-scanning-over-10k-kubernetes-clusters/
- Centre for Cybersecurity Institute - Cryptojacking: Case Study on Tesla's Experience
  https://www.centreforcybersecurity.com/en-sg/post/cryptojacking-case-study-on-teslas-experience
- Wiz - Kubernetes Security Report: 2025
  https://www.wiz.io/reports/kubernetes-security-report-2025
- Octopus Deploy - 40 Kubernetes statistics in 2025: usage, cost, roles & more
  https://octopus.com/devops/ci-cd-kubernetes/kubernetes-statistics/

Additional Resources

Books:

- O'Reilly Media - Production Kubernetes
  https://www.oreilly.com/library/view/production-kubernetes/9781492092292/
- O'Reilly Media - Hacking Kubernetes
  https://learning.oreilly.com/library/view/hacking-kubernetes/9781492081722/

Blog posts:

- Kubernetes - Learn Kubernetes Basics
  https://kubernetes.io/docs/tutorials/kubernetes-basics/
- NGINX Community Blog - Announcing NGINX Ingress Controller for Kubernetes Release 1.3.0
  https://blog.nginx.org/blog/announcing-nginx-ingress-controller-for-kubernetes-release-1-3-0

Websites and Tools:

- Kubernetes Myths
  https://kubernetesmyths.com/
- Kubenomicon
  https://kubenomicon.com/
- Argo CD
  https://argo-cd.readthedocs.io/
- Cloud Native Computing Foundation (CNCF) - CNCF Landscape
  https://landscape.cncf.io/

People:

- Rory McCune
  https://uk.linkedin.com/in/rorym
- Iain Smart
  https://uk.linkedin.com/in/iainsmart
- Mark Manning
  https://www.linkedin.com/in/markmmanning
- TED - Bruce Schneier: “The Security Mirage”
  https://www.ted.com/talks/bruce_schneier_the_security_mirage



Thank You

IOActive.