Fachhochschule Kiel

Faculty of Computer Science and Electrical Engineering

Information Engineering

Master Thesis

Automatic Assessment of Applications Security Aspects running in Cloud Environments

written by:	**Jannik Hollenbach**
First Examiner:	Prof. Dr.-Ing. Meiko Jensen
Second Examiner:	Rüdiger Heins, M.Sc.
Date:	August 2, 2020

Abstract

One challenge security teams face when adopting cloud-based systems is the ability to ensure that all deployed applications follow security guidelines. This work suggests to use the API of the cloud provider to fetch all services running inside the cloud environment of a company. This inventory of services can then be used to automate a set of security scans for the automatically discovered services, including Dynamic Application Security Scans (DAST) for web applications and Static Application Security Scans (SAST) for deployed artifacts such as container images. This approach is demonstrated with a prototype developed to automatically discover and scan containerized applications inside Kubernetes clusters. In an evaluation of the prototype, it was shown that it is possible to use this approach to automatically discover some vulnerabilities with this approach.

Contents

1 Introduction

1.1 Motivation

In recent years, the development process of many software teams has changed drastically. The introduction of new tools and methodologies like agile development[1], DevOps[11] and cloud computing[4] have enabled many software teams to develop and deploy their applications more quickly and more independently.

While this is great for the general productivity of an organization, the increased speed and autonomy can be a daunting perspective for traditional security teams. The security teams have to keep up with the increased speed of the development teams while still ensuring the security of the individual applications and the company's general infrastructure. This is especially hard for security professionals as they are usually heavily outnumbered by developers in their organizations, often coming down to a single security engineer for a hundred developers.[11] [11, 137, 20]

One approach to address these issues are extensions to the DevOps methodology, often referred to as *DevSecOps*. While DevOps is focussed on breaking the silos between development and operation, DevSecOps tries to the same, while also including security in the process. DevSecOps makes security a part of the process, not just an afterthought. While most changes required are organizational, DevSecOps often also includes technical processes to automate certain aspects of the security assessment of the developed software systems. This includes automatic security assessments in the build pipelines of software projects and also the automatic assessment of software systems during their runtime. [11, 137, 20]

One aspect required to perform automatic assessment is to have access to a reliable list of all systems operated by the organizations. This is especially relevant for cloud-based systems where it is straightforward for an autonomous team or even an individual in the

team to spin up a new service, with little to no oversight. As each new service introduces new potential vulnerabilities this can be cause for security professionals to be skeptical about the adoption of cloud systems in an organization. [11, 3, 5]

1.2 Research Plan

This work will explore the possibilities to use the API provided by a cloud environment to automatically generate an inventory of all security relevant resources in the environment. To limit the scope of the work, the work will focus on the Kubernetes container orchestration system as an example of a cloud-based environment. As Kubernetes is primarily focused on running web/HTTP based applications, the focus of the security assessments in this work is on HTTP based applications. The security of the automatically discovered services/resources will then be assessed by different security scanning tools to ensure that these resources are deployed according to best practices and don't contain any obvious vulnerabilities.

Research Question: How can the information from the Kubernetes API be used to automatically assess security aspects of applications running inside a Kubernetes cluster?

To highlight how the learnings from such a Kubernetes based system can be applied to the broader landscape of cloud environments, Kubernetes will be introduced and categorized according to the definition of cloud computing by the National Institute of Standards and Technology (NIST). The different components which make up Kubernetes clusters are introduced to investigate how these components work together and explain which efforts have to be taken to secure Kubernetes as an underlying cloud environment.

Additionally, to protections against regular application threats like SQL Injection (SQLi) and Cross Site Scripting (XSS) attacks, Kubernetes clusters also need to be secured to ensure that applications can't be compromised from within the cluster. These aspects are highlighted by a literature review in chapter 3 and are later used to determine the scanners used to assess application security aspects.

To explore the possibilities, an *auto-discovery* prototype is built, to implement the

concepts described above. The architecture of the prototype is oriented upon the design of Kubernetes internal components, namely Controllers, this allows the prototype to integrate deeply into Kubernetes and gives it full access to the API. This prototype is then used in a demo Kubernetes cluster to demonstrate the possibilities of an auto-discovery system for assessing the security aspects of cloud-based applications.

1.3 Prior Art

Prior art in the exact field is sparse and often only addressed by the industry, not by academic research. As such this work is built upon prior art and research of multiple different fields to address the research question:

- Classification of Kubernetes into the cloud landscape by using both literature describing Kubernetes and the most commonly used definition of cloud computing in literature in chapter 2.

- Investigation into Kubernetes components and best practices to extend the Kubernetes behaviors to research possibilities of addressing the research question.chapter 2

- Security aspects of applications on Kubernetes to research which areas of aspects are relevant to the security of applications on Kubernetes in chapter 3.

- Research into similar projects trying to achieve security assessments automatically for cloud-based applications and Kubernetes in particular in chapter 4.

2 Kubernetes as a Cloud Environment

In this chapter, the Kubernetes container orchestration system is introduced. To classify how Kubernetes fits into the broader landscape of cloud environments, the most common definition of cloud computing is used. This classification is later used to see how much the results of this work can be applied to other types of cloud environments.

2.1 What is a Cloud Environment?

The most commonly used definition for cloud computing[25] stems from the National Institute of Standards and Technologies Special Publication 800-145 (NIST SP 800-145)[4]. In this definition, cloud computing is described by five essential characteristics, three service models, and four deployment models.

The essential characteristics of the NIST SP 800-145 consist of five properties shared by cloud environments. They are useful to understand whether a system fits into the cloud computing landscape or if the technology serves a different purpose.

The service models are a useful tool to sort the different services provided by a cloud provider or a specific open source project into one of these categories. This lets users better understand which responsibilities are handled by the service and which have to be handled by themself. This separation of security responsibilities will be further discussed in section 2.1.1.

The third part of the NIST SP 800-145 is the classification of cloud environments into four deployment models:

- **Public Cloud:** Public clouds are shared by multiple organizations and available to

everybody.

- **Private Cloud:** Private clouds are exclusively used by one organization.
- **Community Cloud:** Community clouds are shared by multiple organizations but are, in contrast to public clouds, not available to the general public.
- **Hybrid Cloud:** Applies when two or more of the other deployment models are combined.

This work will focus mainly on Kubernetes as an example of a cloud computing environment. How Kubernetes fits into the NIST SP 800-145 definition is discussed in section 2.2.3.

2.1.1 Distribution of Security Responsibilities

In cloud applications the responsibility to secure different layers is distributed between two parties, the Cloud Service Provider (CSP), and the consumer. When using a public cloud the CSP is likely one of the big players in the fields like Amazon Web Services (AWS)[52], Microsoft's Azure[141] or Google Cloud Plattform (GCP)[98]. The CSP provides the customer with a large number of different services. These services can be anything from providing access to a virtual machine to run generic software on top of, to extremely specific services catered to only a small section of an industry, like AWS Groundstation, a tool which lets you control a fleet of satellites.[56] Using the NIST SP 800-145 definition of cloud computing, these services can be broken down into the three service models:

- **Infrastructure as a Service (IaaS):** Provisioning of generalized compute resources. E.g. Virtual Machine (VM)
- **Platform as a Service (PaaS):** Provisioning of specialized compute resources, where the underlying infrastructure is managed by the CSP. E.g. managed application runtimes like AWS Lambda[58], Azure Functions[61], or GCP's Cloud Functions [97]
- **Software as a Service (SaaS):** Provisioning of a specific service to users directly.

Clients only have access to the service using a web or native client to access or modify their data. E.g. productivity applications like Microsoft's Office 365[142] or Google G Suite[101]

The service models also indicates which security responsibilities are handled by the CSP and which belong to the user of the cloud service.[25, 5] See fig. **2**-1.

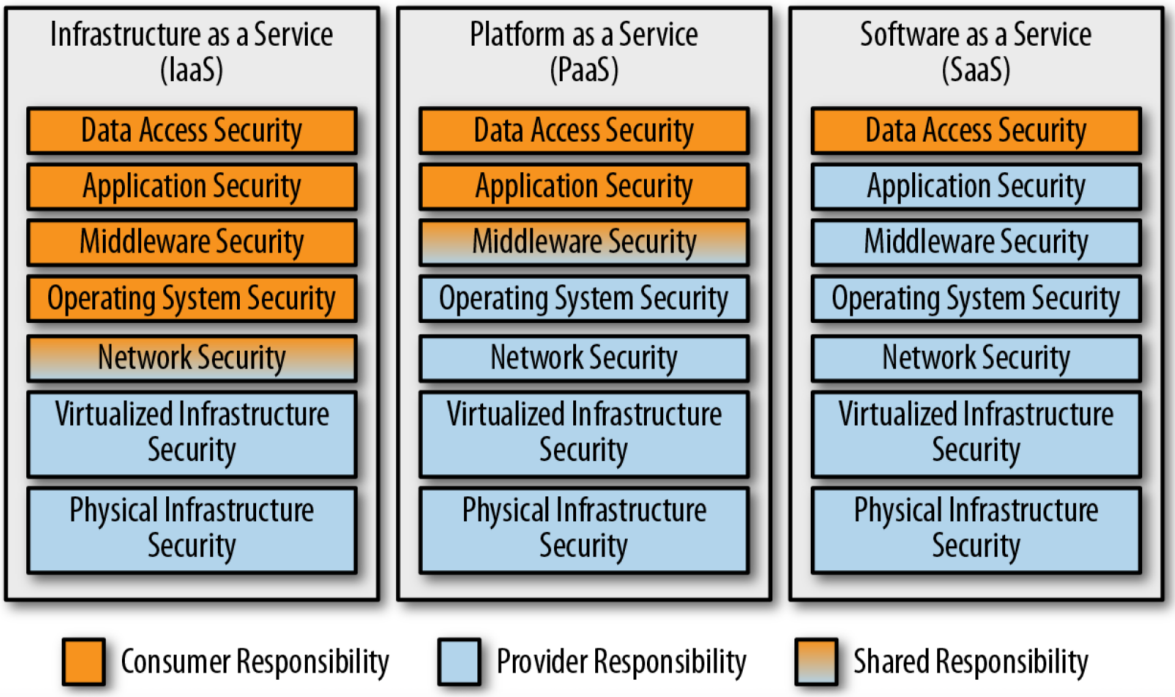

Figure **2**-1: General overview of the distribution of security responsibilities between cloud provider and consumer. From [25]

As a consumer, it is important to be aware which layers are in your responsibility and which are handled by the cloud provider. Whether the consumer or the cloud provider is responsible for a certain security measure differs between the different service offerings and cloud providers. As a consumer of a cloud service, it is important to be always aware which areas of security are your responsibility. [25]

As long as the cloud user is aware, which areas they are responsible for securing, this separation of responsibilities can have a positive impact on the system's security stance. CSP's can often perform better at securing the underlying security fields as they operate on a large scale. This difference is apparent in most layers, but easiest to visualize in the physical infrastructure security layer. Large cloud providers often have a multi-layered perimeter security system with biometrical access controls for their data centers.[49, 100] Such security measures are often too expensive for smaller companies

to implement them on their own. When these companies migrate their applications to a cloud provider, they can run their systems with the same high physical access security as every other system in the cloud provider.[7]

2.2 What is Kubernetes?

2.2.1 History of Kubernetes

Kubernetes is a container orchestration system originally developed by Google and was released as an open-source project in 2014. Kubernetes built upon learnings from Googles internal container orchestration systems Borg and Omega, which are used internally to schedule hundreds of thousands of containerized workloads on tens of thousands of machines.[9]- [10]

In 2015 the project was transferred from a primary leadership from Google to the Cloud Native Computing Foundation (CNCF)[74], a newly formed subsidiary organization of the Linux Foundation.[8] In 2018 Kubernetes was the first project inside the CNCF organization to be promoted to the graduated level. This level is supposed to indicate that Kubernetes is considered stable and mature enough to be a viable option across all industries.[17]

Since its public release, Kubernetes has been adopted by a large number of companies. The adoption of an open-source project is hard to quantify in actual numbers, as there is no central entity able to track which companies have adopted the project. Some indication of the adoption is the large number of companies which have joined the CNCF organization, which is largely defined by the Kubernetes project and the ecosystem surrounding it.[78]
Another indication of the project's success is the large number of public cloud vendors providing managed Kubernetes offerings, in which most internal cluster components are handled by the cloud provider. A overview of the hosted offerings can be found on the CNCF landscape[77] and includes all of the three big public cloud providers AWS with Amazon Elastic Kubernetes Service (EKS)[51], Azure with Azure Kubernetes Service (AKS)[63] and GCP with Google Kubernetes Engine (GKE)[102].

2.2.2 How Kubernetes is Used

Kubernetes allows users to deploy and manage containerized workloads to a cluster of machines. Containers are, compared to virtual machines, a more light weight way to package and run an isolated workload. This is achieved by using features of the Linux kernel to isolated processes from each other in their own namespace.[1]

One of the core principles of Kubernetes is its declarative API. Users don't tell Kubernetes what to do, but how they want the cluster to look like. Kubernetes will then compare this declaration with the current state of the cluster and make the necessary changes to the cluster to meet the definition of the user. How this principle is realized in Kubernetes will be further discussed in section 2.3. [23, Chapter 1]

These declarations in Kubernetes are called *Resources*. For the remainder of this work the capitalized spelling of Resources will refer to Kubernetes Resources rather than the general term. Specific Kubernetes Resources mentioned in the remainder of this work will also be capitalized.

Kubernetes offers several different Resources for users to define different aspects of the deployment of their applications. The following paragraphs will describe three types of Resources required to deploy and expose an application:

Deployments in Kubernetes are a way to deploy stateless applications. Stateless means that the applications don't require persistent write access to a disk, like a database would need to.[23] An example of a Kubernetes Deployment described in YAML can be found in listing 2.1. This example is a Deployment for the Open Web Application Security Project (OWASP) Juice Shop[149] application, which will be later used in this work as an evaluation case for the automatic security assessment. This manifest tells Kubernetes that three replicas of the application should be run at all times. For each of these replicas, Kubernetes will start a separate *Pod*. A Pod is a collection of one or more containers that make up the application. In this example the Pod will only contain one image, which is using the official container image of the Juice Shop application[85].

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

[1] Windows containers also exists, which behave differently and are not directly compatible with Linux based containers.[89] The term containers when used in this work will refer to Linux based containers.

```
  name: owasp-juice-shop
  namespace: juice-shop
spec:
  selector:
    matchLabels:
      app: juice-shop
  replicas: 3
  template:
    metadata:
      labels:
        app: juice-shop
    spec:
      containers:
        - name: juice-shop
          image: docker.io/bkimminich/juice-shop:v11.1.2
          ports:
            - name: http
              containerPort: 3000
```

Listing 2.1: Kubernetes Deployment of the OWASP Juice Shop Application

Services let users address all replicas of the application under a single load-balanced IP address. Most clusters also allow applications to use DNS to lookup the IP addresses of Services in the Cluster which enables automatic service discovery in the clusters.

Ingress Resources let users define how incoming HTTP traffic should be handled. This lets users define rules like all HTTP traffic for `juice-shop.example.com` should be routed to the Juice Shop Service.

2.2.3 Classification of Kubernetes in the Cloud Landscape

This section will discuss how Kubernetes fits into the NIST SP 800-145 definition of cloud computing[4]. As Kubernetes is used in this work, as an example of a cloud computing environment, it is important to highlight which areas of cloud computing are covered by this sole focus on Kubernetes and which are left out.

Essential Characteristics

Kubernetes meets or can meet all of the five essential characteristics of the NIST SP 800-145. Whether or not a characteristic is met can depend both on how the cluster is deployed and how the cluster is used. Listed below is a short overview of how these characteristics apply to Kubernetes.

- **On-demand self-service:** Kubernetes enables users manage their own services by themselves.[23, Chapter 1]

- **Broad network access:** Kubernetes and its API is available to all authenticated users over the network. See API Server in section 2.3

- **Resource pooling:** The compute resources in the cluster are shared by all users in the cluster. The Scheduler will automatically assign new workloads (Pods) to an available node. See *Scheduler* in section 2.3.

- **Rapid elasticity:** Services running on Kubernetes can be scaled to meet demand.[122] By default this scaling is restricted by the total size of the cluster. When using a managed Kubernetes service or running the Kubernetes cluster on an IaaS service it is also possible to scale the cluster size automatically based on its utilization.[118]

- **Measured service:** The resource usage of a specific tenant (called namespace) can be tracked via the *metrics-server*[136] and restricted via the *ResourceQuotas* Resource in Kubernetes.[131]

Deployment Models

Kubernetes is extremely flexible when it comes to where it can run. This lets Kubernetes run in all four of the defined deployment models of the NIST SP 800-145. This flexibility of Kubernetes is often used by public cloud providers to extend their managed Kubernetes offerings to run in on-premise environments, essentially providing a hybrid cloud as a service[62, 50, 96].

Service Models

When it comes to the classification of Kubernetes into one of the service models of the NIST SP 800-145 definition of cloud computing, Kubernetes can be best described using the PaaS service model. Not all aspects of the PaaS definition fit perfectly thought.[133] Kubernetes is in a lot of aspects a lower-level abstraction than the most common PaaS platforms like Heroku[108], Vercel[169], Google App Engine[54] or similar services. One

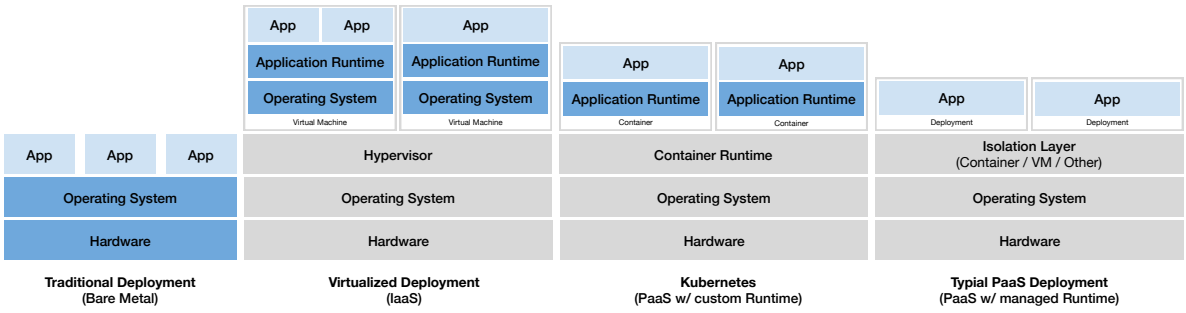

Figure **2**-2: Overview of virtualization layers in different cloud systems. Blue backgrounds indicate layers which are usually in the responsibility of the user. Based on [133]

of the aspects which show this lower level is that PaaS services are usually restrictive in which programming languages can be used for applications running on top of them, as the language runtime is managed by the PaaS platform. Kubernetes doesn't have this restriction, as the deployment target for Kubernetes is containers. This allows Kubernetes to run a wide variety of applications, from decade-old *Fortran* code to code written in modern programming languages.[109] This is possible because the container image contains all dependencies, including the language runtime and system libraries inside the image. A visual comparison of different software stacks between IaaS and different PaaS stacks can be found in fig. **2**-2.

The Kubernetes ecosystem contains multiple projects to enable a more PaaS like experience on top of Kubernetes, including managed application runtimes. Some of these tools are:

- **Cloud Foundry** which aims to provide a PaaS like platform running on Kubernetes.[72]

- **Cloud Native Buildpack** which takes out the application runtime management out of the container image build process[73]

- **OpenFaaS** provides a Function as a Service (FaaS) / serverless platform like AWS Lambda[58], Azure Functions[61] or Google Cloud Function[97] on top of Kubernetes.[146]

Another Aspect in which Kubernetes differs from other PaaS offerings is, that Kubernetes provides several escape-hatches to interact with the underlying nodes directly

One of these escape hatches is the ability to run `privileged` containers, these are containers that have nearly the same capabilities as the host's OS, thus breaking the container isolation. This can be used to configure the host's OS to better match the needs of the applications running on them. Privileged containers can also pose a big security risk in clusters, the use of them can be prohibited. The security aspect of Kubernetes will be further discussed in chapter 3. [42, 21]

Another one of these types of utilities are DaemonSets, which lets users run one Pods (container) on every node of the cluster. These are often used to provide cluster level utilities like the collection of log, metric, or tracing data from applications running inside the cluster. [23]

2.3 Anatomy of a Kubernetes Cluster

Kubernetes clusters are comprised of one or multiple nodes. The nodes are machines on which the applications of the cluster are run. Additionally, to the actual application for which the cluster is intended for, Kubernetes also needs to run a number of its own components that are required to keep the cluster running and provide additional services to the applications.

The Kubernetes components come in two classes:

- **Node Components:** Components required on every node.[16, 124]

- **Control Plane Components:** Components that are required at least once per cluster. If Kubernetes is provided by a cloud provider as a service these components are often fully managed by the cloud provider and run outside of the cluster and thus inaccessible to the user. When self-hosting a Kubernetes cluster, it is gen-

erally recommended to run these on dedicated nodes, so that their performance cannot be degraded by user-controlled workloads. [24, 16, 124]

The section below gives a brief overview of all required components in a cluster which are required for its operation. As these components can be found in every cluster (with some exceptions), these components are also interesting from a security standpoint as a security vulnerability in these components will affect all Kubernetes clusters. A visual overview of the components can be found in fig. **2**-3.

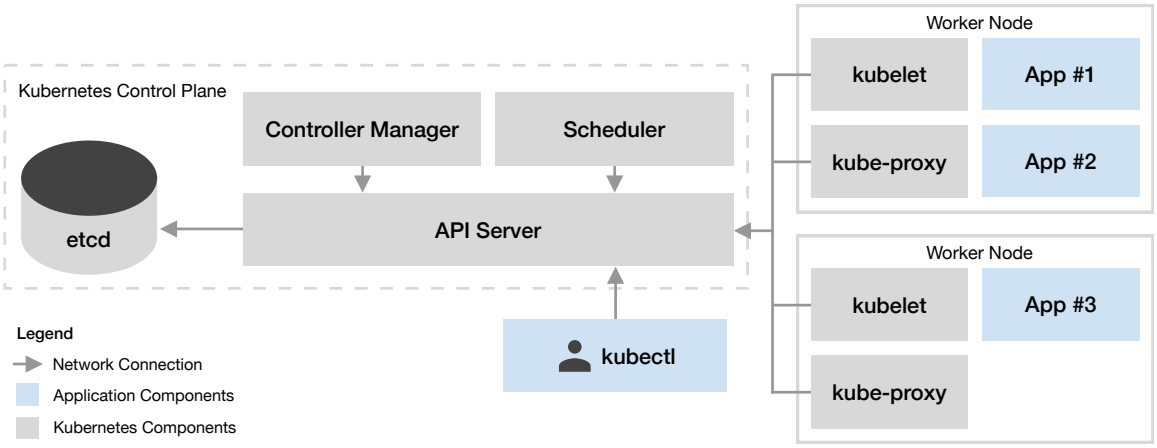

Figure **2**-3: Overview of components in a Kubernetes cluster. Based on [26, 124]

etcd *(Control Plane Component)*

etcd is an open source key-value database originally created by CoreOS, which like Kubernetes is now managed by the CNCF. It is used to store all Resources (see section 2.2.2) in the Kubernetes cluster.[16]

etcd can be replaced by other databases. E.g. the lightweight Kubernetes (re) distribution k3s uses *SQLite* as a replacement for etcd.[111] This work will assume that clusters use etcd as this is the default in Kubernetes.

API Server *(Control Plane Component)*

The *API Server* is the component in Kubernetes which users (developers and operators) and also Continious Integration (CI) systems deploying applications interact with directly. It provides an HTTP API to interact with the Resources stored in etcd. It is the only component in Kubernetes which has direct access to etcd. This makes the API Server

a critical component, as all other components in Kubernetes which need access to the Resources saved in etcd need to communicate with the API Server. [16]

The API Server is also responsible to handle authentication and authorization if these are enabled in the cluster configuration.[16]

Scheduler *(Control Plane Component)*

The Scheduler is responsible to assign newly created Pods to specific worker nodes. Pods in Kubernetes can have a number of restrictions that indicate which types of nodes are unsuitable for them (e.g. the Pod can only run on nodes with a `arm` CPU architecture). Pods can also have optional requirements that the scheduler tries to enforce as much as possible. [16]

Controller Manager *(Control Plane Component)*

The *Controller Manager* is a set of *Controllers* compiled together into one service / one binary. Each of these Controllers are running a reconciliation/control loop which compares the specification of a Resource from the Kubernetes API Server with the state of the cluster and/or the external world and updates them if they don't match the specification. A diagram of the reconciliation loop can be found in fig. **2**-4. [26, 16]

An example of a Controller is the Deployment Controller which manages Deployment Resources described in section 2.2.2. The Deployment Controller is responsible for starting enough Pods to match the number of replicas specified in the Deployment specification.

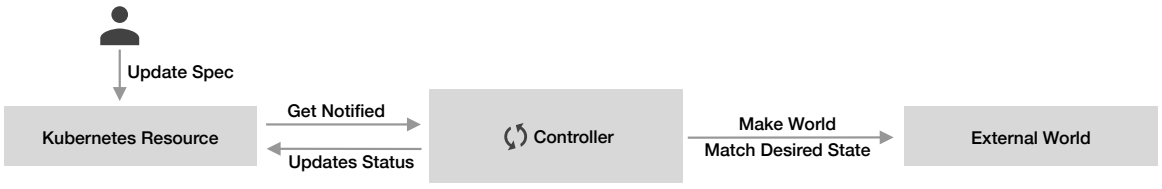

Figure **2**-4: Overview of the Kubernetes Reconciliation / Control Loop. Based on [26]

Most Kubernetes clusters also contain other Controllers which are not directly shipped with Kubernetes. Most CSPs include a *Cloud Controller Manager* which reconciles Kubernetes Resource which directly correspond to objects in the cloud provider like *LoadBalancers* or *PersistentVolumes*. The Cloud Controller Manager will then call

the API of the CSP to create or update these cluster external resources to match the specification of the Kubernetes Resource. [119, 23]

kubelet *(Node Component)*

The *kubelet* runs on all Kubernetes nodes and is responsible to run and manage the Pods assigned to its Node by the Scheduler.

The kubelet itself delegates the execution of the container to a Container Runtime Interface (CRI), which is then responsible for managing the underlying system calls to create the container. This lets users switch out their container runtime without having to recompile the kubelet. The most popular options are *Docker*[84] and *containerd*[80].[29] This separation has also enabled further experimentation to provide a stronger isolation between the containers using tools like *Firecracker*[93], *gVisor*[104] and *KataContainers*[112].[92, 91] The security impact of these additional isolation layers is highlighted in chapter 3.

Part of the managing responsibility is to execute the health-checks defined for the Pod, to check if the Pod is still working as intended. Kubernetes offers different options to check the health of a Pod, e.g. by sending an HTTP `GET` request to a specified endpoint and checking the status code of the HTTP response. [16, 23]

The kubelet is also responsible to configure the Container Networking Interface (CNI), which ensures that Pods can send packages to other Pods, even when they are physically located on different nodes.[2] Similar to the CRI the CNI is pluggable, which lets users swap out a CNI implantation for a different one. One common CNI implementation is Calico[152]. [127]

kube-proxy *(Node Component)*

The kube-proxy is responsible to manage Service Resources in Kubernetes which let user address multiple Pods under one address. By default kube-proxy works by setting rules for *iptables* on the host operating system. [16]

kube-proxy is often run as a privileged container inside a DaemonSet, see section 2.2.3, in the cluster to ensure that is running on every node.

[2] The network access of a Pod can be restricted using NetworkPolicies[128], to prevent unwanted communication between Pods. See chapter 3.

Optional Services

Most clusters also include additional services to full-fill certain additional functions, like CoreDNS[81] to provide service discoverability and the metrics-server[136] to monitor and enforce compute resource usage. These two services are included in most Kubernetes distributions, but not necessary to run the cluster.

3 Security in Kubernetes Applications

This chapter will highlight vulnerabilities that directly influence the security of applications running on Kubernetes. The vulnerabilities listed in this chapter were aggregated from multiple types of sources, including books, project documentation, conference talks, and blog posts. As some of these sources are lacking in their use of scientific methods, multiple sources were consulted at all times to ensure a reliable source of information.

The vulnerabilities listed in this chapter will then be later used to analyze how their detection can be automatically assessed in chapter 4.

3.1 Security Aspects of Kubernetes Components

Kubernetes itself has to be kept secure to ensure that the applications running on top of it are also secure. This section described a set of security aspects of the Kubernetes internal cluster components described in section 2.3. The security aspects of these components are critical as they are part of every Kubernetes cluster and a single compromise of one of the components often leads to a complete compromise of the entire cluster. A high-level overview of the attack points can be found in fig. **3**-1.

This work assumes that the Kubernetes cluster is deployed in a PaaS like manner where both the physical security, virtualized infrastructure security, network security, and operating system security are handled by the CSP and are therefore out of scope for this work. The shared responsibility middleware, see fig. **2**-1, in this case, is Kubernetes and its Components. This shared responsibility introduces a problem as the exact responsibility distribution differs between different CSPs. How the individual Kubernetes offerings of the CSPs have to be configured can be typically found in their documentation.[95, 161, 160] This chapter highlights some of the types of Kubernetes

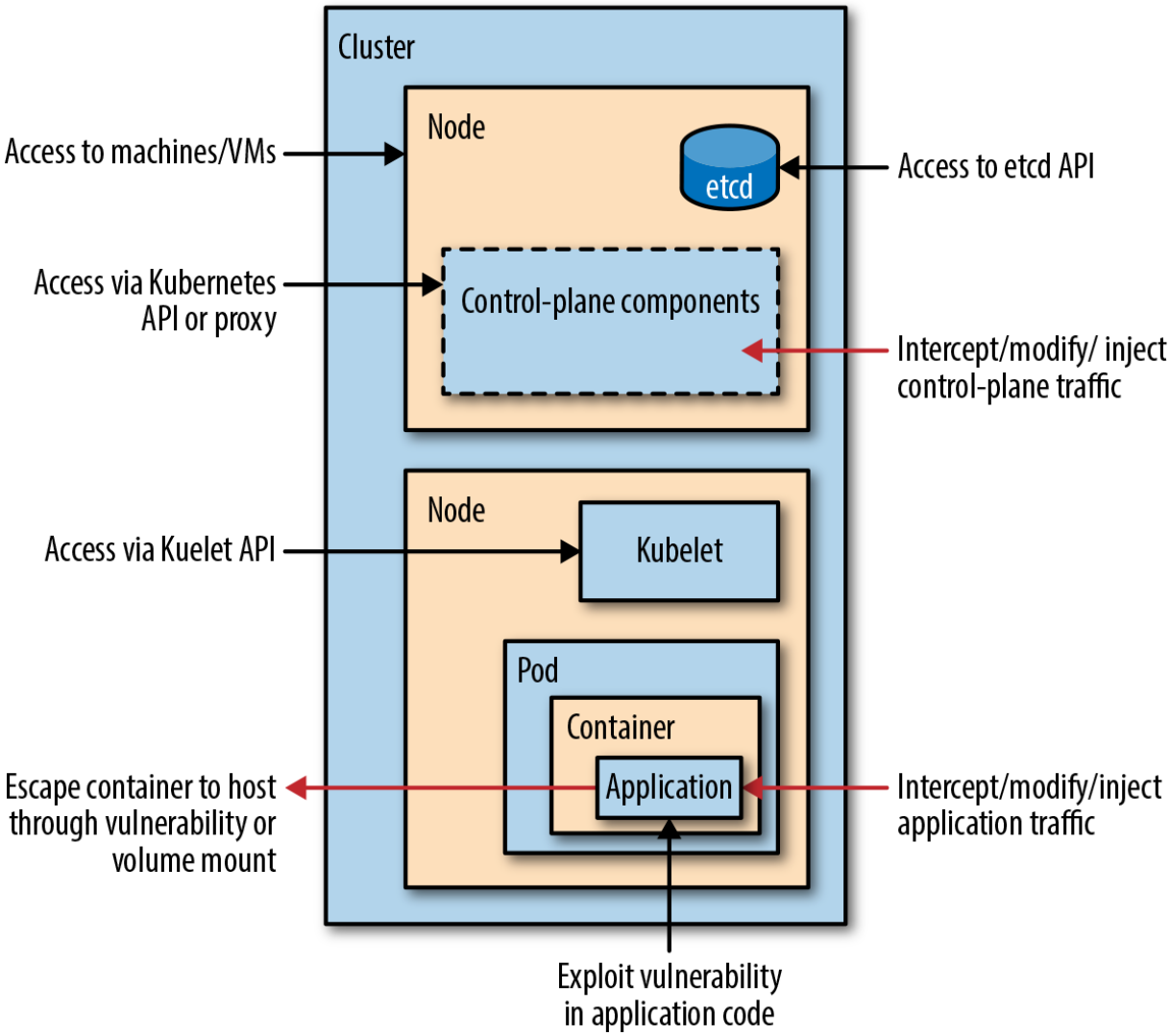

Figure **3**-1: General overview of Attack Vectors in Kubernetes clusters. From [21]

specific vulnerabilities that are often, at least partly, in the responsibility of the user, depending on the CSP.

Missing Authentication / Authorization

Multiple Kubernetes internal components expose an API which lets users or other components of the cluster communicate with them. The only components which should be accessed directly by a user / or tool are the API Server, see section 2.3, which uses a Role-based Access Control (RBAC) model to manage and restrict access to individual Resources. RBAC is turned on by default since Kubernetes Release 1.6.0, which was released in march 2017.[107, 13] Even though it is enabled by default it can be turned

off by the cluster administrators. Additionally to authorization (handled by RBAC) it should be ensured that the cluster uses a secure authentication strategy e.g. using *OpenID Connect* or client certificates.[21] [39, 16]

Outdated Clusters

Kubernetes regularly has vulnerabilities discovered in its codebase. The Kubernetes project has a publicly documented process[125] how disclosed vulnerabilities are handled and addressed quickly to push fixes for the disclosed vulnerabilities out as soon as possible. The fixes are generally released for the last three minor releases of Kubernetes.[126] Kubernetes uses a quarterly release schedule which means that minor releases are supported for nine months. Commercial providers might back-port security fixes to older Kubernetes versions in their Kubernetes offerings. [125]

A list of all reported and confirmed vulnerabilities in Kubernetes can be found in the Kubernetes issue tracker on GitHub.[135]

Exposed Development Tools

The Kubernetes ecosystem contains a large set of tools used by developers and operators to perform regular jobs like deploying and inspecting applications on top of Kubernetes. An overview of this ecosystem of tools and product can be found on the *Cloud Native Interactive Landscape* of the CNCF.[76] Depending on the tool and its configuration they can itself pose a threat to the safety of the cluster.

One commonly used tool in the Kubernetes Ecosystem is the *Kubernetes Dashboard*, which gives users a web user interface to view and manage their Resources in Kubernetes. It's generally recommended to not expose the Dashboard to the internet, but to access it via the proxy functionality of the Kubernetes command-line client *kubectl*.[21, 132]

There have been several cases where the Kubernetes Dashboard has been deployed incorrectly so that it was publicly available without any authentication. One infamous example of this was the cryptocurrency mining attack[59] against Tesla. Tesla had the Kubernetes Dashboard deployed in a publicly accessible way, without any authentication.

Through the Dashboard, attackers had gained access to the cluster and deployed cryptocurrency miners. The attackers were also able to breach a Tesla AWS account, as the cluster contained access keys for the AWS account. [18, 15, 19, 46]

3.2 Security Aspects of Application running inside Kubernetes Clusters

Aside from Kubernetes specific security flaws described in the previous section, Applications running on Kubernetes are also potentially vulnerable to more traditional attack vectors like SQLi and Remote Code Execution (RCE) vectors. These vectors are, compared to the Kubernetes specific vector better understood in the scientific community. A popular publication containing ten of the, in the eyes of the authors, the most important vulnerabilities for web/HTTP based applications is the *OWASP Top Ten 2017*[12]. The OWASP organization and their community publishes other guides/reports like the Web Security Testing Guide[36] which explains how these and other types of vulnerabilities can be tested. Or the Cheat Sheet Series, a series of guides on how different vulnerabilities can be mitigated.[33] As these vulnerabilities are better understood the section below will focus on the types of vulnerabilities that can cause additional harm to Kubernetes Applications, as they can potentially be escalated (by the privilege escalation vulnerabilities described in section 3.3) to compromise other applications or the entire cluster.

Remote Code Execution (RCE)

RCE describes a class of vulnerabilities that enable an attacker to run arbitrary code on a system. RCEs are often caused by other vulnerabilities these are listed below as the potential attack vectors. [12] RCEs are hard to prevent, as they can be caused by a wide variety of different underlying vulnerabilities. Kubernetes as a platform can do very little to prevent RCE vulnerabilities in the applications, but can limit the impact of individual RCEs by ensuring that a compromised container in the cluster does not lead to the entire cluster being compromised. How these types of privilege escalations can be prevented in Kubernetes will be discussed in section 3.3.[21, 46]

Components with Known Vulnerabilities

In contrast to other PaaS like environments containerized systems bring their own language runtime and other Operating System (OS) level dependencies. In most PaaS systems the security of the tools can be centrally enforced as the versions available are centrally managed for the cloud environment. Containers have more flexibility there, which also comes at a security risk as each container image has to be updated to apply patches for their language runtimes and other OS level dependencies.

One general characteristic of container which can help to mitigate this risk is that the software contained in a container image is generally immutable. Immutability in this context means that all containers started from the same container image are using the exact same software. It is possible to update the software in a running container, but this is generally considered a bad practice.[42] Immutability can be enforced using the `readOnlyRootFilesystem` property in Kubernetes.[120] This allows scanning tools to analyze the container images and provide reliable information about the security stance of the software components contained in the container. [42]

3.3 Privilege Escalation in Kubernetes Clusters

This section contains types of vulnerabilities and missing protections that enable attackers to extend their privileges. When an attacker compromises a Pod in a Kubernetes cluster, ideally the security boundaries are strong enough so that the attacker can not escalate the privileges to take over other application in the same namespace, node or cluster.[21, 46] A graphical overview of the different types of security boundaries in a Kubernetes cluster can be found in fig. **3**-2.

The vulnerabilities in this section are especially relevant when the cluster is used by multiple tenants. Who is sharing a cluster with whom should be a conscious decision when adopting Kubernetes, as the isolation layer isn't bulletproof and requires active configuration to get right.[24]

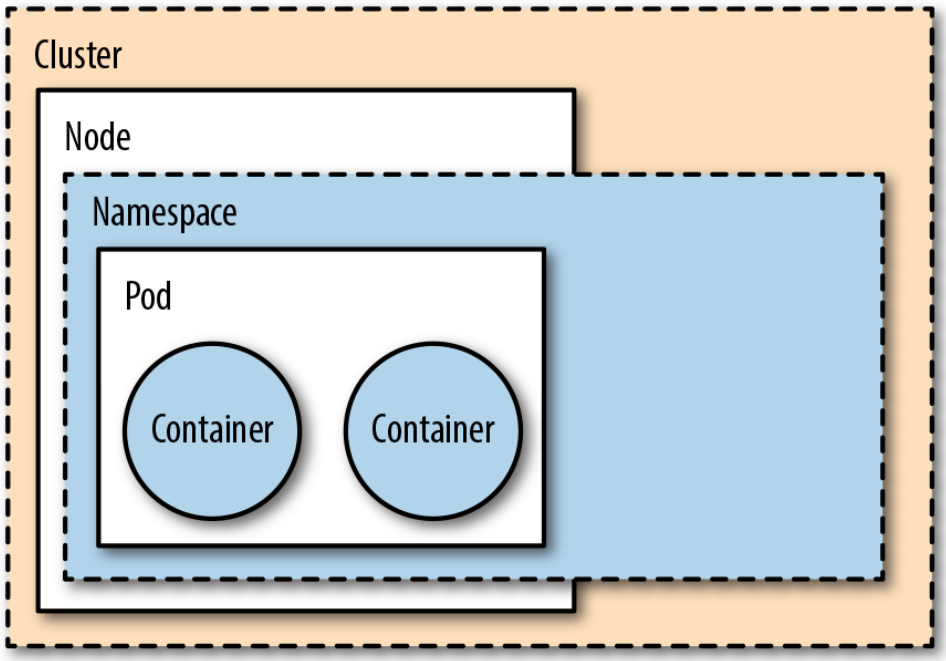

Figure **3**-2: Kubernetes Security Boundaries. From [21]

Container Isolation

One way to escalate the privileges of a containerized workload is to break the container isolation layer. These attacks have a similar impact as hypervisor breaches do in VM based infrastructure, only that container isolation is generally easier to break than the isolation of a VM.[42] The isolation is especially weak when the containerized processes posses a wide range of capabilities in Linux, as each of these poses as a new vector which can potentially be used to break the container isolation.[156, 42, 35, 39]

Two main miss-configurations can lead to these over granted Linux capabilities:

- Containers started using the `privileged` flag, which gives the container all Linux capabilities.[156, 42, 35]
- Containers using a `root` user (default when using Docker as the CRI)[42]. `root` user have access to most Linux capabilities, but less than privileged containers.[156, 42, 35]

Besides over granted Linux privileges, other potential ways exist in which the container isolation can be broken, e.g. by using `HostPath` mounts. HostPath mounts can be used

to make directories from the host available from the host to the container. Depending on the mounted directory HostPath mounts can be used to break the container isolation. [21, 79]

Kubernetes allows cluster administrators to define rules and policies which forbid cluster users from using the above mentioned features by specifying *PodSecurityPolicies*.[129, 21, 35, 129]

Depending on the CRI used in the cluster, the container isolation can also be improved by adding an additional isolation layer between the host and the container. These tools either introduce a light-weight virtual machine (e.g. Firecracker[93] or katacontainers[112]) or a user-space reimplementation of kernel features(e.g. gVisor[104]). [21]

Similarly to reducing the number of Linux capabilities available to the container, it is considered to further minimize the attack surface for an attacker by minimizing the base image to only contain the tools required to run the application. Many commonly used base images contain several tools which are helpful for debugging, e.g. shells like *bash*. These tools are also helpful to attackers when they have compromised a container, e.g. via a RCE vulnerability. [42, 79] If the developers are using a language which can produce statically linked dependencies, e.g. C, C++, Go[167], the container images can be build using the `scratch` base image, completely empty base image. If the language requires a runtime or cannot statically compile, users can use a base image like the *Distroless*[47] base container images from Google, which contain the language runtime and the bare minimum of libraries required for them to run.[47]

Using Authenticated Service Accounts

Kubernetes uses a Resource called *Service Accounts* for the authentication of workloads against the Kubernetes API Server. As a utility Kubernetes can mount an access token for a Service Account into the Pod, so that the workload in the Pod can access the API Server with its Service Account. All namespaces in Kubernetes have a *default* Service Account, which is also literally called `default`. The Service Account token for this default Service Account gets automatically mounted into every container, unless the specification of the Pod has the `automountServiceAccountToken` property set to `false`, or the default Service Account has been modified to disable this behavior.[121]

In properly configured clusters and namespaces with RBAC enabled, this only has a limited security impact, as the default Service Account doesn't have any rights. In clusters without RBAC enabled, the Service Account could perform any operation, as in these clusters it is enough to be authenticated with no authorization checks being performed.

Another case where this can be problematic is when the default Service Account of the namespace has been given additional roles to have access to the Kubernetes API. This can happen when the Service Account is used by applications and the administrators grant privileges to the default Service Account instead of creating a new one for the application.[21, 39]

4 Automatic Security Verification in Kubernetes Prototype

In this chapter the general goals for the prototype are described, which it needs to fulfill to be able to properly answer the research question of this work. The goals stated are compared to open-source projects in this field which aim to solve similar problems. Finally, the architecture of the prototype and its mode of operation are introduced, including the scanning tools used to assess the security of the individual Resources.

4.1 Prototype Goals

The Goal of the prototype is to automatically detect security aspects of applications running inside Kubernetes clusters by using the Kubernetes API to automatically discover all Resources which are relevant to the application's security. This concept of the automatic detection of security relevant Resources will be referred to as *auto-discovery* in the remainder of this work.

The actual assessment of the Resources is not handled by the prototype itself but delegated to specialized open-source security testing tools. The tools are orchestrated by the *OWASP secureCodeBox*[159], an open-source orchestration engine for security testing tools, see section 4.5. This enables the prototype to only handle the auto-discovery aspects and lets it delegate the assessment to the secureCodeBox and the integrated security testing tools.

The prototype will primarily focus on security aspects on web/HTTP based applications running on Kubernetes, which is arguably Kubernetes' primary focus. Kubernetes also supports non web/HTTP based applications by exposing them via NodePort or

LoadBalancer[1] Services[23], but these aren't as deeply integrated as HTTP based application via the Ingress Resource, see section 4.6.

Application specific security aspects, as described in section 3.2, will be covered both on their internal and their external attack surface. The term external attack surface is used in this work to describe any Resource (e.g. Ingress) which can be directly addressed by traffic from outside the cluster. The internal attack surface refers to Resources (e.g. ClusterIP Service) which can only be addressed by workloads from inside the cluster. The benefits of scanning both internal and external attack surface are further discussed in section 4.6.

The auto-discovery prototype aims to track the applications running on the Kubernetes cluster automatically and as immediately as possible, to ensure that the scans are always up to date. E.g. when a new application gets deployed to the cluster the auto-discovery should detect this application automatically and directly with minimal time delay. This behavior is further described on a conceptual level in section 4.2 and on a more technical level in section 4.4.

While this prototype is only focussed on Kubernetes, the same concept can be applied to other (private, public, or hybrid) cloud environments. The same auto-discovery aspects can be used in these environments by using the API of the environment to automatically list all Resources of a specific type.

4.2 Application Lifecycle Tracking

The auto-discovery in the prototype isn't a one time process. Instead of using the API to once list all applications and then dispatching security scans, it listens for relevant events in the Kubernetes API to react directly to new events. These can generally be grouped into two categories:

- **New Resources:** When a new Resource is deployed to Kubernetes the auto-discovery prototype should automatically dispatch a scan to assess the security of the Resource.

[1] If LoadBalancers are supported in a cluster depend on the support of the Cloud Controller Manager installed in the cluster and the underlying CSP

- **Updated Resources:** Updated Resources should be compared to the existing scans in the cluster and ensure that the existing scans still apply to the Resource. If the application has changed in a significant way (e.g the application has changed the container image) the scans should be repeated for the new Resource version.

Tracking the events, rather than listing all Resources on a specific schedule, gives the auto-discovery the ability to directly react to updates and not have to rely on a scan schedule to repeat the scans.

How the lifecycle tracking is handled on a technical level in the prototype is described in section 4.4.

4.2.1 Comparison to Continuous Integration Approaches

The most common approach to implement security automation for software projects is to add scanning tools into the CI pipelines of the projects. This ensures that the scanners are run once for every change made to the software.[11, 27]

The auto-discovery prototype doesn't integrate into CI pipelines as it only looks at the runtime environment. The lifecycle is inferred by the events regarding the applications in the cluster.

A comparison between these two approaches to automatically assess application security aspects is out of scope for this work as its focus is the assessment of application using the information of the runtime environment.

4.3 Prior Art

Some products and projects exist in this space which have at least in part similar goals. They come in two varieties, commercial products provided by cloud providers or security vendors or open-source or open-core[2] tools developed by organizations or individuals.

[2]Open-core tools in this work refer to open-source tools with closed-source extensions or integrations provided by the company maintaining the project.

All of the three major cloud providers offer products in this space including AWS Inspector[57], Google Security Command Center[103] and the Azure Security Center[64]. These products provide several automatic assessments for specific products in their portfolio. Most of these are limited in their ability to auto-discover cloud resources, and the lifecycle tracking goals of the prototype listed in section 4.2, as these tools focus on their specific cloud platform and are closed source these will not be further investigated in this comparison of prior art.

There exist some tools by independent security vendors that aim to provide a unified security platform across the different cloud providers. Most of these tools are closed-source commercial products which make it hard to assess their functionality. Other than the commercial tools there also exists some open-source / open-core projects. One notable tool is the *CloudSploit*[75] project maintained by the security vendor Aqua Security[55]. CloudSploit performs configuration scans of cloud projects/accounts for its users, by using the API of the CSP to list all cloud resources of the project/account .[75]

Listed below is a list of three Kubernetes specific open-source / open-core tools in this field. This entire field is relatively new, especially the Kubernetes specific tools. Two of the three projects listed below were only released during the duration of this work. As such, they had only a limited influence on the prototype, as most of the work was already completed at the time of their release.

- **Starboard (open-source / open-core):** Starboard is a Kubernetes specific security tool by the security vendor *Aqua Security*[55].[43, 164] Starboard performs container image vulnerably scanning, using the *Trivy*[168] scanner, in the Kubernetes cluster, see section 4.7, and Kubernetes specific scans using *kube-hunter* and *kube-bench*, see section 4.8. Starboard at the moment doesn't perform auto-discovery, the scans need to be started manually. An auto-discovery like feature is included in the roadmap of the project.[165] [164]

- **DAST Operator (open-source):** Dynamic Application Security Testing (DAST) Operator by the cloud vendor *Banzai Cloud*[65] is a project to automate dynamic security scans using OWASP ZAP[151].[22, 82] The DAST Operator help to deploy ZAP and automatically dispatching ZAP Scans for both Kubernetes Services and Ingress Resources based on the annotations set on the Resources. The DAST Operator is the only of the three mentioned projects to be released before the prototype was developed. This project behaves similarly to the automatic

assessments of HTTP Service and Ingress Resources of the prototype, see section 4.6. The auto-discovery implementation is based on the same underlying technology to detect Resources in the Kubernetes API(*kubebuilder*[115]), but is lacking in the lifecycle detection of Services and Ingress Resources, as the scans are only run once, without automatic re-scans on changes to the underlying applications, see section 4.2. [82]

- **Kubei (open-source):** Kubei is Kubernetes specific open-source project to run container image vulnerability scans using the *Clair*[71] image vulnerability scanner by the security vendor *Portshift*.[45, 116] Kubei scans are not completely automated but require a manual button click on their web UI to dispatch a scan. Kubei will then dispatch scans for all container images used in all namespaces of the cluster[3]. [116]

4.4 Architecture of the Prototype

One of the primary research focus of this work is to investigate how the Kubernetes API can be used to discover and then automatically assess security aspects of applications which run on the Kubernetes cluster.

Research Question: How can the information from the Kubernetes API be used to automatically assess security aspects of applications running inside a Kubernetes cluster?

To investigate the possibilities of Kubernetes and its API this work has documented what components make up a Kubernetes cluster and how they work in chapter 2. One of the most important Kubernetes components highlighted in chapter 2 was the Controller Manager, which is the component in Kubernetes which implements most of the features in Kubernetes. One interesting aspect of the Controller Manager is, that it only communicates with the Kubernetes API Server. The API Server is also available to users and workloads of the cluster, e.g. API access via auto mounted Service Account Tokens, see section 3.3. This gives users and workloads the possibility to add or extend Kubernetes behavior by adding custom controllers. As this is using the same

[3] It can be provided with a deny list of namespaces to ignore, which default to a list of system-level namespaces.

architecture and API as the API Server this lets these custom controllers integrate very deeply with Kubernetes. Controllers are described in the relevant Kubernetes focused literature as the recommended way to extend the behavior of Kubernetes Resources. In the case of the auto-discovery prototype, the added behavior is the security assessment of the Resources. [26, 28]

Figure **4**-1: Control / reconciliation loop of the auto-discovery controller.

This architecture of the auto-discovery being build as a custom Kubernetes controller lets the auto-discovery implement all goals described in section 4.1. As a Kubernetes controller, the auto-discovery subscribes to all events related to the Resources considered relevant for the security of an application, e.g. Ingress Resources. For every event (Resource creation, update or delete) the auto-discovery controller decides if a new scan should be dispatched, or if the Resource is already covered by an existing Scan. As the controller is using an event stream[4], these events are processed directly as they happen. As these events are dispatched for all updates to a Resource, including its creation and deletion, this enabled the auto-discovery to track the complete lifecycle of the Resources directly with a close to zero delay, see application lifecycle tracking goals in section 4.2. An adjusted diagram of the reconciliation loop for the auto-discovery prototype can be found in fig. **4**-2.

The auto-discovery service for Kubernetes is build using the *Kubebuilder* project[115], which allows users to write reconciler functions, following the reconciliation loop pattern from the Kubernetes Controller Manager. Kubebuilder itself is part of the larger Kubernetes project and contains several utilities and helper to write Controllers in the *Go*[167] programming language.

[4] Using a watch operation in the Kubernetes API.[117, 26]

4.5 SecureCodeBox Security Test Orchestration

The complete architecture of the prototype consists of two parts. The auto-discovery controller which is responsible for discovering security relevant Resources and creating scans to assess their security aspects. The execution of these scans is not handled by the auto-discovery, but by the open-source security test orchestration system OWASP secureCodeBox.[159] The version of the secureCodeBox used is an alpha version the second major release of the secureCodeBox.[158] This alpha version is build based on Kubernetes, using Kubernetes Custom Resource Definitions (CRDs)[26]to model its API. CRDs allow to add new Resource Types to the Kubernetes API which can then be used by custom controllers to add additional behavior to Kubernetes. This is used by the secureCodeBox to allow Kubernetes users to directly schedule and run security scanning tools on their cluster, with the execution and orchestration of the scans being controlled by the secureCodeBox. This allows the auto-discovery service to start scans by creating the corresponding Resource in the Kubernetes API.

```
apiVersion: "execution.experimental.securecodebox.io/v1"
kind: Scan
metadata:
  name: "zap-scan-juiceshop"
spec:
  scanType: "zap-baseline"
  parameters:
    - "-t"
    - "http://juice-shop.example.com:3000"
```

Listing 4.1: Example secureCodeBox v2 Scan Definition

To provide a simpler interface for the auto-discovery to scan the discovered services, a new category of CRDs in the secureCodeBox were created. The default Resources require to describe the exact scan which should be run, including its configuration via command-line flags. An example of the CRD of a scan in the secureCodeBox can be found in listing 4.1. This is not optimal, as the Kubernetes auto-discovery is meant to be one of multiple auto-discovery services for specific cloud environments (e.g. AWS[52], Azure[141], OpenStack[147]) and each of these would have to include this technical configuration for each scanner. To avoid this duplication of scan configuration for every type of auto-discovery, a new target CRD was introduced, which allows the auto-discovery services, in this case, the Kubernetes auto-discovery service, to describe the target which should be scanned, without describing exactly how it should be scanned.

The secureCodeBox is then responsible to start the scans which fit the definition of the targets. For this work two target types were introduced:

- **Hosts:** A *Host* target describes a network addressable machine, with one ore multiple services / ports. These ports can be of different types, though this work only uses auto-discovery for HTTP and HTTPS type ports. An example of a Host target can be found in listing 4.2.

- **ContainerImages:** A *ContainerImage* target contains the image reference URL of a container registry from which the image can be pulled. Optionally the ContainerImage target can also contain a reference to a *ImagePullSecret* in Kubernetes if the registry is not public and requires authentication.

```
apiVersion: targets.experimental.securecodebox.io/v1
kind: Host
metadata:
  name: juice-shop.example.com
spec:
  hostname: juice-shop.example.com
  ports:
    - type: ssh
      port: 22
    - type: http
      port: 3000
```

Listing 4.2: Example secureCodeBox v2 Target Definition

The areas of separation between the Kubernetes auto-discovery and the secureCodeBox is visualized in fig. **4**-2.

Another aspect in which the secureCodeBox running directly on Kubernetes is supporting this prototype, is the area of transparency of the scan results and the scan process. An autonomous team using one or multiple namespaces in a shared Kubernetes cluster will be able to see the scans performed by the auto-discovery and be able to see the results of the scans for their individual services, as the scans are executed inside their Kubernetes namespaces.

A central security team inside the organization can access all security findings in the organization by configuring the secureCodeBox to persist all findings into an Elasticsearch cluster.[90, 157] From the Elasticsearch cluster, the security team can aggregate and

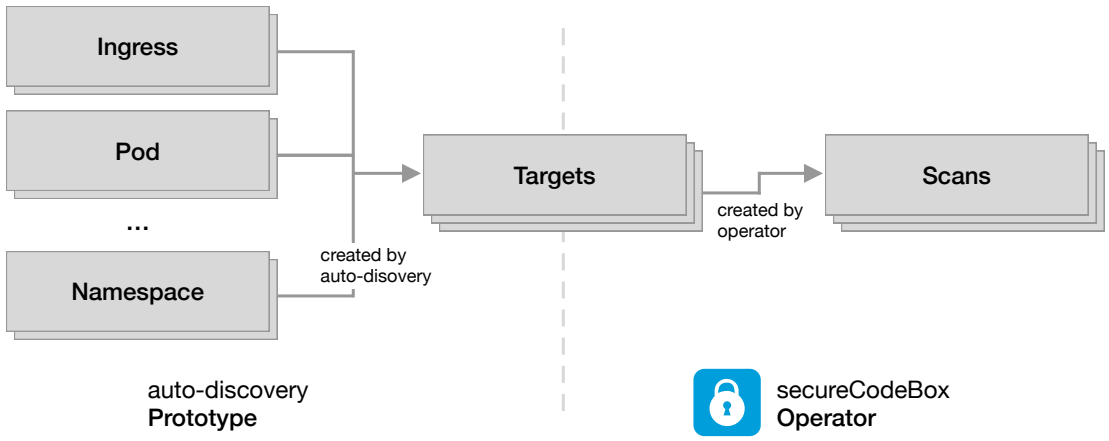

Figure **4**-2: Separation of concerns between Scan Execution and Scan Definition.

analyze the organization's findings and coordinate and support efforts to fix individual vulnerabilities. This enables both autonomous systems to address vulnerabilities on their own while still proving central security teams with the ability to get an overview of the general security stance of an organization, which is a recommended way to organize modern security programs.[20, 137]

4.6 Automatic Security Assessment of HTTP Services

4.6.1 External Attack Surface via Ingress Resources

Ingress Resources in Kubernetes let the user specify how external HTTP directed to the cluster should be handled. In applications not running on Kubernetes, this is often handled by a reverse proxy / virtual hosting solution which routes the HTTP traffic of a single IP address to a specified service that is configured to handle the HTTP traffic for a specific hostname. Kubernetes let users define the same concept in the Ingress Resource, which is then used by an Ingress Controller which was either provided by the CSP or installed onto the cluster by the user. A popular Ingress Controller implementation is the NGINX Ingress Controller[143], which uses the popular NGINX Project to route the traffic to the right service. [23, 123]

Kubernetes Ingress also allows to specify a certificate that should be used by the Ingress Controller to terminate the TLS connection of the HTTPS request.

A complete example of an Ingress can be found in listing 4.3. This example uses the NGINX Ingress Controller[143] to route all HTTP traffic with the hostname *juice-shop.demo.securecodebox.io* to the *juice-shop* Service in the same Kubernetes Namespace as the Ingress Resource. The example only contains the Ingress definition for a single Application, but a single Ingress Resource can also be used to define how traffic for multiple applications is routed. This is handled by the prototype by starting individual scans for every hostname found in the Ingress Resource, as not all scanners support to scan multiple targets at the same time or make the configuration of these scans more complex.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: juice-shop
  namespace: juice-shop
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-production
spec:
  tls:
    - secretName: juice-shop-tls-secret
      hosts:
        - juice-shop.demo.securecodebox.io
  rules:
    - host: juice-shop.demo.securecodebox.io
      http:
        paths:
          - path: /
            backend:
              serviceName: juice-shop
              servicePort: 3000
```

Listing 4.3: Example Kubernetes Ingress manifest for the Juice Shop application

All of the application level vulnerabilities described in section 3.2 can be detected, at least to some extend, by running DAST scans or manual penetration tests against the hostnames specified in the Ingress Resources.

4.6.2 Internal Attack Surface via Service Resources

Aside from the assessment of the external attack surface via the Ingress Resources, the auto-discovery uses the Service Resources to assess the internal attack surface of the application inside the cluster. Services, in their default configuration, are used to route cluster internal traffic.[5] This allows the auto-discovery system to also assess the internal attack-surface. Accessing both the external as well as the internal attack surface provides several benefits:

- Assessing the entire attack surface is more along the line with modern security methodologies like Zero Trust[44, 174]/BeyondProd[68] which recommend not to imply a certain trust level to traffic only because it coming from an internal network.

- Assessing both internal and external attack surface also allows easier evaluation of security tools like Web Application Firewalls (WAFs), which in Kubernetes are often deployed as part of the Ingress controller.[144] This lets users easily evaluate if a newly found vulnerability in a service is covered by the rules of the WAF.

- Better integration with third-party Kubernetes Add-ons like Service Meshes (e.g. Istio[110] or linkerd[139]) which often still use Services but replace Ingresses with the own custom solution. If the prototype would only work with Ingress Resources, these setups would not be able to be auto-discovered at all.

A problem with Service Resources in this prototype is that they are not HTTP specific but allow all TCP and UDP traffic, in contrast to Ingress Resources which only support HTTP / HTTPS traffic. This was worked around, by filtering on official and commonly used HTTP ports[6] or service ports which use the name `http` or `https`. This allows the auto-discovery to work around this limitation. Scans for ports that are mistakenly guessed to be HTTP / HTTPS ports will fail as the scanners won't be able to connect to it. This could potentially be fixed by probing the port, e.g. using an NMAP portscan with version detection[2], to detect which protocol is supported. This was not implemented in the version of the prototype.

[5] Unless they are configured to be of type *NodePort* or *LoadBalancer*.

[6] The used port list in this prototype is 80, 443, 8080, 8443, 3000, 5000, and 8000. With 443 and 8443 being directly discovered to support HTTPS.

4.6.3 Scanners used for the Security Assessment

OWASP ZAP

The *OWASP ZAP* project is the most popular (open-source) scanner for web applications.[30, 31] ZAP is a DAST scanning tool, meaning that it performs black-box scans that only use HTTP / HTTPS to find vulnerabilities in the application without access to its source code. The ZAP project is part of the OWASP foundation. ZAP is primarily a desktop application, but it also provides an API which can be used to programmatically start and configure scans.[171] [66]

ZAP is able to automatically scan for all of the application level vulnerabilities listed in section 3.2. That ZAP can detect all the vulnerabilities is no guaranty that it will detect them in a scan, as many other factors can prevent ZAP from being able to identify specific vulnerabilities. The ZAP documentation contains a list detailing how the scans ZAP performs map over to the OWASP Top Ten and also give a brief indication of how well each of these vulnerabilities is covered by ZAP.[173] Some of the tests, e.g. test regarding broken authentication, see OWASP Top Ten[12] , only work correctly if ZAP has been configured specifically for the application which it is scanning as the test need to have application specific knowledge to detect any meaningful vulnerability. The auto-discovery prototype currently does not implement a way that lets users specify application specific configuration for the automatically created scans meaning that these vulnerabilities cannot be automatically detected, see more on this limitation in section 5.2.3.

Nikto

Nikto is a tool to identify and assess web servers and their security. To do this Nikto sends out requests and compares various aspects of the response against its database to see if parts of the response are directly attributable to specific server technologies. Besides identifying server types and versions Nikto contains a database of default files/programs for specific server types, e.g. management/administration interfaces. [145]

Nikto is useful to identify components with known vulnerabilities, see section 3.2, as

it can identify both the version of software used but also insecure extensions for the software.

Nikto also provides some functionally to actively scan for vulnerabilities like SQLi and XSS. These active detection features have been disabled in the auto-discovery, as they are already handled by ZAP and they increase the time Nikto takes to complete its scan.

SSLyze

SSLyze is a tool to scan SSL / TLS hosts for miss-configuration and vulnerable implementations.[163] SSLyze is used in this prototype to scan the TLS stack of the HTTPS hosts to ensure that their deployment follows best practices and uses modern and secure protocol versions and cipher suites. Additionally, SSLyze is also able to detect security vulnerabilities, like the *Heartbleed* vulnerability in OpenSSL[6], in the TLS stack of the server.

SSLyze scans are automatically created for Ingress Resources containing TLS configuration and Services with typical HTTPS ports (ports 443 and 8443).

4.7 Automatic Security Assessment of Container Images

Pods are the smallest individually deployable Resource in the Kubernetes API.[130] A Pod consist of one or multiple containers, but it is not possible to deploy a container in Kubernetes without wrapping it in a Pod.[130] Other workloads like Deployments, StatefulSets or Jobs are higher-level Resources which are used to manage one or multiple Pods.[134, 23]

```
apiVersion: v1
kind: Pod
metadata:
  name: juice-shop
  namespace: juice-shop
```

```
spec:
  containers:
  - image: bkimminich/juice-shop:v11.1.2
    name: juice-shop
    ports:
    - containerPort: 3000
      name: http
status:
  containerStatuses:
  - containerID: docker://87c2638594558627eb6741febbc574e03303a01d1a4ce...
    image: bkimminich/juice-shop:snapshot
    imageID: docker-pullable://bkimminich/juice-shop@sha256:7524757ae91...
    name: juice-shop
    ready: true
    started: true
    state:
      running:
        startedAt: "2020-07-07T09:19:58Z"
```

Listing 4.4: Excerpt from an example Kubernetes Pod definition for the Juice Shop application, including relevant status fields

As Pods are the only way to deploy containers in Kubernetes, they pose a good target for the auto-discovery, as covering Pods in the auto-discovery will also cover all of the other workloads in Kubernetes. An example of a Pod definition, including relevant fields from its status, can be found in listing 4.4.

For the auto-discovery, it is important to ensure that it can scan the same version of the container image that is deployed as a Pod to ensure that the scan results match the image. This is not always given when using the image reference from the Pod specification. The image reference (e.g. `bkimminich/juice-shop:latest`) consist of two relevant parts the image repository (`bkimminich/juice-shop` or the fully qualified name, as this image uses the default image repository from Docker Hub: `docker.io/bkimminich/juice-shop`) and the image tag (`:latest`). Image Tags are not immutable, which means that the same image pulled at different times can lead to two different versions of the image pulled. Using immutable image tags is considered to be a best practice, as it ensures that deployments are reproducible.[42][7]

The problem of immutable tags is avoided in the auto-discovery by using the `imageID` status field of the Pod. This field is set by the kubelet after pulling the image from

[7] The immutability of container image tags can be enforced by some image registry.[106]

the registry and includes a hash of the image content. This hash is then used in the image reference used by the auto-discovery prototype to run the scan against the exact container image version deployed in the Pod.

Trivy

Containers are started from container images. Container images consist of a root filesystem and general information (e.g. exposed ports or environment variables). The root filesystem contains the code or compiled artifacts of the application alongside operating system utilities like language runtime or utilities like shells (e.g. *bash* / *zsh*). [42]

Scanning tools can use these images to automatically inspect security aspects of the images. The extend of security aspects depends on the tool. The most common is the detection of vulnerable OS packages. Some tools also cover other aspects like detecting known malware in the image or inspecting if the container image is built to best practices (e.g. using a non `root` user). [42]

While the previously mentioned tools are usually classified as DAST tools as the scan the running application, image scanning tools better fit into the Static Application Security Testing (SAST) classification as the scan the bundled application artifacts and not a running instance.

To analyse the container images for security vulnerabilities the auto-discovery prototype uses the image scanning tool Trivy[168]. Trivy is an open-source image scanning tool by the security vendor Aqua Security[55]. Other comparable open-source tools include *Clair* (maintained by RedHat[153])[71] and *Anchor* (maintained by Anchor Inc.)[53].[42]

Trivy was chosen as the container image scanning tool for the prototype for two primary reasons:

1. Trivy is simpler to set up and its execution model fits better into the secureCodeBox execution flow as it doesn't require a Postgres database for its execution like Clair and Anchor. [168, 71, 53]

2. Trivy is able to not only scan operating system packages (e.g. installed via `apt-get` (Debian / Ubuntu), `yum` (RedHat, Fedora, CentOS) or `apk` (Alpine)) but also application level packages (e.g. installed via `npm` (Node.js) or `composer` (PHP)). [168]

Scanning of container images is a relatively new discipline in the Security field. At the moment, there are no established best practices to where the image scan should be performed. Three different locations where the image scan can be performed exist:

1. **On the Build Server:** In this case, the container image gets scanned directly after it was built on the build / CI system. This ensures that no vulnerable artifacts get published. Scanning directly after the image is build has the downside that vulnerabilities that are discovered after the image has been build and scanned are not discovered as scanning on CI is a *point in time* scan and the results cannot be easily updated when new vulnerabilities get published.[42]

2. **On the Image Registry:** Many container image registries now come with options to automatically scan uploaded images for vulnerabilities.[105, 99, 154] Depending on the registry this can often be easy to enable and in contrast to scanning on the build server. Scanning inside the registry also allows a continuous scanning model where new vulnerabilities can be detected in previously build images. [99, 42]

3. **On the Runtime Environment:** Scanning the Images in the same environment where they are used. This has the benefit that the runtime environment knows exactly which image versions are actually in use, which cuts down the number of false-positive findings for old versions (tags or digests) of the image which are not used anymore. As the prototype runs inside the runtime environment this is the only viable option to perform the scan in the auto-discovery prototype. This approach is similar to the *kubei*[116] tool mentioned in the prior art section, see section 4.3.

4.8 Automatic Security Assessment of Kubernetes Namespaces

So far the auto-discovery has only assessed general, non Kubernetes specific, security aspects of the discovered applications. Kubernetes specific security aspects are as-

sessed by the auto-discovery by dispatching Kubernetes specific security scanners for each namespace. These assess if the configuration deployments follow best practices, e.g. using the privilege escalation mitigations mentioned in section 3.3.

Kubeaudit

Kubeaudit[114] is an open-source tool by the e-commerce platform Shopify[162] to assess if the configuration for Resources in Kubernetes Cluster follow security best practices. It can be used in manifest mode, where it scans a local deployment file before it is deployed, or in a mode where it assesses the Resources in a running cluster.

The checks (auditors) integrated into kubeaudit, cover all best-practices mentioned in section 3.3, to mitigate privilege escalation in Kubernetes workloads, e.g. checking if containers use the `privileged` flag or if containers are using `root` users.

Kube-Hunter

kube-hunter[113] is an open-source tool by the security vendor AquaSecurity[55] which can scan Kubernetes clusters for security vulnerabilities. Kube-hunter provides multiple different modes in which it can be used. The mode used in the prototype is the `pod` mode, in which kube-hunter is deployed as a Kubernetes Job and emulate which attacks an attacker could perform when they have compromised a Pod in the namespace to elevate their privileges. [113, 155]

4.9 Example Auto-Discovery Process

To demonstrate the behavior of the prototype, listed below is a step by step example of a user deploying a new application and the steps taken by the auto-discovery to discover and scan it. A simplified visualization of these steps can be seen in fig. **4**-3. To keep this somewhat concise this will only show the auto-discovery of Pod Resources, other Resources like Services and Ingress behave similarly.

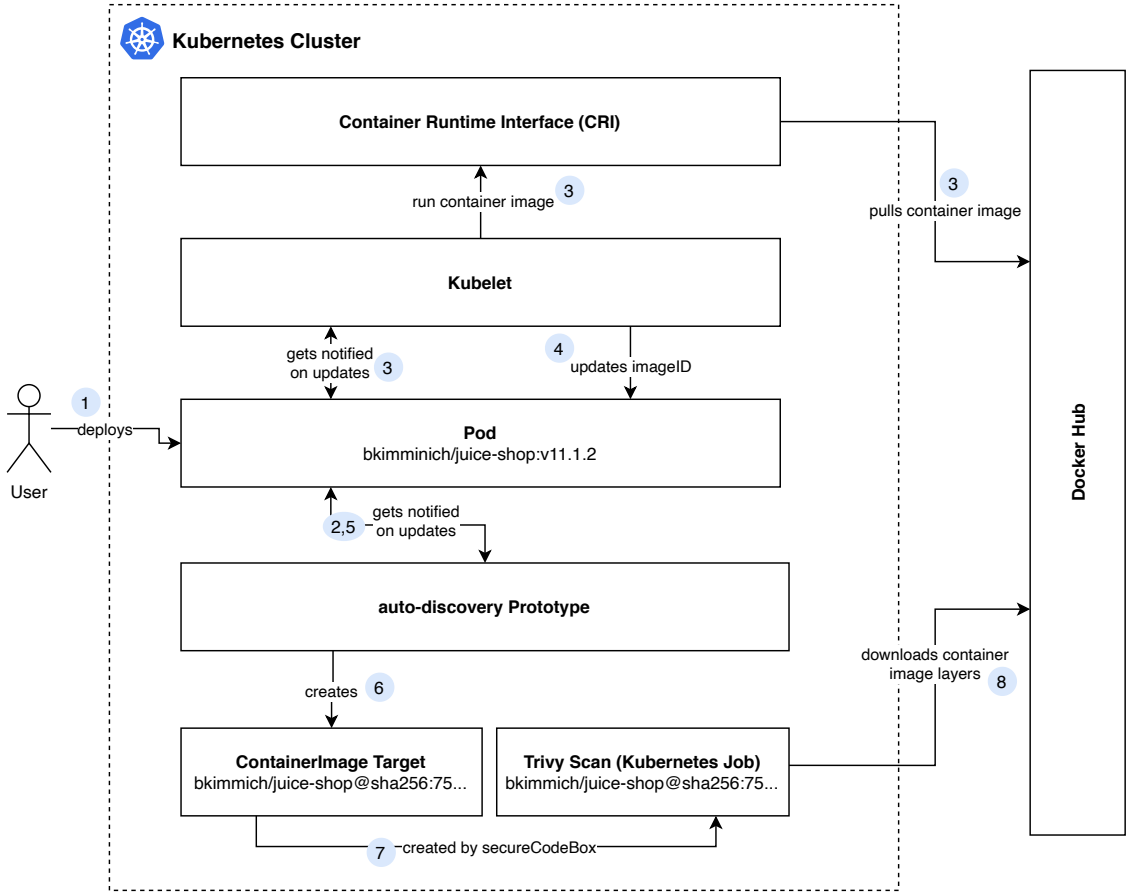

Figure **4**-3: Simplified system interactions of the auto-discovery prototype.

1. The user creates a Deployment in Kubernetes by running `kubectl create deployment --image bkimminich/juice-shop:v11.1.2 juice` in their command line. This sends a request to the Kubernetes API Server and creates the Resource in etcd. The Deployment Controller in the Kubernetes Controller Manager gets notified of the new deployment and creates a new Pod to match the desired replication count of one of the Deployment.

2. The auto-discovery `PodScanReconciler` get automatically called, as it is performing a watch operation on Pod Resources in the cluster, indicating that something a Pod has been created or updated. The `PodScanReconciler` then inspects the Pod specification of the event. As the Pod was just created in the API but wasn't properly started by the kubelet, the `imageID` field in the container status isn't yet populated. This causes the `PodScanReconciler` to ignore this event as it can not perform a scan until this field is populated, see section 4.7

3. Parallel to the previous step, the Pod gets assigned to an available Node by the Kubernetes Scheduler. The kubelet running the assigned Node will then instruct the CRI to pull the container image from the specified registry and start the container.

4. Once the container image was downloaded and the container itself was started the kubelet updates the status field of the Pod. This status contains the exact `imageID` of the container image started.

5. As the Pod Resource was updated again, the `PodScanReconciler` of the auto-discovery gets called again. This time the `imageID` is set, the `PodScanReconciler` checks if this particular image version was already scanned by checking if such a scan already exists in the Kubernetes API.

6. As the image is deployed for the first time, no scan for this image version exists yet. This prompts the `PodScanReconciler` to create a new `ContainerImage` secureCodeBox target for the version of the container image.

7. The secureCodeBox will detect this new target type and automatically create a Trviy scan for the container image of the `ContainerImage` target.

8. Trivy will then get started as a Kubernetes Job by the secureCodeBox, download the layers of the image, and perform its scans on these layers. The results from Trivy scans will then get converted by a parser inside the secureCodeBox to a standardized findings format.

5 Prototype Verification

In this chapter, the functionality of the prototype is evaluated by its ability to properly discover and runs security assessments against a specially setup Kubernetes cluster with regards to the research question of this work.

5.1 Evaluation Cluster Setup

To evaluate the prototype a Kubernetes cluster was set up, on which the prototype could be tested and evaluated. The cluster contained two Kubernetes Namespaces, both containing an intentionally vulnerable open-source web application. Both applications consist of a Deployment creating three replicas (Pods), a ClusterIP Service to load-balance traffic to the application under a single address, and an Ingress that exposes the applications to the Internet. A visual overview of the deployments can be found in fig. **5**-1.

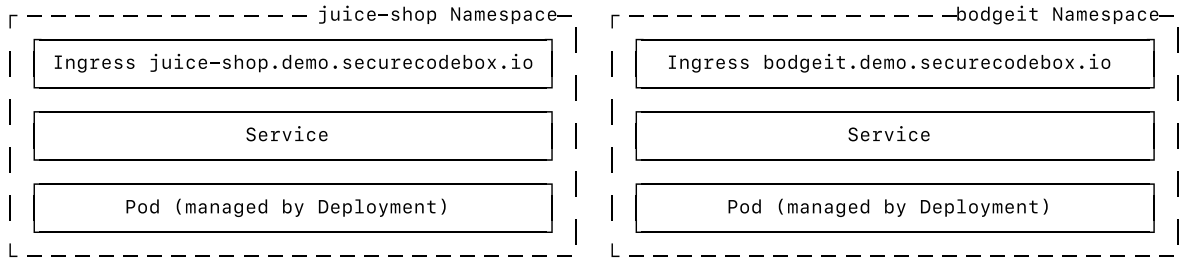

Figure **5**-1: Kubernetes Resources of the Demo Environment

The first application used in the evaluation is the OWASP Juice Shop, an intentionally vulnerable web application written as a Single Page Application (SPA).[149][1] SPA in this case refers to applications, which are applications where the HTML is primarily

[1] Disclaimer: The author of this work is part of the Juice Shop Core Team.[150]

rendered in the browser of the user and not on the server. The frontend of Juice Shop is written in TypeScript using the Angular framework. The backend is written in JavaScript using Node.js as the runtime. Juice Shop provides an official Docker image on Docker Hub which was used for the Deployment.[85] The version of Juice Shop used in the evaluation was `11.1.2`.[38] The results of the auto-discovery prototype and problems encountered during the evaluation for Juice Shop can be found in section 5.2.

The second application used, is the BodgeIt store. BodgeIt is a intentionally vulnerable application written in Java using Java Server Pages (JSP).[166] BodgeIt is an older application, with the last actual code change committed in 2014.[69] This makes BodgeIt an interesting scan target for the evaluation as it is a good example for an older application which was migrated to Kubernetes without adopting or updating them. BodgeIt, like Juice Shop, also provides an official container image[87], but this was not used in the evaluation as the image manifest format was outdated and not compatible with trivy, the image vulnerability scanning in the prototype, see section 5.3.3.[37]

The evaluation was run on a managed Kubernetes cluster from DigitalOcean[83] running Kubernetes version `v1.18.6`. The cluster used *nginx-ingress* version `v0.34.1`[143] as its Ingress Controller, with *cert-manager* version `v0.16.0`[70] to automatically creating valid TLS certificates for the Ingress Resources via Let's Encrypt[138].

5.2 Auto-Discovery Scan Results for OWASP Juice Shop

5.2.1 Scans Created by the Auto-Discovery Prototype

The auto-discovery prototype was able to create the scans laid out in chapter 4. An overview of the scans can be found in fig. **5**-2. A screenshot showing the actual scans via the Kubernetes command-line tool *kubectl* can be seen in fig. **5**-3.

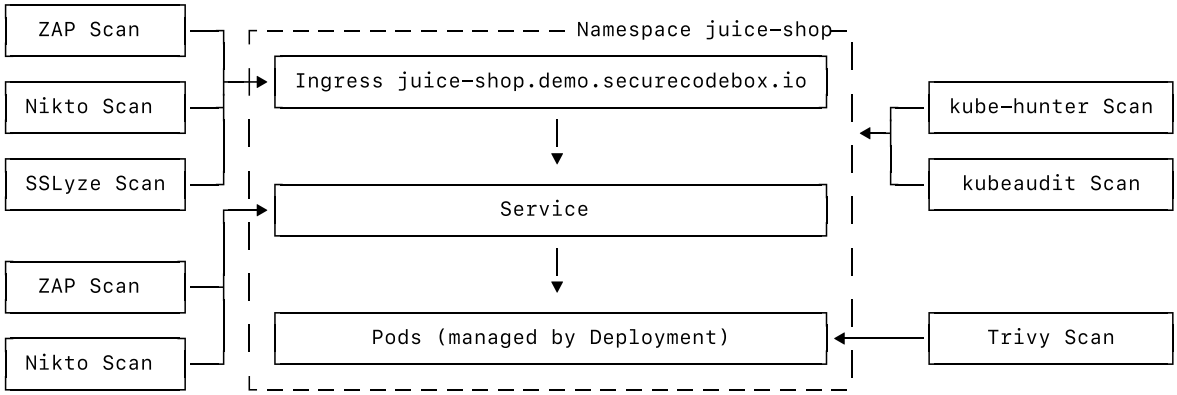

Figure **5**-2: Overviews of the Scans created for the Individual Resources of the juice-shop Kubernetes Namespace.

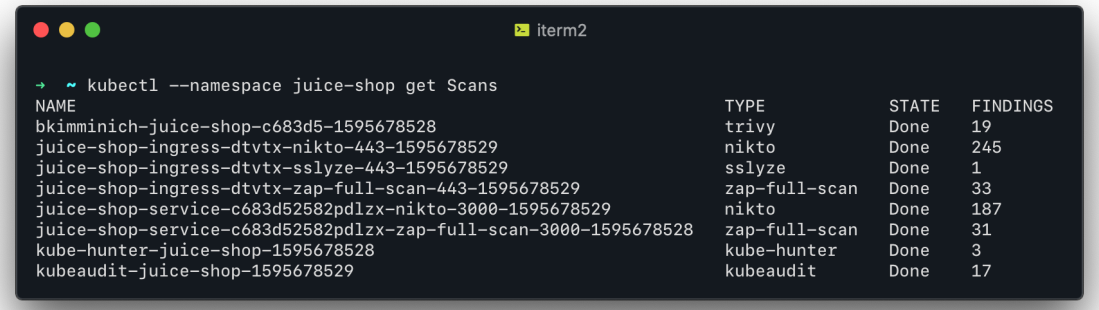

Figure **5**-3: Scans created by the auto-discovery for the Resources in the juice-shop Kubernetes Namespace.

5.2.2 Finding Overview

The scans were able to identify a large number of findings (536 findings). A visual overview of the findings, broken down by their scanner and by their categories assigned by the secureCodeBox, can be seen in figure fig. **5**-4. The diagram shows that most of the findings (420 of 501 findings) were identified by Nikto and are of the category `Potential Backup File`. These were all found to be false-positive findings, which Nikto has falsely identified as it could not properly identify the HTTP response of the Juice Shop as a failed requests. These findings are excluded in further diagrams, as they increase the complexity of the results. A version of the diagramm without the `Potential Backup File` findings can be found in fig. **5**-5, this shows a more concise and balanced set of findings. A complete table of all findings name, severity, and scan type can be found in the appendix A.1.2. Listed below is a summary of notable results

grouped by the scanners:

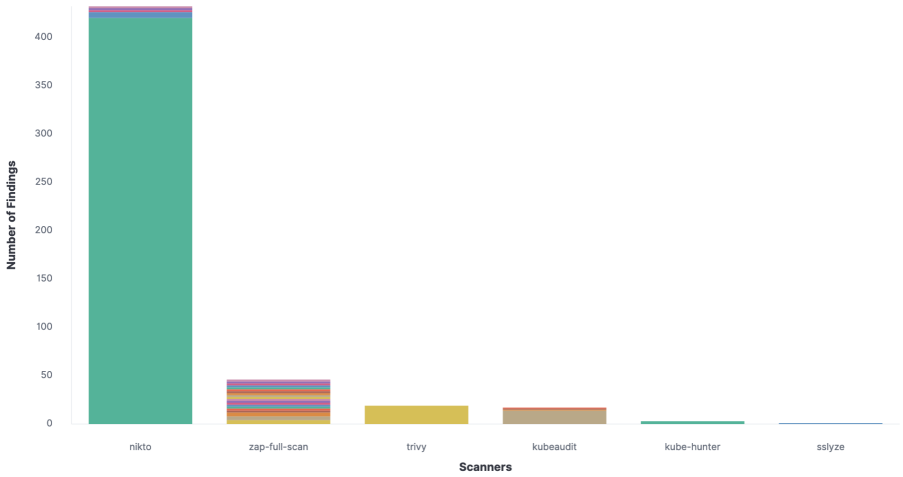

Figure **5**-4: All findings identified by the scans created by the auto-discovery for OWASP Juice Shop, grouped by scanner and finding category. Colored stripes indicating different finding categories.

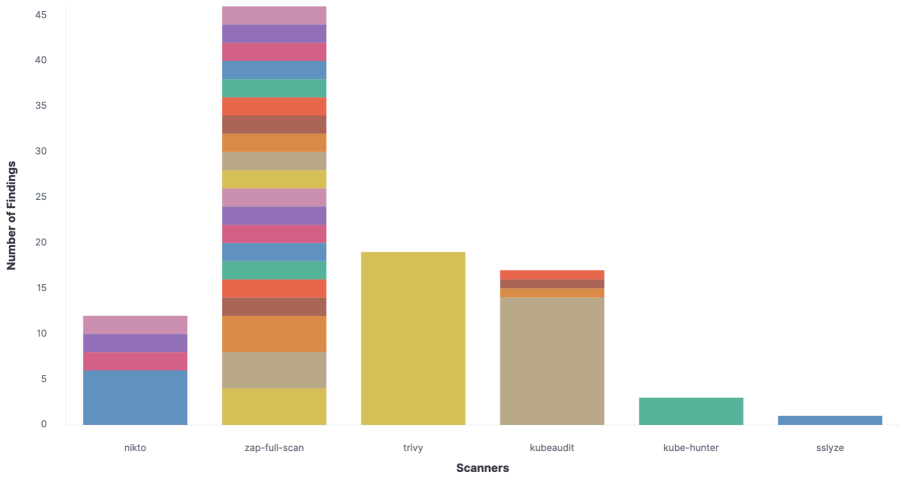

Figure **5**-5: All findings identified by the scans created by the auto-discovery for OWASP Juice Shop, excluding Potential Backup File findings. Colored stripes indicating different finding categories.

OWASP ZAP Findings

- SQLi vulnerability in the search api of the Juice Shop. Other SQLi vulnerabilities, e.g. in the login of Juice Shop, unfortunately were not identified

- Missing CSP HTTP header

- Identified hidden `/ftp/` directory and files contained in it
- Identified overly permissive Cross-origin resource sharing (CORS) / Cross-Domain Misconfiguration of the `Access-Control-Allow-Origin` HTTP header
- Multiple missing Cross-Site Request Forgery (CSRF) tokens

Other then the findings listed above, ZAP also identified some various low severity findings. The ZAP Scan for the Ingress produced more two more findings than the Service scan, these were caused by the nginx-ingress controller adding additional HTTP header to the response. The number of false positives was relatively low, around 5 false-positive findings for the 31 Service / 33 Ingress findings identified by ZAP.

Trivy Findings:

- Authentication bypass in the `jsonwebtoken` npm package. This allows attackers to forge JSON Web Tokens (JWTs) of other users.
- XSS vulnerability in the `sanitize-html` npm package.

Trivy didn't identify any vulnerabilities in the operating system libraries. This is likely not a defect in Trivy but caused by the used Juice Shop image using an up to date minimal base image, using Alpine Linux[48]. Alpine Linux only contains a limited number of dependencies to reduce the attack surface.[48]

Nikto Findings

- Identified the hidden `/ftp/` directory from the `/robots.txt` file from Juice Shop.

The Nikto Findings for Juice Shop were not helpful. Findings that Nikto reported were already reported by ZAP with greater detail. Nikto also introduced a lot of noise to the overall findings by introducing a large number of false-positive findings (see note on `Potential Backup File` Finding above). Nikto also created some Findings to outdated / deprecated best-practices like a missing `x-xss-protection` header.[34, 140]

SSLyze Findings

SSLyze did not find any notable results. The default TLS config of the *nginx-ingress* controller uses a modern set of TLS versions (TLS 1.2 and higher) and a set of modern cipher suites.

Kube-hunter Findings

- Granted `CAP_NET_RAW` Linux capability, which would allow the Pod to perform advanced networking attacks like Address Resolution Protocol (ARP) spoofing.
- Access to the automounted ServiceAccount token. As the default service account in the namespace does not any associated RBAC roles this didn't lead to a compromise.

The kube-hunter results were relatively limited, as the namespace didn't contain any setting compromising its security. But they showed the namespace could be better configured to further limit the ability of an attacker to extend their privileges.

Kubeaudit Findings

- Juice Shop Deployment doesn't enforce a non `root` user
- The default ServiceAccount of the namespace doesn't have the `automountServiceAccountToken` property set to disabled
- Juice Shop Deployment has access to 14 Linux capabilities, which should be reviewed if they can be dropped

The kubeaudit findings are relatively concise and pointed to potential mitigations which can be applied to improve the security posture of the deployment.

5.2.3 Problems while Scanning Juice Shop

Container Images

For the Trivy container image scan to work properly the OWASP Juice Shop container image had to be altered. OWASP Juice Shop had changed their build process configuration in version `v10.2.1` to not create a `package-lock.json` file during its installation of Node.js dependencies via npm.[41, 40] This file, which keeps track of the installed versions of npm dependencies, is used by Trivy to compare the installed versions against databases of vulnerable packages. This change has been reverted for the container image build in OWASP Juice Shop version `v11.1.2` by the author of this work, to allow the image scanning to work correctly.[38]

ZAP Scan Configuration

Another problem identified while using the prototype to evaluate the performance of the prototype is that the ZAP Scans created by the auto-discovery were not properly configured to effectively scan the Juice Shop application. ZAP is not a *point and shoot* tool but requires a base level of configuration for the scan target. When this configuration is left out, the results ZAP can detect are limited. An example here is how ZAP explores the application to find its endpoints. This is handled by spiders, which fetch the web page and then recursively explore (*spider*) every link to other pages included. ZAP provides two main spiders, the default Spider[172] and the AJAX Spider[170]. The default Spider is not able to properly spider applications that rely on JavaScript. The default spider may only be able to find a limited number of pages/endpoints for these applications as they require the execution of JavaScript to render links to their pages. To avoid these problems, it is recommended to use the *AJAX Spider* which uses a browser to request and render out webpages, which allows it to properly handle pages which use JavaScript.[67, 170]

This is a problem for the Juice Shop as an application as it is written as a SPA, which uses JavaScript to render its pages on the client. As a result, the default ZAP configuration using the default spider was not able to properly spider the application, which caused ZAP to miss vulnerabilities it should be able to identify when configured correctly.

This poses a problem for the auto-discovery, as it's not possible to identify which spider is best for a particular Kubernetes Service or Ingress directly based on their definition. This problem was solved by allowing users to provide additional information/hints on how the application can be properly scanned in the labels of the Kubernetes Resources. This is used in the prototype and the Juice Shop scans shown in this chapter to hint that the ZAP scans should be using the AJAX Spider by using a `zap-hints.auto-discovery.experimental.securecodebox.io/spider` label set to `ajax` on the Juice Shop Service and Ingress, see the YAML manifest used to deploy Juice Shop in appendix A.1.1.

The solution can in the future also be applied to other applications, which require special configuration for scanners to scan them properly. In this case, new types of labels can be created which allow application developers to give additional information about the application to the auto-discovery system. This is also portable to most other cloud environments other than Kubernetes, as most of them allow to attach labels, tags or more generally meta information to objects/resources .[14, 94, 60]

This specific problem might also be mitigated in the future by ZAP improving their detection of applications and selecting the best Spider for them automatically. Improved handling for modern web apps is included on the list of high-level plans of the ZAP Project.[32]

5.3 Auto-Discovery Scan Results for BodgeIt Store

5.3.1 Scans Created by the Auto-Discovery Prototype

As the deployment for both applications are similar, with the main difference being that they use different container images, the types of scans created by the auto-discovery prototype are identical to the scans created for OWASP Juice Shop. A screenshot of the Kubernetes command-line tool kubectl showing all scans created by the auto-discovery for the BodgeIt Store can be seen in fig. **5**-6

```
iterm2

→  ~ kubectl --namespace bodgeit get Scans
NAME                                                                TYPE           STATE   FINDINGS
bodgeit-ingress-zntkt-nikto-443-1595684181                          nikto          Done    6
bodgeit-ingress-zntkt-sslyze-443-1595684181                         sslyze         Done    1
bodgeit-ingress-zntkt-zap-full-scan-443-1595684181                  zap-full-scan  Done    31
bodgeit-service-e22331dba0n8gdm-nikto-8080-1595684181               nikto          Done    7
bodgeit-service-e22331dba0n8gdm-zap-full-scan-8080-1595684181       zap-full-scan  Done    45
j12934-bodgeit-e22331-1595684181                                    trivy          Done    334
kube-hunter-bodgeit-1595684181                                      kube-hunter    Done    3
kubeaudit-bodgeit-1595684181                                        kubeaudit      Done    17
```

Figure **5**-6: Scans created by the auto-discovery for the Resources in the bodgeit Kubernetes Namespace.

5.3.2 Finding Overview

The auto-discovery has identified a similarly large number of finding for BodgeIt (444 findings) as it did for Juice Shop (536 findings). Like in the results for Juice Shop most findings are from a single scanner, other than Juice Shop most of the findings were generated by the Trivy vulnerability scanner. A visualization of the findings grouped by their scanner and finding categories can be found in fig. **5**-7. A similar visualization excluding the findings of the Trivy scanner for better visibility can be found in fig. **5**-8.

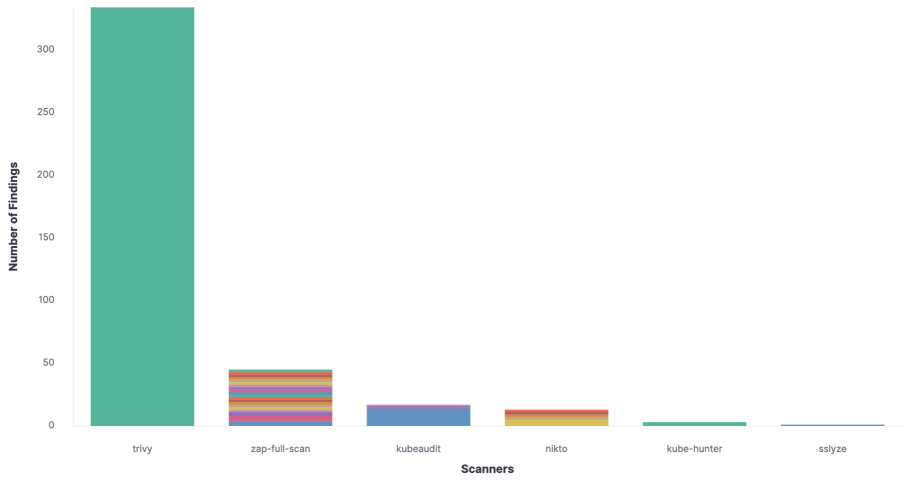

Figure **5**-7: All findings identified by the scans created by the auto-discovery for the BodgeIt Store, grouped by scanner and finding category. Colored stripes indicating different finding categories.

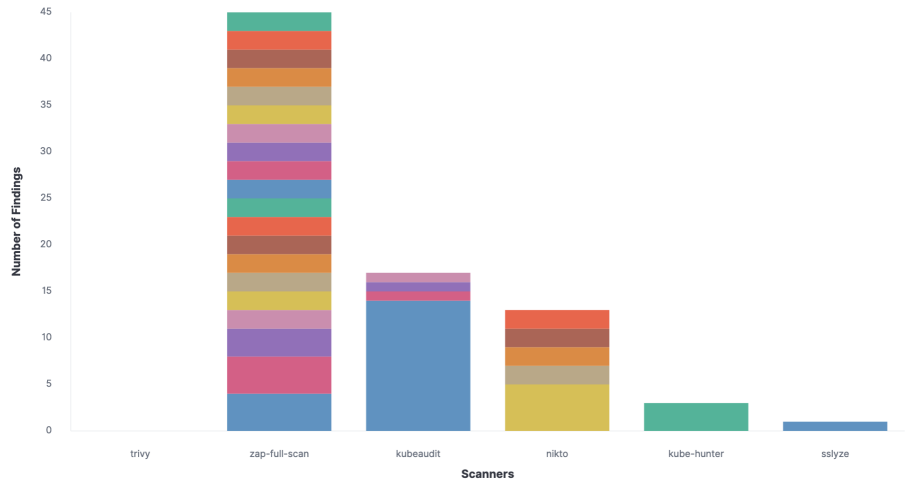

Figure **5**-8: All findings identified by the scans created by the auto-discovery for the BodgeIt Store, excluding container image vulnerability findings. Colored stripes indicating different finding categories.

OWASP ZAP Findings

- SQLi in the basket page
- Integer Overflow Error in the basket page
- Multiple missing CSRF tokens
- Reflected XSS in the `search` page
- Buffer Overflow in the basket page

ZAP was able to find a good number of findings for the BodgeIt Store, with some of high severity.

Trivy Findings:

Trviy did identify 334 findings for the BodgeIt image. In contrast to the Trivy results for Juice Shop, all of these are coming from OS level packages. The high number of findings can be explained by the age of the BodgeIt container image and the dependencies

contained in it. An overview of the vulnerable packages identified by Trivy broken-down by the package containing the vulnerability can be found in fig. **5**-9.

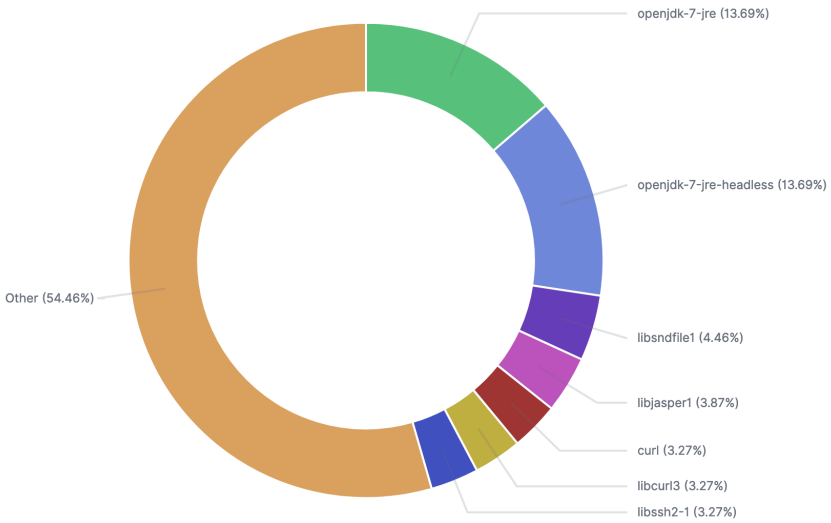

Figure **5**-9: Package names of the vulnerable packages in the BodgeIt container image identified by trivy

Notable findings of the trivy scan include several vulnerabilities for the deprecated version of java (`openjdk-7`) used in the image.[148]

Nikto Findings

- Correctly identified the web server as apache, including a matching version range.

In contrast to the Nikto results for Juice Shop, Nikto was able to correctly identify the webserver used and did not produce false-positive findings.

SSLyze Findings

Same as the findings for Juice Shop, as the certificates are handled by the cluster-wide ingress-controller. See section 5.2.2

Kube-hunter Findings

Same as the findings for Juice Shop, as the configuration of both deployments does not contain any security settings. See section 5.2.2

Kubeaudit Findings

Same as the findings for Juice Shop, as the configuration of both deployments does not contain any security settings. See section 5.2.2

5.3.3 Problems while Scanning BodgeIt Store

Outdated Image Manifest of the official Container Image

Initially, the trivy scans for BodgeIt have failed, because the official container image[87] of the BodgeIt Store uses an outdated container image manifest version, which is not compatible with trivy. To work around these issues the container image was rebuilt to use the newer manifest version. The rebuild image was publicly published to the private Docker Hub account of the author.[86] This has likely lowered the number of identified findings for the container image, as the original BodgeIt image was originally published in September of 2016[87], while the rebuild version was build based on the `tomcat:8.0` base image published in September of 2018[88], which gives the rebuild image around two years of vulnerability fixes the official image doesn't have.

5.4 Application Lifecycle Tracking

The auto-discovery prototype will automatically dispatch a new scan when the container images for a deployed application is updated. In the prototype, not all Resources are supported by this, as some of the Resources make it hard to track if the underlying container images have been changed. E.g. Ingress Resources don't directly reference Pods but point to a Service which then references a set of Pods. This indirection makes the lifecycle tracking hard, as the auto-discovery has to follow this multi-leveled chain to detect changes to the underlying container images. This is possible to implement but was skipped in this work for time reasons. Supported Kubernetes Resources for the lifecycle tracking in the prototype developed for this work are Pod and Service Resources.

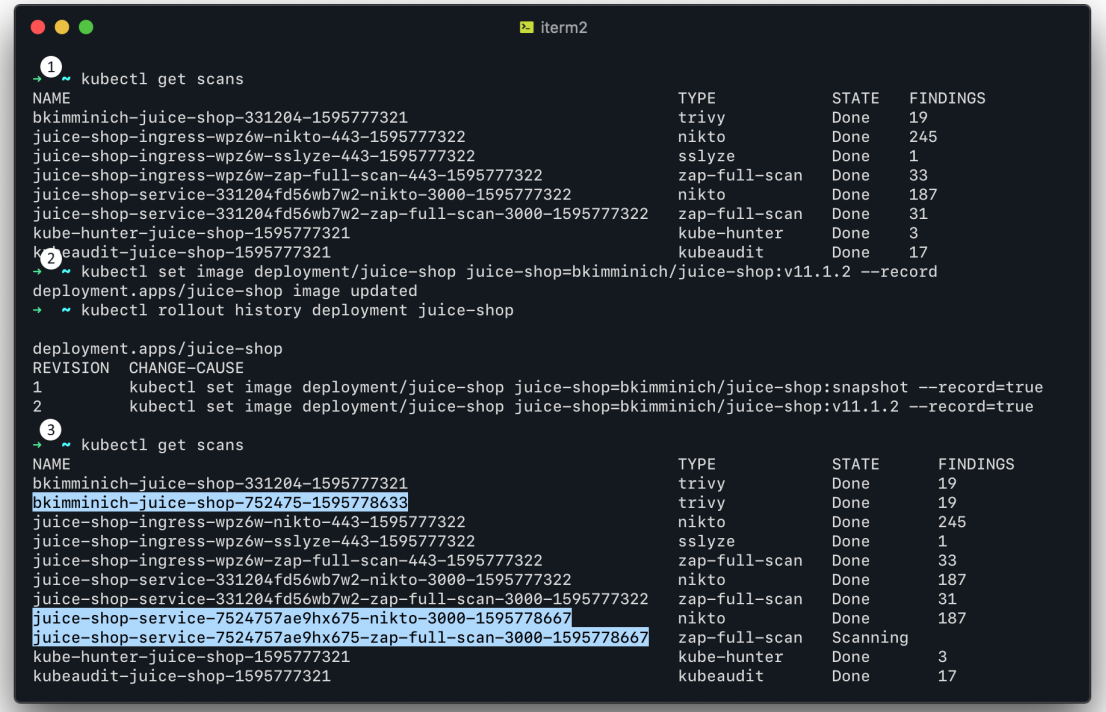

Figure **5**-10: Demonstration of Lifecycle Tracking for updated Deployments

An example of this lifecycle tracking can be seen in fig. **5**-10. In this example, Juice Shop is deployed and scanned like described in section 5.2, except that it is initially using the `:snapshot` container image tag of the Juice Shop container image.[85] The scans for this version can be seen in Step 1 of the figure. The Juice Shop Deployment is then updated to use the `:v11.1.2` tag of the image in Step 2. In Step 3, executed

about one minute after updating the container image, the updated list of scans created for Juice Shops is shown. Highlighted in light blue are the newly added scans for the Pod and the Service. The auto-discovery prototype has created a new Trivy scan for the new image version and a new ZAP and Nikto scan for the updated revision.

5.5 Prototype Result Summary

The scan results of the auto-discovery prototype have discovered major security defects in both applications. The probably most severe weaknesses discovered for both applications were the SQLi and XSS vulnerabilities discovered for both applications. If these results would have been found in a real-world setting, these vulnerabilities together with the fact that these were discovered in public internet-facing Ingress Resources would pose strong indicators that these applications need to be patched as soon as possible and might even have to be turned off until such patches are applied.

Other findings, e.g. a large number of vulnerabilities in the BodgeIt container image or the defects discovered by kubeaudit in the security configuration of the deployments, would give the application and security team of a company actionable indicators on how the security stance of the individual application can be improved.

One potentially interesting field, whose effectiveness is hard to test in syntactical test, is the possibilities of an auto-discovery system to find potentially forgotten services, like the case of the Kubernetes Dashboard at Tesla described in section 3.1. An auto-discovery system would be able to discover these services and make them visible to developers and security engineers.

Even though the findings discovered by the auto-discovery pose a good security baseline, not nearly all vulnerabilities in these applications were discovered, including other SQLi, XSS, and RCE weaknesses. The results of these scans can be used by security teams to prioritize obvious vulnerable applications and inform decisions for which applications manual penetration tests should be executed.

6 Conclusion

6.1 Summary

The Kubernetes API can be used to list all Resources in Kubernetes clusters. Applications in Kubernetes are comprised of a set of individual Resources, each of the types of Resources fulfills a set of responsibilities e.g. Services are used for networking, Pods are used to run the containerized applications. Some security aspects can be directly identified just on basis of these definitions, e.g. Kubernetes specific security miss-configurations. For a deeper investigation into the security aspects of a Resource, specialized scans (SAST and DAST) can then be dispatched to assess their security aspects, e.g. OWASP ZAP to find web-based vulnerabilities or Trivy to find known vulnerabilities in the deployed container images.

Research Question: How can the information from the Kubernetes API be used to automatically assess security aspects of applications running inside a Kubernetes cluster?

This work has shown, that in addition to using the Kubernetes API to list the applications, it can also be used to continuously track the lifecycle of applications and automatically repeat individual scans for changed Resources. This can be used to both immediately discover new applications deployed to Kubernetes, as well as track updated to existing applications.

The concepts described above were implemented in a prototype that can run inside the Kubernetes cluster to automatically create the above-mentioned scans for workloads it has discovered inside the cluster and repeat the scans when the Resources have been updated.

This prototype was evaluated inside a demonstration cluster to scan two intentionally vulnerable applications. During the evaluation, it was shown that the auto-discovery was able to discover the applications and create matching scans for the different types of Kubernetes Resources that make up the applications. The scans were able to detect multiple high severity findings in the applications including SQLi and XSS vulnerabilities. Even though major vulnerabilities were found by the scans, the applications used in the evaluation contained many more vulnerabilities, which the scans did not identify. This has shown that such an auto-discovery concept can be used to establish a security baseline and help application and security teams to make informed decisions by uncovering obvious vulnerabilities in applications, but it cannot guarantee that the scanned applications are secure.

To see how these concepts used here for Kubernetes can be used for the broader landscape of cloud computing, Kubernetes as a system was introduced and classified as a PaaS system. As such it is assumed that the results of this work can be projected onto similar environments. How far the results can be projected on these environments doesn't only depend on the service model of the environment (IaaS, PaaS, and SaaS) but is also heavily influenced by the possibilities provided by the API of the individual environments.

6.2 Future Work

Finding Analysis

During the evaluation of the prototype, several findings were discovered for the two demo applications. Not all of these findings were valid, some were falsely identified as venerabilities (false-positive findings). For the auto-discovery concepts to be applied in a useful manner against clusters of a real organization, the system would need to have a way to enable users to mark and then automatically identify these findings as false-positives. Without such a mechanism other valid findings might be overshadowed by false-positives.

In a similar vein, it was discovered that some of the open-source tools have overlapping foci, where they would produce findings for the vulnerabilities. Ideally, these findings would be grouped as one finding to avoid confusion for the users.

Auto-Discovery for other Cloud Environments

The concepts used in this work to build an auto-discovery system to automatically assess vulnerabilities in applications running on Kubernetes can be applied to other cloud environments. Some concepts, e.g. the selection of scanners for web/HTTP based services can be applied directly.

Other concepts applied in this prototype, like the reconciliation loop/controller pattern used to connect to the Kubernetes API, might not be applicable for other cloud-based systems as these patterns are not supported by their API. This might limit the possibilities for auto-discovery systems in other cloud systems to provide proper application lifecycle tracking or limit these to only work on a periodic schedule without directly scans on changes. Depending on the platform other approaches might work, some CSPs allow users to attach Publish / Subscribe (Pub/Sub) event queues to subscribe to certain security-relevant events or to register serverless functions to be executed on these events.[103, 58]

As described in chapter chapter 2, Kubernetes can be best classified as a PaaS system. As this work has only investigated the possibility of an auto-discovery system, it might not be possible to apply the learnings directly to other environments using a different Service Model (IaaS, SaaS).

Comparison against Policy based Security Models

The approach to ensure a baseline for cloud-based services used in this work is to assess Resources in Kubernetes during their runtime, which means that they already have been created. Another approach to improve the security of an environment is to define policies, which then limit the user's abilities to deploy insecure applications to an environment. This ensures that inherently insecure Resources are never deployed to the environment. This is great for security but is limiting to the users as it reduces the benefits of cloud computing (e.g. `On-demand self-service`, see NIST SP 800-145).

The best security program likely consists of a healthy mix between both approaches, to ensure that obvious miss-configurations are forbidden while still allowing valid deployments to be deployed in self-service. Where this line between policy and testing should be drawn depends on the security requirements of an organization. Highly regulated

industries might require policy-based approaches by law. A potential future research topic would be to establish a model that balances this line between policy and testing for different organizations with varying security requirements.

Comparison against Security Automation in CI Pipelines

As mentioned in section 4.2.1, the approach was taken by this work is not the only way to automate parts of the security assessment for applications. Most other approaches found in the literature base on integrating security testing tools, as mentioned in this work, into the CI pipelines of software projects. How these approaches differ, which benefits the individual approaches offer might be an interesting field for future study.

Acronyms

AKS Azure Kubernetes Service. 11

ARP Address Resolution Protocol. 53

AWS Amazon Web Services. 9, 11, 24, 32

CI Continious Integration. 17, 31, 44, 65

CNCF Cloud Native Computing Foundation. 11, 17, 23

CNI Container Networking Interface. 19

CORS Cross-origin resource sharing. 52

CRD Custom Resource Definition. 35

CRI Container Runtime Interface. 19, 26, 27, 47

CSP Cloud Service Provider. 9, 10, 18, 19, 21, 22, 30, 32, 37, 51, 64

CSRF Cross-Site Request Forgery. 52, 57

DAST Dynamic Application Security Testing. 32, 38, 40, 43, 62

EKS Amazon Elastic Kubernetes Service. 11

FaaS Function as a Service. 16

Acronyms

GCP Google Cloud Plattform. 9, 11

GKE Google Kubernetes Engine. 11

IaaS Infrastructure as a Service. 9, 14, 15, 63, 64

JSP Java Server Pages. 49

JWT JSON Web Token. 52

NIST National Institute of Standards and Technology. 6

NIST SP 800-145 National Institute of Standards and Technologies Special Publication 800-145. 8, 9, 13–15, 64

OS Operating System. 25, 43, 57

OWASP Open Web Application Security Project. 12, 24, 29, 35, 40

PaaS Platform as a Service. 9, 15, 16, 21, 25, 63, 64

Pub/Sub Publish / Subscribe. 64

RBAC Role-based Access Control. 22, 28, 53

RCE Remote Code Execution. 24, 27, 61

SaaS Software as a Service. 9, 63, 64

SAST Static Application Security Testing. 43, 62

SPA Single Page Application. 48, 54

SQLi SQL Injection. 6, 24, 41, 51, 57, 61, 63

Acronyms

VM Virtual Machine. 9, 26

WAF Web Application Firewall. 39

XSS Cross Site Scripting. 6, 41, 57, 61, 63

Bibliography

[1] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: https://agilemanifesto.org/ (visited on 07/23/2020).

[2] Gordon Lyon. *Nmap Network Scanning*. insecure.com LLC, 2009. ISBN: 978-0-9799587-1-7.

[3] George Feuerlicht, Lukas Burkon, and Michal Sebesta. "Cloud Computing Adoption: What are the Issues?" In: *System Integration* 18.2 (2011), pp. 187–192.

[4] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology, Sept. 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (visited on 06/24/2020).

[5] Vic (J.R.) Winkler. *Securing the Cloud: Cloud computer Security techniques and tactics*. Elsevier Inc., 2011. ISBN: 978-1-59749-592-9.

[6] Zakir Durumeric et al. "The Matter of Heartbleed". In: *Proceedings of the 2014 conference on internet measurement conference*. 2014, pp. 475–488.

[7] M.A.C. Dekker and Dimitra Liveri. "Cloud Security Guide for SMEs". In: *European Union Agency for Network and Information Security* (2015). URL: https://www.enisa.europa.eu/publications/cloud-security-guide-for-smes/at_download/fullReport (visited on 06/24/2020).

[8] *New Cloud Native Computing Foundation to Drive Alignment Among Container Technologies*. May 2015. URL: https://www.cncf.io/announcement/2015/06/21/new-cloud-native-computing-foundation-to-drive-alignment-among-container-technologies/ (visited on 06/18/2020).

[9] Abhishek Verma et al. "Large-scale cluster management at Google with Borg". In: *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France, 2015.

[10] Brendan Burns et al. "Borg, Omega, and Kubernetes". In: *Queue* 14.1 (2016), pp. 70–93.

[11] Gene Kim et al. *The DevOps Handbook*. IT Revolution Press, Oct. 2016. ISBN: 978-1-94278-800-3.

[12] OWASP Foundation & Contributors. *OWASP Top Ten 2017*. Open Web Application Security Project, 2017. URL: https://owasp.org/www-project-top-ten/ (visited on 06/30/2020).

[13] *Kubernetes Release - 1.6.0*. Mar. 2017. URL: https://github.com/kubernetes/kubernetes/releases/tag/v1.6.0 (visited on 07/09/2020).

[14] *AWS Whitepaper - Tagging Best Practices*. Dec. 2018. URL: https://d1.awsstatic.com/whitepapers/aws-tagging-best-practices.pdf (visited on 07/29/2020).

[15] Joe Beda. *On Securing the Kubernetes Dashboard*. Feb. 2018. URL: https://blog.heptio.com/on-securing-the-kubernetes-dashboard-16b09b1b7aca (visited on 05/16/2020).

[16] Brendan Burns and Craig Tracey. *Managing Kubernetes*. O'Reilly Media, Inc., 2018. ISBN: 978-1-49203-391-2.

[17] Sarah Conway. *Kubernetes Is First CNCF Project To Graduate*. Mar. 2018. URL: https://www.cncf.io/blog/2018/03/06/kubernetes-first-cncf-project-graduate/ (visited on 06/20/2020).

[18] Dan Goodin. *Tesla cloud resources are hacked to run cryptocurrency-mining malware*. Feb. 2018. URL: https://arstechnica.com/information-technology/2018/02/tesla-cloud-resources-are-hacked-to-run-cryptocurrency-mining-malware/ (visited on 05/16/2020).

[19] RedLock Inc. *Lessons from the Cryptojacking Attack at Tesla*. Feb. 2018. URL: https://redlock.io/blog/cryptojacking-tesla (visited on 07/09/2020).

[20] Zane Lackey and Rebecca Huehls. *Building a Modern Security Program*. O'Reilly Media, Inc., Aug. 2018. ISBN: 978-1-49195-631-1.

[21] Liz Rice and Michael Hausenblas. *Kubernetes Security*. O'Reilly Media, Inc., Nov. 2018. ISBN: 978-1-49203-906-8.

[22] Peter Balogh. *DAST Operator - Dynamic application security testing in Kubernetes*. Sept. 2019. URL: https://banzaicloud.com/blog/auto-dast/ (visited on 07/27/2020).

[23] Brendan Burns, Joe Beda, and Kelsey Hightower. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. Second Edition. O'Reilly Media, Inc., 2019.

[24] Brendan Burns et al. *Kubernetes Best Practices*. O'Reilly Media, Inc., Nov. 2019. ISBN: 978-1-49205-647-8.

[25] Chris Dotson. *Practical Cloud Security*. O'Reilly Media, Inc., Mar. 2019. ISBN: 978-1-49203-751-4.

[26] Michael Hausenblas and Stefan Schimanski. *Programming Kubernetes: Developing Cloud-Native Applications*. O'Reilly Media, Inc., 2019.

[27] Tony Hsiang-Chih Hsu. *Practical Security Automation and Testing: Tools and techniques for automated security scanning and testing in DevSecOps*. Packt Publishing Ltd., Feb. 2019. ISBN: 978-1-78980-202-3.

[28] Bilgin Ibryam and Roland Huß. *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*. O'Reilly Media, Inc., 2019.

[29] Sysdig Inc. *2019 Container Usage Report*. 2019. URL: https://sysdig.com/blog/sysdig-2019-container-usage-report/ (visited on 06/27/2020).

[30] Simon Bennetts. *Is ZAP the World's most Popular Web Scanner?* Apr. 2020. URL: https://www.zaproxy.org/blog/2020-04-02-is-zap-the-worlds-most-popular-web-scanner/ (visited on 07/07/2020).

[31] Simon Bennetts. *Tweet - Zap is the Most Popular Web Scanner*. Apr. 2020. URL: https://twitter.com/psiinon/status/1250034318530969607 (visited on 07/07/2020).

[32] Simon Bennetts. *ZAP Google Groups - ZAP High Level Plans*. May 2020. URL: https://groups.google.com/g/zaproxy-develop/c/t997yIGcSd4 (visited on 07/14/2020).

[33] OWASP Foundation & Contributors. *OWASP Cheat Sheet Series*. Open Web Application Security Project, 2020. URL: https://owasp.org/www-project-cheat-sheets/ (visited on 07/06/2020).

[34] OWASP Foundation & Contributors. *OWASP Cheat Sheet Series - Cross Site Scripting Prevention - X-XSS-Protection Header*. Open Web Application Security Project, 2020. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#x-xss-protection-header (visited on 07/21/2020).

[35] OWASP Foundation & Contributors. *OWASP Cheat Sheet Series - Docker Security Cheat Sheet*. Open Web Application Security Project, 2020. URL: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html (visited on 07/16/2020).

[36] OWASP Foundation & Contributors. *OWASP Web Security Testing Guide Version 4.1*. Open Web Application Security Project, Apr. 2020.

[37] Jannik Hollenbach. *BodgeIt Store Issues - Docker Image is using a deprecated Docker Schema*. July 2020. URL: https://github.com/psiinon/bodgeit/issues/26 (visited on 07/15/2020).

[38] Jannik Hollenbach. *OWASP Juice Shop - Release v11.1.2*. July 2020. URL: https://github.com/bkimminich/juice-shop/releases/tag/v11.1.2 (visited on 07/14/2020).

[39] Center for Internet Security Inc. *CIS Kubernetes Benchmark (v1.5.1 - 02-14-2020)*. Feb. 2020. URL: https://www.cisecurity.org/benchmark/kubernetes/ (visited on 07/02/2020).

[40] Björn Kimminich. *OWASP Juice Shop - Commit b0ec8f3*. Apr. 2020. URL: https://github.com/bkimminich/juice-shop/commit/b0ec8f3 (visited on 07/14/2020).

[41] Björn Kimminich. *OWASP Juice Shop - Release v10.2.1*. Apr. 2020. URL: https://github.com/bkimminich/juice-shop/releases/tag/v10.2.1 (visited on 07/14/2020).

[42] Liz Rice. *Container Security*. O'Reilly Media, Inc., Apr. 2020. ISBN: 978-1-49205-670-6.

[43] Liz Rice. *Starboard: The Kubernetes-Native Toolkit for Unifying Security*. June 2020. URL: https://blog.aquasec.com/starboard-kubernetes-tools (visited on 07/27/2020).

[44] Scott Rose et al. *Zero Trust Architecture (Draft 2)*. National Institute of Standards and Technology, Feb. 2020. URL: https://csrc.nist.gov/publications/detail/sp/800-207/draft (visited on 07/02/2020).

[45] Ariel Shuper. *Kubei : A Kubernetes Runtime Vulnerabilities Scanner*. Mar. 2020. URL: https://www.portshift.io/blog/kubernetes-runtime-vulnerabilities-scanner-launch/ (visited on 07/27/2020).

[46] Yossi Weizman. *Threat matrix for Kubernetes*. Apr. 2020. URL: https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/ (visited on 06/07/2020).

[47] *"Distroless" Docker Images*. URL: https://github.com/GoogleContainerTools/distroless (visited on 07/03/2020).

[48] *Alpine Linux - Alpine Linux is a security-oriented, lightweight Linux distribution based on musl libc and busybox.* URL: https://www.alpinelinux.org/ (visited on 07/21/2020).

[49] *Amazon AWS Data Center Security*. URL: https://aws.amazon.com/compliance/data-center/perimeter-layer/ (visited on 06/07/2020).

[50] *Amazon EKS on AWS Outposts*. URL: https://docs.aws.amazon.com/eks/latest/userguide/eks-on-outposts.html (visited on 06/24/2020).

[51] *Amazon Elastic Kubernetes Service*. URL: https://aws.amazon.com/eks/ (visited on 07/28/2020).

[52] *Amazon Web Services*. URL: https://aws.amazon.com/ (visited on 07/20/2020).

[53] *Anchore Container Analysis*. URL: https://anchore.com/ (visited on 07/15/2020).

[54] *App Engine - Fully managed serverless application platform*. URL: https://cloud.google.com/appengine/ (visited on 06/20/2020).

[55] *Aqua Security*. URL: https://www.aquasec.com/ (visited on 07/21/2020).

[56] *AWS Ground Station*. URL: https://aws.amazon.com/ground-station/ (visited on 06/06/2020).

[57] *AWS Inspector*. URL: https://aws.amazon.com/inspector/ (visited on 07/19/2020).

[58] *AWS Lambda*. URL: https://aws.amazon.com/lambda/ (visited on 06/06/2020).

[59] *Azure Blog - Detect large-scale cryptocurrency mining attack against Kubernetes clusters*. URL: https://azure.microsoft.com/en-gb/blog/detect-largescale-cryptocurrency-mining-attack-against-kubernetes-clusters/ (visited on 07/22/2020).

[60] *Azure Documentation - Tag support for Azure resources*. URL: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/tag-support (visited on 07/29/2020).

[61] *Azure Functions*. URL: https://azure.microsoft.com/en-us/services/functions/ (visited on 06/06/2020).

[62] *Azure Hybrid Cloud*. URL: https://azure.microsoft.com/en-us/overview/azure-hybrid/ (visited on 06/24/2020).

[63] *Azure Kubernetes Service (AKS)*. URL: https://azure.microsoft.com/en-us/services/kubernetes-service/ (visited on 07/28/2020).

[64] *Azure Security Center*. URL: https://azure.microsoft.com/en-us/services/security-center/ (visited on 07/19/2020).

[65] *Banzai Cloud*. URL: https://banzaicloud.com/ (visited on 07/21/2020).

[66] Simon Bennetts. “OWASP ZAP Intro”. In: OWASP Hamburg Stammtisch (Apr. 23, 2020). URL: https://www.youtube.com/watch?v=SD28HdVI-Wk (visited on 07/08/2020).

[67] Simon Bennetts and Mark Miller. *ZAP in Ten*. URL: https://www.alldaydevops.com/zap-in-ten (visited on 06/27/2020).

[68] *BeyondProd: A new approach to cloud-native security*. URL: https://cloud.google.com/security/beyondprod/ (visited on 07/02/2020).

[69] *BodgeIt Store - Commit History*. URL: https://github.com/psiinon/bodgeit/commits/master (visited on 07/15/2020).

[70] *Cert Manager - Automatically provision and manage TLS certificates in Kubernetes*. URL: https://github.com/jetstack/cert-manager (visited on 07/14/2020).

[71] *Clair - Vulnerability Static Analysis for Containers*. URL: https://github.com/quay/clair (visited on 06/30/2020).

[72] *Cloud Foundry Open Source Cloud Application Platform*. URL: https://www.cloudfoundry.org/ (visited on 06/20/2020).

[73] *Cloud Native Buildpack*. URL: https://buildpacks.io/ (visited on 06/20/2020).

[74] *Cloud Native Computing Foundation*. URL: https://www.cncf.io/ (visited on 07/20/2020).

[75] *CloudSploit - Cloud Security Best Practices as a Service*. URL: https://cloudsploit.com/ (visited on 07/21/2020).

[76] *CNCF Landscape*. URL: https://landscape.cncf.io/ (visited on 07/09/2020).

[77] *CNCF Landscape Certified Kubernetes - Hosted*. URL: https://landscape.cncf.io/category=certified-kubernetes-hosted&format=card-mode&grouping=category (visited on 06/21/2020).

[78] *CNCF Members*. URL: https://www.cncf.io/about/members/ (visited on 06/21/2020).

[79] Ian Coldwater and Brad Geesaman. "Advanced Persistence Threats: The Future of Kubernetes Attacks". In: RSA Conference 2020 San Francisco (Feb. 28, 2020). URL: https://www.rsaconference.com/usa/agenda/advanced-persistence-threats-the-future-of-kubernetes-attacks (visited on 05/16/2020).

[80] *containerd - An industry-standard container runtime with an emphasis on simplicity, robustness and portability*. URL: https://containerd.io/ (visited on 06/27/2020).

[81] *CoreDNS: DNS and Service Discovery*. URL: https://coredns.io/ (visited on 06/24/2020).

[82] *DAST Operator - Dynamic Application Security Testing*. URL: https://github.com/banzaicloud/dast-operator (visited on 07/19/2020).

[83] *DigitalOcean*. URL: https://www.digitalocean.com/ (visited on 07/21/2020).

[84] *Docker*. URL: https://www.docker.com/ (visited on 06/27/2020).

[85] *Docker Hub - OWASP Juice Shop*. URL: https://hub.docker.com/r/bkimminich/juice-shop (visited on 06/26/2020).

[86] *Docker Hub - Rebuild -The BodgeIt Store*. URL: https://hub.docker.com/r/j12934/bodgeit (visited on 07/15/2020).

[87] *Docker Hub - The BodgeIt Store*. URL: https://hub.docker.com/r/psiinon/bodgeit/ (visited on 07/15/2020).

[88] *Docker Hub - tomcat:8.0*. URL: https://hub.docker.com/layers/tomcat/library/tomcat/8.0/images/sha256-3c45e165dc72e3fc0f147dfa0c4712145cde00c2efc7 (visited on 07/21/2020).

[89] *Docker Windows Containers*. URL: https://www.docker.com/products/windows-containers (visited on 07/31/2020).

[90] *Elastic Stack*. URL: https://www.elastic.co/products/elastic-stack (visited on 12/16/2019).

[91] Phil Estes. “Let’s Try All the CRI Runtimes: Part 2: Answering the Why Question!” In: KubeCon + CloudNativeCon North America 2019 (Nov. 20, 2019). URL: https://kccncna19.sched.com/event/Uaag (visited on 06/27/2020).

[92] Phil Estes. “Let’s Try Every CRI Runtime Available for Kubernetes. No, Really!” In: KubeCon + CloudNativeCon Europe 2019 (Mar. 23, 2019). URL: https://kccnceu19.sched.com/event/MPdB (visited on 06/27/2020).

[93] *Firecracker - Secure and fast microVMs for serverless computing*. URL: https://firecracker-microvm.github.io/ (visited on 06/27/2020).

[94] *GCP Documentation - Labeling resources*. URL: https://cloud.google.com/compute/docs/labeling-resources (visited on 07/29/2020).

[95] *GKE - Security Overview*. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/security-overview (visited on 07/06/2020).

[96] *Google Anthos GKE*. URL: https://cloud.google.com/anthos/gke/ (visited on 06/24/2020).

[97] *Google Cloud Functions*. URL: https://cloud.google.com/functions/ (visited on 06/06/2020).

[98] *Google Cloud Plattform*. URL: https://cloud.google.com/ (visited on 07/20/2020).

[99] *Google Container Registry - Vulnerability scanning*. URL: https://cloud.google.com/container-registry/docs/vulnerability-scanning (visited on 07/21/2020).

[100] *Google Data Center Security*. URL: https://www.google.com/about/datacenters/data-security/ (visited on 06/07/2020).

[101] *Google G Suite*. URL: https://gsuite.google.com/ (visited on 06/06/2020).

[102] *Google Kubernetes Engine*. URL: https://cloud.google.com/kubernetes-engine/ (visited on 07/28/2020).

[103] *Google Security Command Center*. URL: https://cloud.google.com/security-command-center (visited on 07/19/2020).

[104] *gVisor - gVisor is an application kernel for containers that provides efficient defense-in-depth anywhere*. URL: https://gvisor.dev/ (visited on 06/27/2020).

[105] *Harbor Blog - Harbor 1.10 Puts Security and Pluggable Scanners in the Limelight*. URL: https://goharbor.io/blog/harbor-1.10-release/ (visited on 07/21/2020).

[106] *Harbor Documentation - Tag Immutability Rules*. URL: https://goharbor.io/docs/1.10/working-with-projects/working-with-images/create-tag-immutability-rules/ (visited on 07/15/2020).

[107] *Helm Documentation - RBAC*. URL: https://helm.sh/docs/topics/rbac/ (visited on 07/09/2020).

[108] *Heroku - Cloud Application Platform*. URL: https://www.heroku.com/ (visited on 06/20/2020).

[109] Kelsey Hightower. “Kubernetes and the Path to Serverless”. In: KubeCon + CloudNativeCon North America 2018 (Dec. 13, 2020). URL: https://kccna18.sched.com/event/Gsy5 (visited on 06/19/2020).

[110] *Istio - Connect, secure, control, and observe services.* URL: https://istio.io/ (visited on 06/30/2020).

[111] *k3s - Lightweight Kubernetes*. URL: https://k3s.io/ (visited on 06/23/2020).

[112] *Kata Containers - The speed of containers, the security of VMs*. URL: https://katacontainers.io/ (visited on 06/27/2020).

[113] *kube-hunter - Hunt for security weaknesses in Kubernetes clusters*. URL: https://github.com/aquasecurity/kube-hunter (visited on 07/21/2020).

[114] *Kubeaudit - kubeaudit helps you audit your Kubernetes clusters against common security controls*. URL: https://github.com/Shopify/kubeaudit/releases (visited on 07/23/2020).

[115] *Kubebuilder - SDK for building Kubernetes APIs using CRDs*. URL: https://book.kubebuilder.io/ (visited on 07/18/2020).

[116] *Kubei - A flexible Kubernetes Runtime Scanner*. URL: https://github.com/Portshift/kubei (visited on 07/19/2020).

[117] *Kubernetes API Reference - v1.18*. URL: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/ (visited on 07/18/2020).

[118] *Kubernetes Cluster Autoscaler*. URL: https://github.com/kubernetes/autoscaler (visited on 06/24/2020).

[119] *Kubernetes Documentation - Cloud Controller Manager*. URL: https://kubernetes.io/docs/concepts/architecture/cloud-controller/ (visited on 07/07/2020).

[120] *Kubernetes Documentation - Configure a Security Context for a Pod or Container*. URL: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ (visited on 07/09/2020).

[121] *Kubernetes Documentation - Configure Service Accounts for Pods*. URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/ (visited on 07/16/2020).

[122] *Kubernetes Documentation - Horizontal Pod Autoscaler*. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (visited on 11/18/2019).

[123] *Kubernetes Documentation - Ingress*. URL: https://kubernetes.io/docs/concepts/services-networking/ingress/ (visited on 07/07/2020).

[124] *Kubernetes Documentation - Kubernetes Components*. URL: https://kubernetes.io/docs/concepts/overview/components/ (visited on 06/20/2020).

[125] *Kubernetes Documentation - Kubernetes Security and Disclosure Information*. URL: https://kubernetes.io/docs/reference/issues-security/security/ (visited on 07/09/2020).

[126] *Kubernetes Documentation - Kubernetes Version and Version Skew Support Policy*. URL: https://kubernetes.io/docs/setup/release/version-skew-policy/ (visited on 07/09/2020).

[127] *Kubernetes Documentation - Network Plugins*. URL: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/ (visited on 06/23/2020).

[128] *Kubernetes Documentation - Network Policies*. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/ (visited on 06/24/2020).

[129] *Kubernetes Documentation - Pod Security Policies*. URL: https://kubernetes.io/docs/concepts/policy/pod-security-policy/ (visited on 06/30/2020).

[130] *Kubernetes Documentation - Pods*. URL: https://kubernetes.io/docs/concepts/workloads/pods/pod/ (visited on 07/15/2020).

[131] *Kubernetes Documentation - Resource Quotas*. URL: https://kubernetes.io/docs/concepts/policy/resource-quotas/ (visited on 06/24/2020).

[132] *Kubernetes Documentation - WebUI (Dashboard)*. URL: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/ (visited on 07/09/2020).

[133] *Kubernetes Documentation - What is Kubernetes?* URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (visited on 05/14/2020).

[134] *Kubernetes Documentation - Workloads*. URL: https://kubernetes.io/docs/concepts/workloads/controllers/ (visited on 07/15/2020).

[135] *Kubernetes Issue Tracer - CVE's*. URL: https://github.com/kubernetes/kubernetes/issues?q=is:issue%20label:area/security%20in:title%20CVE (visited on 06/24/2020).

[136] *Kubernetes Metrics Server*. URL: https://github.com/kubernetes-sigs/metrics-server (visited on 06/24/2020).

[137] Zane Lackey. "DevSecOps Lessons Learned with Zane Lackey". In: Spotlight on Cloud (Oct. 16, 2019). O'Reilly Media, Inc. URL: https://learning.oreilly.com/live-training/courses/spotlight-on-cloud-devsecops-lessons-learned-with-zane-lackey/0636920305804/ (visited on 07/23/2020).

[138] *Let's Encrypt - A nonprofit Certificate Authority providing TLS certificates to 225 million websites*. URL: https://letsencrypt.org/ (visited on 07/14/2020).

[139] *Linkerd - Ultralight, security-first service mesh for Kubernetes*. URL: https://linkerd.io/ (visited on 06/30/2020).

[140] *MDN Web Docs - X-XSS-Protection*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection (visited on 07/21/2020).

[141] *Microsoft Azure*. URL: https://azure.microsoft.com/ (visited on 07/20/2020).

[142] *Microsoft Office365*. URL: https://www.microsoft.com/en-us/microsoft-365 (visited on 06/06/2020).

[143] *NGINX Ingress Controller*. URL: https://kubernetes.github.io/ingress-nginx/ (visited on 07/07/2020).

[144] *NGINX Ingress Controller - Documentation ModSecurity*. URL: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#modsecurity (visited on 07/13/2020).

[145] *Nikto Documentation - Introduction*. URL: https://cirt.net/nikto2-docs/introduction.html (visited on 07/08/2020).

[146] *OpenFaaS Main Website*. URL: https://www.openfaas.com/ (visited on 10/05/2019).

[147] *OpenStack*. URL: https://www.openstack.org/ (visited on 07/29/2020).

[148] *Oracle Java Release Cycle*. URL: https://www.oracle.com/java/technologies/java-se-support-roadmap.html (visited on 07/21/2020).

[149] *OWASP Juice Shop*. URL: https://owasp.org/www-project-juice-shop/ (visited on 06/26/2020).

[150] *OWASP Juice Shop - Contributors*. URL: https://github.com/bkimminich/juice-shop#contributors (visited on 07/15/2020).

[151] *OWASP ZAP*. URL: https://github.com/zaproxy/zaproxy (visited on 12/09/2019).

[152] *Project Calico*. URL: https://www.projectcalico.org/ (visited on 06/23/2020).

[153] *Red Hat*. URL: https://www.redhat.com/ (visited on 07/27/2020).

[154] *Red Hat Quay*. URL: https://www.openshift.com/products/quay (visited on 07/21/2020).

[155] Liz Rice. "Lessons from hacking Kubernetes with kube-hunter". In: O'Reilly Velocity Conference 2019 (June 12, 2018). URL: https://conferences.oreilly.com/velocity/vl-ca/public/schedule/detail/74861 (visited on 07/21/2020).

[156] Liz Rice. "Running with Scissors". In: KubeCon + CloudNativeCon Europe 2018 (May 4, 2018). URL: https://kccnceu18.sched.com/event/EMyr/keynote-running-with-scissors-liz-rice (visited on 07/03/2020).

[157] *secureCodeBox Docs - Elasticsearch Persistence Provider*. URL: https://www.securecodebox.io/integrations/persistence-provider/elasticsearch (visited on 07/23/2020).

[158] *secureCodeBox v2 Alpha - Repository*. URL: https://github.com/secureCodeBox/secureCodeBox-v2-alpha (visited on 07/18/2020).

[159] *secureCodeBox Website*. URL: https://www.securecodebox.io (visited on 11/21/2019).

[160] *Security concepts for applications and clusters in Azure Kubernetes Service (AKS)*. URL: https://docs.microsoft.com/en-us/azure/aks/concepts-security (visited on 07/06/2020).

[161] *Security in Amazon EKS*. URL: https://docs.aws.amazon.com/eks/latest/userguide/security.html (visited on 07/06/2020).

[162] *Shopify - All-In-One Commerce Solution*. URL: https://www.shopify.com/ (visited on 07/23/2020).

[163] *SSLyze - Fast and Powerful SSL/TLS Scanning Library*. URL: https://github.com/nabla-c0d3/sslyze/ (visited on 07/13/2020).

[164] *Starboard - Kubernetes-native Security Tool Kit*. URL: https://github.com/aquasecurity/starboard (visited on 07/19/2020).

[165] *Starboard - Roadmap*. URL: https://github.com/aquasecurity/starboard/blob/master/ROADMAP.md (visited on 07/19/2020).

[166] *The BodgeIt Store*. URL: https://github.com/psiinon/bodgeit (visited on 07/15/2020).

[167] *The Go Programming Language*. URL: https://golang.org/ (visited on 07/22/2020).

[168] *Trivy - A Simple and Comprehensive Vulnerability Scanner for Containers, Suitable for CI*. URL: https://github.com/aquasecurity/trivy (visited on 06/30/2020).

[169] *Vercel - Develop. Preview. Ship.* URL: https://vercel.com/ (visited on 06/20/2020).

[170] *ZAP Documentation - AJAX Spider*. URL: https://www.zaproxy.org/docs/desktop/addons/ajax-spider/ (visited on 07/14/2020).

[171] *ZAP Documentation - API*. URL: https://www.zaproxy.org/docs/api/ (visited on 07/07/2020).

[172] *ZAP Documentation - Spider*. URL: https://www.zaproxy.org/docs/desktop/start/features/spider/ (visited on 07/14/2020).

[173] *ZAP Documentation - ZAPping the OWASP Top 10*. URL: https://www.zaproxy.org/docs/guides/zapping-the-top-10/ (visited on 07/08/2020).

[174] *Zero Trust*. URL: https://www.microsoft.com/en-us/security/business/zero-trust (visited on 07/02/2020).

List of Figures

A Appendix

A.1 Prototype Evaluation using OWASP Juice Shop

A.1.1 OWASP Juice Shop Deployment

```
apiVersion: v1
kind: Namespace
metadata:
  name: juice-shop
  annotations:
    auto-discovery.experimental.securecodebox.io/enabled: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: juice-shop
  namespace: juice-shop
  labels:
    app: juice-shop
spec:
  selector:
    matchLabels:
      app: juice-shop
  replicas: 3
  template:
    metadata:
      labels:
        app: juice-shop
    spec:
      containers:
        - name: juice-shop
          image: bkimminich/juice-shop:snapshot
          ports:
            - name: http
```

```
            containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: juice-shop
  namespace: juice-shop
  labels:
    invasive-scans.auto-discovery.experimental.securecodebox.io: "true"
    zap-hints.auto-discovery.experimental.securecodebox.io/spider: "ajax"
spec:
  type: ClusterIP
  ports:
    - port: 3000
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: juice-shop
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: juice-shop
  namespace: juice-shop
  labels:
    invasive-scans.auto-discovery.experimental.securecodebox.io: "true"
    zap-hints.auto-discovery.experimental.securecodebox.io/spider: "ajax"
  annotations:
    kubernetes.io/ingress.class: nginx
    # An annotation indicating the issuer to use.
    cert-manager.io/cluster-issuer: letsencrypt-production
spec:
  tls:
    - secretName: juice-shop-tls-secret
      hosts:
        - juice-shop.demo.securecodebox.io
  rules:
    - host: juice-shop.demo.securecodebox.io
      http:
        paths:
          - path: /
            backend:
              serviceName: juice-shop
```

```
    servicePort: 3000
```

Listing A.1: Kubernetes manifests used to deploy OWASP Juice Shop application during the prototype evaluation

A.1.2 Findings Identified for OWASP Juice Shop

All findings identified for Juice Shop: (excluding obvious Nikto false positive findings of the category `Potential Backup File` and with duplicate findings, produced by both scanning Ingress and Service Resources, filtered out)

Scanner	Severity	Name
zap-full-scan	INFORMATIONAL	.env Information Leak
zap-full-scan	INFORMATIONAL	Base64 Disclosure
zap-full-scan	INFORMATIONAL	Cookie Poisoning
zap-full-scan	INFORMATIONAL	Cookie Slack Detector
zap-full-scan	INFORMATIONAL	Information Disclosure - Suspicious Comments
zap-full-scan	INFORMATIONAL	Modern Web Application
zap-full-scan	INFORMATIONAL	Non-Storable Content
zap-full-scan	INFORMATIONAL	Storable and Cacheable Content
zap-full-scan	INFORMATIONAL	Storable but Non-Cacheable Content
zap-full-scan	INFORMATIONAL	Timestamp Disclosure - Unix
zap-full-scan	INFORMATIONAL	Trace.axd Information Leak
zap-full-scan	INFORMATIONAL	User Agent Fuzzer
zap-full-scan	MEDIUM	Hidden File Found
zap-full-scan	MEDIUM	.env Information Leak
zap-full-scan	MEDIUM	Apache Range Header DoS (CVE-2011-3192)
zap-full-scan	MEDIUM	Backup File Disclosure
zap-full-scan	MEDIUM	Cross-Domain Misconfiguration
zap-full-scan	MEDIUM	Session ID in URL Rewrite
zap-full-scan	MEDIUM	Source Code Disclosure - Java
zap-full-scan	MEDIUM	Trace.axd Information Leak
zap-full-scan	MEDIUM	X-Frame-Options Header Not Set
zap-full-scan	MEDIUM	HTTP Only Site
zap-full-scan	MEDIUM	Proxy Disclosure
zap-full-scan	LOW	Content Security Policy (CSP) Header Not Set

zap-full-scan	LOW	Cross-Domain JavaScript Source File Inclusion
zap-full-scan	LOW	Dangerous JS Functions
zap-full-scan	LOW	Feature Policy Header Not Set
zap-full-scan	LOW	Private IP Disclosure
zap-full-scan	LOW	X-Content-Type-Options Header Missing
zap-full-scan	LOW	Cookie Without Secure Flag
zap-full-scan	LOW	Incomplete or No Cache-control and Pragma HTTP Header Set
zap-full-scan	LOW	Server Leaks Version Information via "Server" HTTP Response Header Field
zap-full-scan	HIGH	SQL Injection - SQLite
zap-full-scan	HIGH	Cloud Metadata Potentially Exposed
trivy	HIGH	Allocation of Resources Without Limits or Throttling
trivy	HIGH	Verification Bypass
trivy	HIGH	nodejs-jsonwebtoken: verification step bypass with an altered token
trivy	HIGH	Forgeable Public/Private Tokens
trivy	HIGH	Out-of-bounds Read
trivy	HIGH	lodash: Prototype pollution in utilities function
trivy	HIGH	nodejs-lodash: prototype pollution in defaultsDeep function leading to modifying properties
trivy	MEDIUM	Authorization bypass in express-jwt
trivy	MEDIUM	Cross Site Scripting
trivy	MEDIUM	Moderate severity vulnerability that affects sanitize-html
trivy	MEDIUM	Prototype pollution in lodash
trivy	MEDIUM	XSS - Sanitization not applied recursively
trivy	MEDIUM	moment.js: regular expression denial of service
trivy	MEDIUM	nodejs-lodash: prototype pollution in zipObject-Deep function
trivy	MEDIUM	nodejs-moment: Regular expression denial of service
trivy	LOW	lodash: Prototype pollution in utilities function
kubeaudit	LOW	Capability 'AUDIT_WRITE' Not Dropped
kubeaudit	LOW	Capability 'CHOWN' Not Dropped
kubeaudit	LOW	Capability 'DAC_OVERRIDE' Not Dropped

kubeaudit	LOW	Capability 'FOWNER' Not Dropped
kubeaudit	LOW	Capability 'FSETID' Not Dropped
kubeaudit	LOW	Capability 'KILL' Not Dropped
kubeaudit	LOW	Capability 'MKNOD' Not Dropped
kubeaudit	LOW	Capability 'NET_BIND_SERVICE' Not Dropped
kubeaudit	LOW	Capability 'NET_RAW' Not Dropped
kubeaudit	LOW	Capability 'SETFCAP' Not Dropped
kubeaudit	LOW	Capability 'SETGID' Not Dropped
kubeaudit	LOW	Capability 'SETPCAP' Not Dropped
kubeaudit	LOW	Capability 'SETUID' Not Dropped
kubeaudit	LOW	Capability 'SYS_CHROOT' Not Dropped
kubeaudit	LOW	Container Uses a non ReadOnly Root Filesystem
kubeaudit	LOW	Default ServiceAccount uses Automounted Service Account Token
kubeaudit	MEDIUM	NonRoot User not enforced for Container
nikto	INFORMATIONAL	"robots.txt" contains 1 entry which should be manually viewed.
nikto	INFORMATIONAL	Entry '/ftp/' in robots.txt returned a non-forbidden or redirect HTTP code (200)
nikto	INFORMATIONAL	Retrieved access-control-allow-origin header: *
nikto	INFORMATIONAL	Uncommon header 'feature-policy' found, with contents: payment 'self'
nikto	INFORMATIONAL	The Content-Encoding header is set to "deflate" this may mean that the server is vulnerable to the BREACH attack.
nikto	INFORMATIONAL	The site uses SSL and Expect-CT header is not present.
nikto	LOW	The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
kube-hunter	LOW	Access to pod's secrets
kube-hunter	LOW	CAP_NET_RAW Enabled
kube-hunter	LOW	Read access to pod's service account token
sslyze	INFORMATIONAL	TLS Service

A.2 Prototype Evaluation using Bodgelt Store

A.2.1 Bodgelt Store Deployment

```
apiVersion: v1
kind: Namespace
metadata:
  name: bodgeit
  annotations:
    auto-discovery.experimental.securecodebox.io/enabled: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bodgeit
  namespace: bodgeit
  labels:
    app: bodgeit
spec:
  selector:
    matchLabels:
      app: bodgeit
  replicas: 1
  template:
    metadata:
      labels:
        app: bodgeit
    spec:
      containers:
        - name: bodgeit
          image: j12934/bodgeit:latest
          ports:
            - name: http
              containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: bodgeit
  namespace: bodgeit
  labels:
    invasive-scans.auto-discovery.experimental.securecodebox.io: "true"
spec:
  type: ClusterIP
```

```
  ports:
    - port: 8080
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: bodgeit
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: bodgeit
  namespace: bodgeit
  labels:
    invasive-scans.auto-discovery.experimental.securecodebox.io: "true"
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/app-root: /bodgeit
    # An annotation indicating the issuer to use.
    cert-manager.io/cluster-issuer: letsencrypt-production
spec:
  tls:
    - secretName: bodgeit-tls-secret
      hosts:
        - bodgeit.demo.securecodebox.io
  rules:
    - host: bodgeit.demo.securecodebox.io
      http:
        paths:
          - path: /
            backend:
              serviceName: bodgeit
              servicePort: 8080
```

Listing A.2: Kubernetes manifests used to deploy the BodgeIt Store application during the prototype evaluation

A.2.2 Findings Identified for BodgeIt Store

All findings identified for BodgeIt Shop: (Excluding trivy findings to preserve space and with duplicate findings, produced by both scanning Ingress and Service Resources, filtered out)

Scanner	Severity	Name
zap-full-scan	LOW	Absence of Anti-CSRF Tokens
zap-full-scan	LOW	Application Error Disclosure
zap-full-scan	LOW	Content Security Policy (CSP) Header Not Set
zap-full-scan	LOW	Cookie No HttpOnly Flag
zap-full-scan	LOW	Cookie Slack Detector
zap-full-scan	LOW	Cookie Without SameSite Attribute
zap-full-scan	LOW	Feature Policy Header Not Set
zap-full-scan	LOW	In Page Banner Information Leak
zap-full-scan	LOW	Information Disclosure - Debug Error Messages
zap-full-scan	LOW	Server Leaks Version Information via "Server" HTTP Response Header Field
zap-full-scan	LOW	X-Content-Type-Options Header Missing
zap-full-scan	LOW	Cookie Without Secure Flag
zap-full-scan	LOW	Dangerous JS Functions
zap-full-scan	LOW	Incomplete or No Cache-control and Pragma HTTP Header Set
zap-full-scan	LOW	Private IP Disclosure
zap-full-scan	LOW	Strict-Transport-Security Header Not Set
zap-full-scan	INFORMATIONAL	Information Disclosure - Suspicious Comments
zap-full-scan	INFORMATIONAL	Cookie Slack Detector
zap-full-scan	INFORMATIONAL	Modern Web Application
zap-full-scan	INFORMATIONAL	Non-Storable Content
zap-full-scan	INFORMATIONAL	Storable and Cacheable Content
zap-full-scan	INFORMATIONAL	User Agent Fuzzer
zap-full-scan	INFORMATIONAL	User Controllable HTML Element Attribute (Potential XSS)
zap-full-scan	INFORMATIONAL	Base64 Disclosure
zap-full-scan	INFORMATIONAL	Content-Type Header Missing
zap-full-scan	INFORMATIONAL	Cookie Poisoning
zap-full-scan	INFORMATIONAL	GET for POST
zap-full-scan	INFORMATIONAL	Information Disclosure - Sensitive Information in URL
zap-full-scan	INFORMATIONAL	Timestamp Disclosure - Unix
zap-full-scan	MEDIUM	Insecure HTTP Method - PUT
zap-full-scan	MEDIUM	X-Frame-Options Header Not Set

zap-full-scan	MEDIUM	XSLT Injection
zap-full-scan	MEDIUM	.env Information Leak
zap-full-scan	MEDIUM	Application Error Disclosure
zap-full-scan	MEDIUM	Buffer Overflow
zap-full-scan	MEDIUM	Format String Error
zap-full-scan	MEDIUM	HTTP Only Site
zap-full-scan	MEDIUM	Integer Overflow Error
zap-full-scan	MEDIUM	Proxy Disclosure
zap-full-scan	MEDIUM	Relative Path Confusion
zap-full-scan	MEDIUM	Reverse Tabnabbing
zap-full-scan	MEDIUM	Source Code Disclosure - ActiveVFP
zap-full-scan	MEDIUM	Source Code Disclosure - SQL
zap-full-scan	MEDIUM	Source Code Disclosure - Servlet
zap-full-scan	MEDIUM	Trace.axd Information Leak
zap-full-scan	MEDIUM	Weak Authentication Method
zap-full-scan	HIGH	Anti-CSRF Tokens Check
zap-full-scan	HIGH	Cross Site Scripting (Reflected)
zap-full-scan	HIGH	SQL Injection
kubeaudit	LOW	Capability 'AUDIT_WRITE' Not Dropped
kubeaudit	LOW	Capability 'CHOWN' Not Dropped
kubeaudit	LOW	Capability 'DAC_OVERRIDE' Not Dropped
kubeaudit	LOW	Capability 'FOWNER' Not Dropped
kubeaudit	LOW	Capability 'FSETID' Not Dropped
kubeaudit	LOW	Capability 'KILL' Not Dropped
kubeaudit	LOW	Capability 'MKNOD' Not Dropped
kubeaudit	LOW	Capability 'NET_BIND_SERVICE' Not Dropped
kubeaudit	LOW	Capability 'NET_RAW' Not Dropped
kubeaudit	LOW	Capability 'SETFCAP' Not Dropped
kubeaudit	LOW	Capability 'SETGID' Not Dropped
kubeaudit	LOW	Capability 'SETPCAP' Not Dropped
kubeaudit	LOW	Capability 'SETUID' Not Dropped
kubeaudit	LOW	Capability 'SYS_CHROOT' Not Dropped
kubeaudit	LOW	Container Uses a non ReadOnly Root Filesystem
kubeaudit	LOW	Default ServiceAccount uses Automounted Service Account Token
kubeaudit	MEDIUM	NonRoot User not enforced for Container

nikto	INFORMATIONAL	/favicon.ico file identifies this app/server as: Apache Tomcat (possibly 5.5.26 through 8.0.15), Alfresco Community
nikto	INFORMATIONAL	The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
nikto	INFORMATIONAL	Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS
nikto	INFORMATIONAL	HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server.
nikto	INFORMATIONAL	HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web server.
nikto	INFORMATIONAL	The site uses SSL and Expect-CT header is not present.
nikto	INFORMATIONAL	The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
nikto	LOW	The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
nikto	LOW	The anti-clickjacking X-Frame-Options header is not present.
kube-hunter	LOW	Access to pod's secrets
kube-hunter	LOW	CAP_NET_RAW Enabled
kube-hunter	LOW	Read access to pod's service account token
sslyze	INFORMATIONAL	TLS Service