Scanning Containers for Vulnerabilities on Kubernetes Clusters

AUGUST 2018

AUTHOR:
Roberto Soares

CERN IT-CM

SUPERVISORS(S):
Ricardo Brito da Rocha

Project Specification

The interest in using containers to package applications is constantly growing in the software development community, especially with new technologies such as Kubernetes being adopted more frequently as well [1] [2]. At the same time, there is not an easy way for developers to check if a container image contains security vulnerabilities, this means that they could be running containers in their infrastructures that are vulnerable to serious cybersecurity attacks.

The idea of this project is to develop a tool that will find common security vulnerabilities such as CVE (Common Vulnerabilities and Exposures) in container images and report the findings back to the people using such images. To help with that, there are a few tools that try to do that by using static analysis of the layers that make up the container image. The first one is an open-source tool called Clair developed by CoreOS [3], and the other is Atomic Scan developed by Red Hat [4].

The aim of this project is to use one of these tools in Kubernetes clusters to scan running containers and report the results back to the cluster owner. Ideally, this functionality would also be implemented on the OpenStack Magnum project so that all OpenStack clouds can provide this to their users.

Abstract

On this project, we chose to work with Clair, the tool developed by CoreOS, which uses static analysis to find vulnerabilities in container images. To use Clair, we had to build a Python client, called ClairScanner, that communicates with the Clair v1 API. We also had to build a tool called KubeScanner, which runs on Kubernetes clusters and uses the ClairScanner in order to analyse containers that are running on Kubernetes pods. After receiving the results of the analysis from the ClairScanner, the KubeScanner then communicates with the Kubernetes API in order to report the results back to the cluster owner by labeling the pods that were analysed with the findings.

After deploying this solution on the CERN cloud, this project also had the goal of pushing this to the OpenStack Magnum project upstream, which is the OpenStack component responsible for creating clusters for OpenStack clouds.

Contents

1. Introduction

1.1 Context

There is a rapidly growing interest in the software development community to move from monolithic applications to microservices, and in the heart of this change is the idea of packaging applications in containers.

A container is a standardized unit of software. They are like a package where developers put everything they need to run their applications: code, runtime, system tools, libraries, settings, etc. This means that a container is very portable. Containers are also very lightweight, especially compared to virtual machines, so it is quite fast to start a container that runs your application. This performance combined with the portability is what makes it perfect to treat containers as disposable units by using orchestration engines such as Kubernetes.

Kubernetes is a container orchestration engine that is used to manage containers automatically across a cluster of machines. Kubernetes deletes and creates new containers by itself by obeying some configuration constraints (e.g. the number of replicas of a container needs to always be three).

With this scenario in mind, it is easy to see that it is very trivial and fast to have various containers running in an infrastructure. One just needs to go to public repositories of container images, download images found there and run them with an engine of choice.

1.2 Problem

The problem we are facing right now is that, although it is very fast and easy to have multiple containers running in your cluster, there is not an easy way for developers to check if a container image is vulnerable to cybersecurity attacks. This means that we could be running containers in our clusters that are compromised.

1.3 Solution

The idea that we had for this project was to somehow detect vulnerabilities affecting a container and alert the people that are using that container image in order for them to take action and try to mitigate the problem. In order to do that, we used a tool called Clair, developed by CoreOS, that uses static analysis to detect vulnerabilities in containers. We implemented a client for Clair, called ClairScanner, and a tool called KubeScanner that uses the ClairScanner to analyse all the containers running in a Kubernetes cluster. Finally, this solution will be available in future versions of OpenStack Magnum, the OpenStack component responsible for creating clusters in OpenStack.

2. Background

2.1 Docker

Docker is one the most popular and used container engines currently [5]. It is an open-source tool that wraps your application code, libraries, runtime, etc, in a package that is known as a container [6]. In a way, a container is similar to a virtual machine, but it is much more lightweight because there is no virtualization involved. Using Docker, it is very easy to make portable applications and ship them to different environments where they are able to run without having to re-configure everything in the new environment.

Unlike virtual machines, containers share the operating system with the host where they are running on. See figure 2.1 for a comparison between the two.

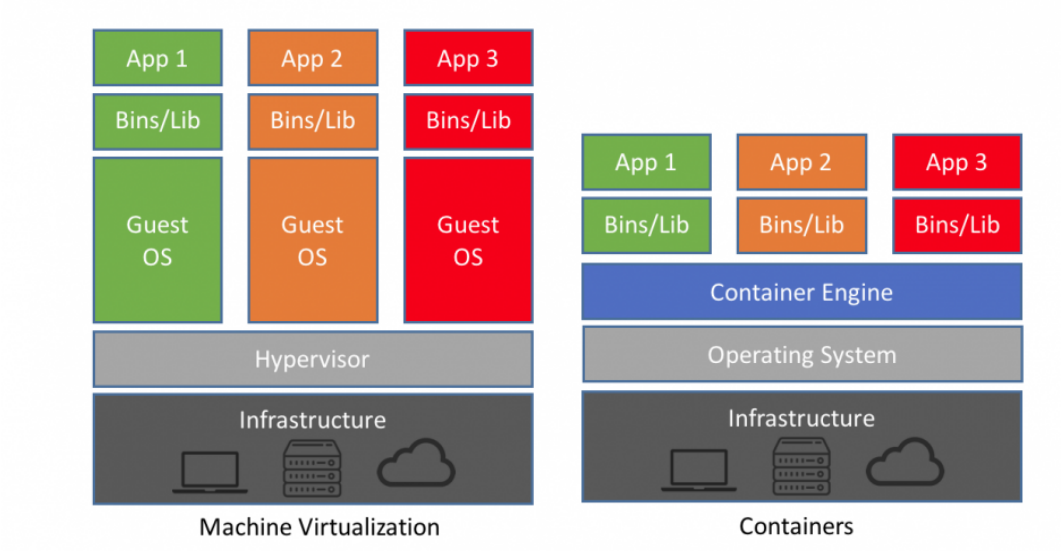

Figure 2.1: Difference between virtual machines and containers [7].

2.2 Kubernetes

Kubernetes is an open-source tool for managing containerized workloads and services. Kubernetes provides a container-centric management environment. It orchestrates computing, networking, and storage infrastructure on behalf of the user [8].

The way it does that is by having this concept referred to as the desired state of users' workloads. This means that users can tell Kubernetes how they want their workloads to behave and Kubernetes will always try to stick to that configuration automatically (e.g. auto scaling when

needed, rescheduling broken containers, etc). See figure 2.2 for an overview of Kubernetes' architecture.

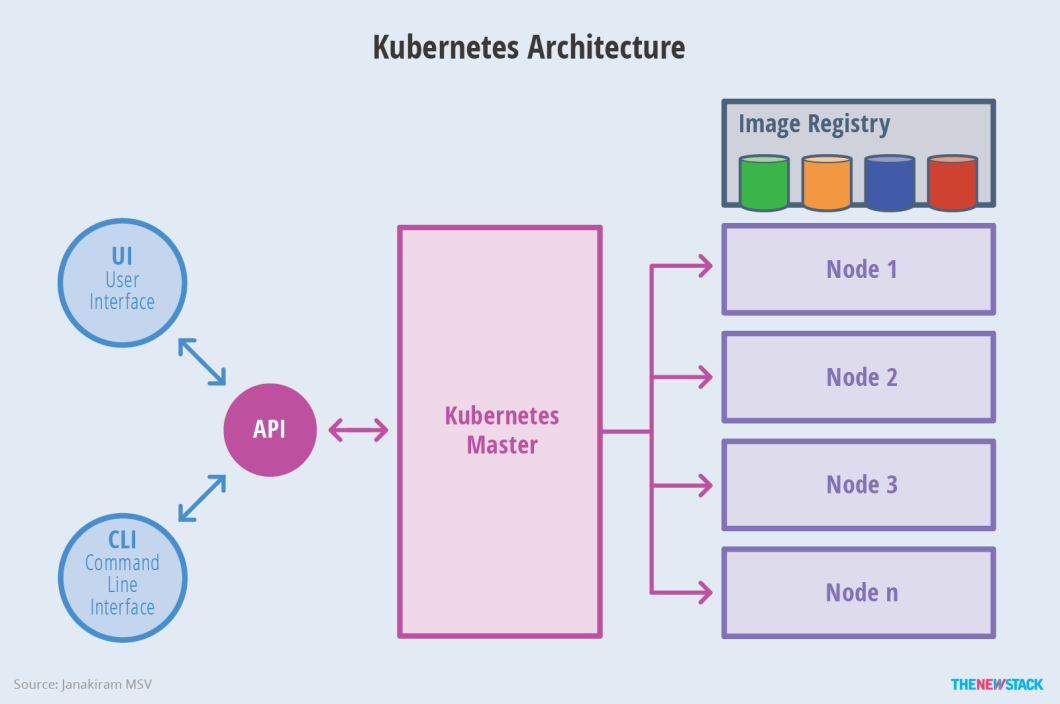

Figure 2.2: An overview of Kubernetes [9].

The master node is where Kubernetes stores the configuration files for the workloads, and the worker nodes are where the containers actually run.

2.3 OpenStack & Magnum

OpenStack is an open-source platform used to manage cloud resources (both public and private). OpenStack has a very large community behind it and it is also backed by some big IT companies [10]. The main idea is that by using OpenStack we can offer cloud resources on-demand to users; for instance, they can create virtual machines, volumes, clusters, networking rules, etc, by using the OpenStack API.

We can think of OpenStack as the open-source equivalent of the software that is running Amazon Web Services, Microsoft Azure or Google Cloud Platform.

Being such a big project, OpenStack is divided in several components, each one being responsible for one aspect of the infrastructure. For example, Keystone deals with identity and authentication; Nova deals with virtual machine creation; Neutron deals with networking, etc. The one interesting to us is Magnum, the component responsible for creating clusters (Kubernetes, Docker Swarm, Apache Mesos) when requested by a user.

Users can use the Magnum API to create, for example, a Kubernetes cluster. When attending this request, Magnum will then talk to the other components in order to create and connect all the nodes of a Kubernetes cluster for the users.

2.4 CoreOS Clair

Clair is an open-source tool created by CoreOS used for static analysis of vulnerabilities in containers. Clair has its own database used to store vulnerability metadata (e.g. CVE, RHSA, etc) from various data sources, like Ubuntu CVE Tracker, Red Hat Security Data, and a few others [11]. This way, Clair ensures that it is always up to date with new vulnerability findings.

Clair works by receiving container images from clients through its API, storing the images in its database and analysing them. The clients can then query Clair to receive the results of the analysis. Clair can also send notifications to configurable endpoints to alert clients that a change has occurred to vulnerability metadata. See figure 2.3 for an overview of Clair's architecture.

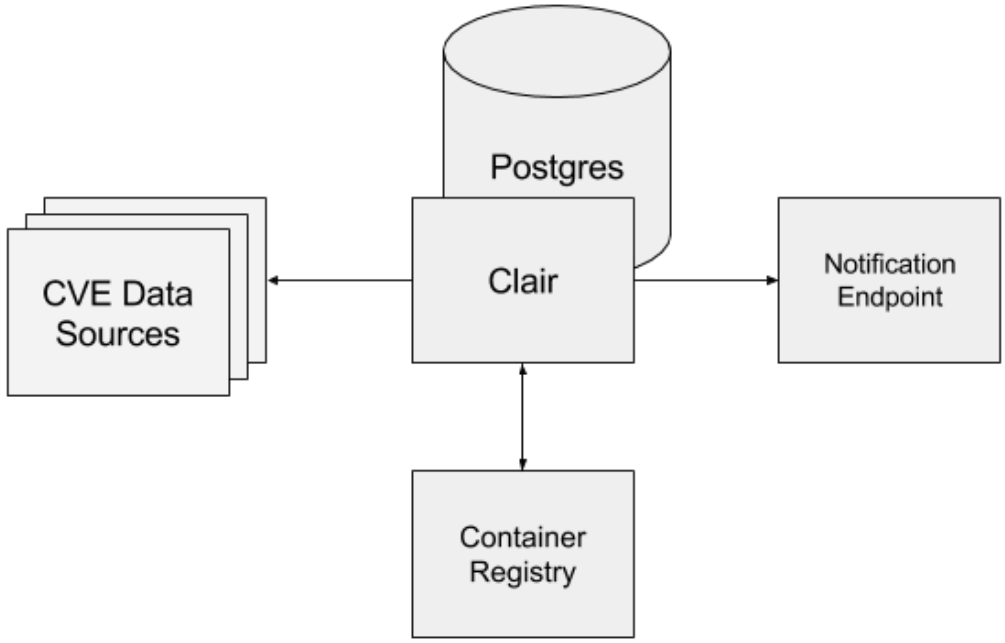

Figure 2.3: An overview of CoreOS Clair [12].

The container registry in figure 2.3 is representing the client that sends the container images to Clair. It can either talk directly to a container image registry or it can be another kind of client that consumes the Clair API.

As you might have noticed, Clair itself is just a server waiting for a client to use its API, so there are a lot of different approaches on how to use it and this allows the community to get creative when implementing clients. The most common use case is integrating Clair into a CI/CD pipeline where it watches the images that are pushed to a container image registry and analyses them to make sure new changes do not introduce vulnerabilities.

3. Implementation

The solution we implemented to solve the problem detailed in section 1.2 is briefly discussed in section 1.3. Here we are going to give a more detailed view on the development of the solution and how we tackled the problem. See figure 3.1 for an overview of the solution.

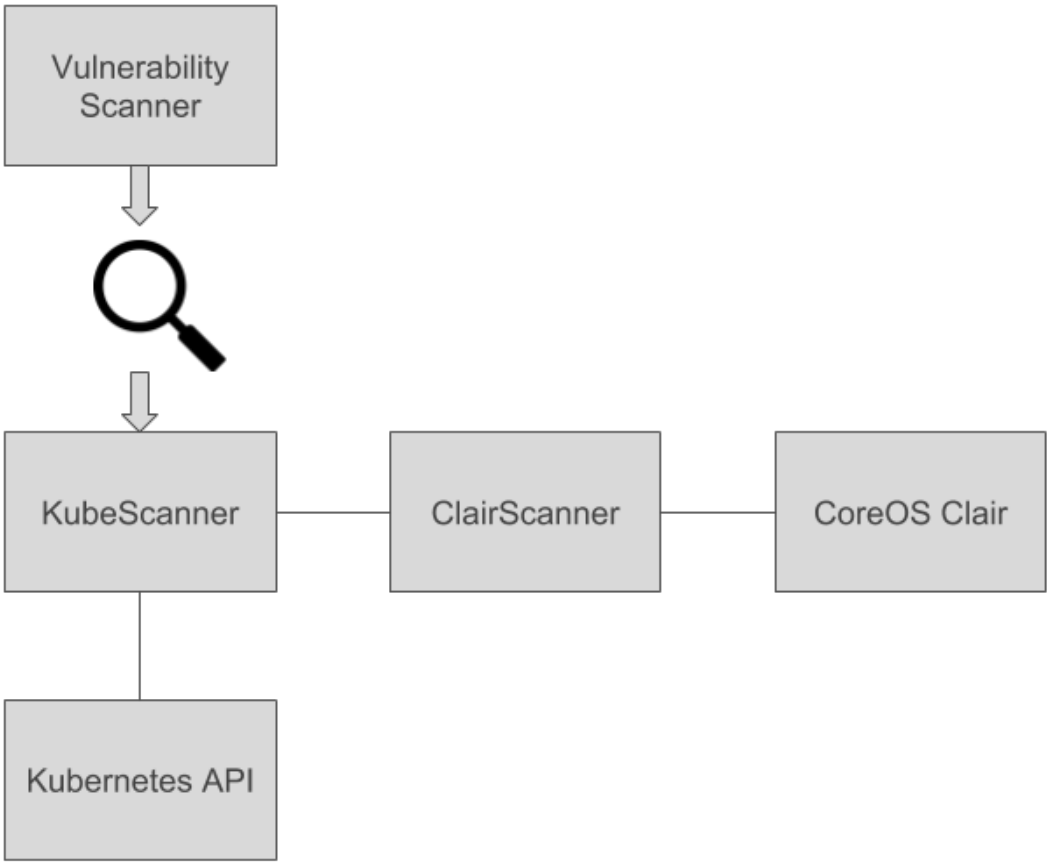

Figure 3.1: Overview of the vulnerability scanner.

The vulnerability scanner is the tool we developed the solve our problem. It basically uses two main libraries developed by us as well, the KubeScanner and the ClairScanner.

The KubeScanner is responsible for interacting with the Kubernetes API to do a few things: first, it will get all the containers that are running in the node where the vulnerability scanner is deployed and pass the names of the images of the containers to the ClairScanner.

The ClairScanner is a client for the CoreOS Clair API, so it receives the container images from the KubeScanner and sends them to Clair for analysis. When it gets the results, it will return them to the KubeScanner.

The second thing KubeScanner does is process the results received from the ClairScanner. It will interact with the Kubernetes API again to add labels and annotations on the pods that were analysed.

The label that the KubeScanner adds is called `security_level` and it has either the value of `safe` or `vulnerable`. This makes it easy for the cluster owner to select only the pods that are safe or vulnerable.

For the annotations, KubeScanner adds two of them. The first one is called `unapproved_vulnerabilities_amount` and it indicates the amount of vulnerabilities that were found in the pod (counting all the containers running there). The second one is called `unapproved_vulnerabilities` and it contains a JSON formatted string with which key being the name of the image analysed, and the corresponding value being an array with the IDs of the vulnerabilities (CVE, RHSA, etc) that were found. With this, the cluster owner can check what are the vulnerabilities affecting the containers and how to mitigate the problem.

When passing the container images for Clair to analyse, the ClairScanner tries to detect the URL of the registry based on the image name and currently it only works with public registries that do not require authentication to pull images.

For the deployment of this solution, we created Kubernetes Helm charts [13] to make the process easier and faster. For clusters without Helm, we created Kubernetes manifests. With either of those options, the vulnerability scanner runs every twelve hours (twice a day) to search for vulnerabilities.

4. Results

The vulnerability scanner has been deployed in a Kubernetes cluster on the CERN cloud and it is properly marking the pods with the appropriate labels and annotations according to the analysis made by Clair. See figure 4.1 for a glimpse of the scanner in action.

```
2018-08-21 14:37:26,085 INFO root - Starting scan...
2018-08-21 14:37:28,572 INFO root - Marking pod dvwa-5bb6cd9987-nknsx as vulnerable. A total of 138 vulnerabilities were found.
2018-08-21 14:37:28,572 INFO root - Image docker.io/vulnerables/web-dvwa:latest has 138 vulnerabilities.
2018-08-21 14:37:28,931 INFO root - Marking pod coredns-65778958d5-fkc7r as vulnerable. A total of 12 vulnerabilities were found.
2018-08-21 14:37:28,931 INFO root - Image gitlab-registry.cern.ch/cloud/atomic-system-containers/coredns:1.0.1 has 12 vulnerabilities.
2018-08-21 14:37:29,670 INFO root - Marking pod csi-attacher-0 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:30,569 INFO root - Marking pod csi-cephfsplugin-hscl5 as vulnerable. A total of 21 vulnerabilities were found.
2018-08-21 14:37:30,569 INFO root - Image gitlab-registry.cern.ch/cloud/atomic-system-containers/cephfsplugin:v0.2.0 has 21 vulnerabilities.
2018-08-21 14:37:30,570 INFO root - Image gitlab-registry.cern.ch/cloud/atomic-system-containers/driver-registrar:v0.2.0 has 0 vulnerabilities.
2018-08-21 14:37:31,126 INFO root - Marking pod csi-cvmfs-attacher-0 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:31,614 INFO root - Marking pod csi-cvmfs-provisioner-0 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:32,160 INFO root - Marking pod csi-cvmfsplugin-tzxnj as vulnerable. A total of 6 vulnerabilities were found.
2018-08-21 14:37:32,160 INFO root - Image gitlab-registry.cern.ch/cloud/atomic-system-containers/cvmfsplugin:v0.2.0 has 6 vulnerabilities.
2018-08-21 14:37:32,160 INFO root - Image gitlab-registry.cern.ch/cloud/atomic-system-containers/driver-registrar:v0.2.0 has 0 vulnerabilities.
2018-08-21 14:37:32,522 INFO root - Marking pod csi-provisioner-0 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:32,928 INFO root - Marking pod heapster-6bf9b9d8d9-d9722 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:33,236 INFO root - Marking pod kubernetes-dashboard-7974bf6d45-svkl7 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:33,704 INFO root - Marking pod manila-provisioner-68686c5bdc-hlb2h as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:34,327 INFO root - Marking pod monitoring-grafana-6d4ffdbb98-q6mnj as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:34,631 INFO root - Marking pod monitoring-influxdb-6c6c9b54fd-ntbhm as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:37:35,412 INFO root - Marking pod tiller-deploy-759cb9df9-kgdft as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:38:07,717 INFO root - Marking pod wandering-quail-vulnerability-scanner-55w76 as safe. 0 vulnerabilities found on all containers.
2018-08-21 14:38:07,739 INFO root - Scan finished.
```

Figure 4.1: Vulnerability scanner running in a Kubernetes cluster.

Another great result from this project is the fact that there does not exist a Python client for the Clair API, so the ClairScanner implemented during this project will be a good contribution to the open-source community, along with the KubeScanner, which integrates the ClairScanner into Kubernetes clusters.

Other contributions to the open-source community that came from this project: an update that was made to the Helm charts of the Clair project upstream [14], and also a contribution to the OpenStack Magnum project upstream [15].

5. Conclusions & Future Work

This project was a success in achieving its goals. We now have a tool that we can deploy on Kubernetes clusters that will automatically scan the containers that are running there in order to search for vulnerabilities and report them back to the cluster owner.

There is still some work to be done on the vulnerability scanner. For example, it still needs to be integrated into the OpenStack Magnum project upstream, and also there are some features that could be added to it to make it more powerful, such as:

- A feature where the users can provide a blacklist containing vulnerabilities they wish to ignore. This can be useful for users that are aware of vulnerabilities found by Clair where the attack vectors are not relevant for their infrastructures.
- Some vulnerabilities detected by Clair already contain a fix available, so the idea is to have a kind of self-healing mechanism which would detect that a vulnerability that was found already has a fix and then would try to fix it automatically.

Finally, it is important to note that the Clair project is going through big changes at the moment and a new API will be available in the near future which will reduce the complexity of Clair clients significantly, so it is a good idea to reduce the complexity of the vulnerability scanner by making it compatible with future changes of Clair.

List of Figures

Bibliography

[1] Increase of container usage rate. https://www.dailyhostnews.com/software-container-adoption-increased-by-50-in-one-year/. ii

[2] Increase usage of Docker containers and Kubernetes. https://www.datadoghq.com/docker-adoption/. ii

[3] CoreOS Clair. https://github.com/coreos/clair. ii

[4] Atomic Scan. https://developers.redhat.com/blog/2016/05/02/introducing-atomic-scan-container-vulnerability-detection/. ii

[5] Docker popularity. https://electric-cloud.com/blog/2017/05/docker-numbers-dockercon-shines-light-containers-enterprise/. 2

[6] What is Docker. https://opensource.com/resources/what-docker. 2

[7] Difference between virtual machines and containers. https://blog.netapp.com/blogs/containers-vs-vms/. 2, 9

[8] What is Kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. 2

[9] An overview of Kubernetes. https://thenewstack.io/kubernetes-an-overview/. 3, 9

[10] What is OpenStack. https://opensource.com/resources/what-is-openstack. 3

[11] CoreOS Clair data sources. https://github.com/coreos/clair/blob/master/Documentation/drivers-and-data-sources.md. 4

[12] An overview of CoreOS Clair. https://github.com/coreos/clair/blob/master/Documentation/running-clair.md. 4, 9

[13] What is Kubernetes Helm. https://helm.sh/. 6

[14] Contribution to CoreOS Clair project upstream. https://github.com/coreos/clair/pull/586. 7

[15] OpenStack Magnum storyboard entry for the project. https://storyboard.openstack.org/#!/story/2003505. 7