Leveraging the Serverless Architecture for Securing Linux Containers

Nilton Bila, Paolo Dettori, Ali Kanso, Yuji Watanabe*, Alaa Youssef
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
*IBM Research - Tokyo, IBM Japan Ltd.
{nilton, dettori, akanso, asyousse}@us.ibm.com, *muew@jp.ibm.com

Abstract— **Linux containers present a lightweight solution to package applications into images and instantiate them in isolated environments. Such images may include vulnerabilities that can be exploited at runtime. A vulnerability scanning service can detect these vulnerabilities by periodically scanning the containers and their images for potential threats. When a threat is detected, an event may be generated to (1) quarantine or terminate the compromised container(s) and optionally (2) remedy the vulnerability by rebuilding a secure image. We believe that such event-driven process is a great fit to be implemented in a serverless architecture. In this paper we explore the design of an automated threat mitigation architecture based on OpenWhisk and Kubernetes.**

Keywords—Linux containers, serverless architecture, Kubernetes, OpenWhisk, Docker, security analysis.

I. INTRODUCTION

With the widespread adoption of cloud technologies, workloads that were once served from on-premise servers, are now shifting to cloud-based execution environments [1]. Linux containers are accelerating this shift by presenting a compelling model for simplified packaging and deployment of applications. Linux containers sandbox applications using built-in kernel mechanisms called namespaces and cgroups. This allows for fast start-up times for containerized applications without penalizing the system performance by adding a virtualization layer. Unlike VMs, containers are instantiated based on lightweight images that include mainly the files and libraries constituting the containerized application [2]. While removing the virtual machine abstraction layer improves performance, it leaves the system more exposed to security threats due to sharing of a single operating system kernel among containers on the same host. As a result, container security is a major concern that has been the focus of recent studies [3, 4]. Containers can have built-in vulnerabilities based on their configuration (e.g. insecure remote shells enabled), or by including executable binaries with known security flaws. Such vulnerabilities can be discovered by vulnerability detection tools that scan container images at build time or images at rest in image registries. While DevOps best practices lean towards immutable containers, the reality is that many container owners apply updates to their running containers, which lead to the introduction of vulnerabilities, even when none were there initially. Moreover, databases of known vulnerabilities are updated periodically with new threat information, resulting in the need to monitor containers at runtime to identify and isolate newly discovered vulnerabilities.

Upon discovering vulnerable containers, state of the art vulnerability scanning tools notify devops professionals to take appropriate actions. This manual reaction may take a long time giving an ample opportunity for exploitation of the detected vulnerability. In large clusters, container clustering and management solutions like Kubernetes [6] and Docker Swarm [5] are used to manage the scheduling and life-cycle of containers. In fact, major cloud providers such as IBM, Microsoft, and Google offer users the ability to automatically deploy their own Kubernetes clusters on their clouds. However, existing container clustering solutions do not provide the mechanisms needed to deal with vulnerable containers. In this work, we seek to address this issue. We believe that users should have control over how security management is enforced on their containers without requiring additional, self-managed, infrastructure. We propose exposing a serveless archtiecture in the container cloud that allows the users to create their own security policies in a generic way. Those policies can be enforced by the serverless framework when a threat alert is triggered. This model alleviates the burden of implementing a security policy manager that would otherwise fall on the users. Another advantage of using the serverless archtiecture is that it maintains consistency by providing a centralized policy manager across multiple Kubernetes clusters. By maintaining the policies outside of the Kubernetes clusters, adding or removing clusters at runtime does not affect the existing user policies.

Our contributions for this work are as follows: first, we introduce API extensions to a Kubernetes based container runtime platform which enables quarantining of vulnerable containers and isolating them from the network while preserving their state for future forensics. Second, we introduce a lightweight policy manager based on a serverless framework, namely OpenWhisk [7], which enables DevOps and security professionals to rapidly develop ad hoc policies that enforce compliance and isolation of offending containers by reacting to events from a vulnerability detection engine and triggering actions that exercise the API extensions introduced above. And third, we extend a vulnerability scanning service to generate event feeds that trigger OpenWhisk policy execution. In this system, the serverless event listener, acting as a policy manager, links the event (vulnerability notice) generated by the vulnerability scanner to the action (quarantine or terminate) using a quickly rigged policy that can be easily modified or augmented to support new unforeseen security scenarios.

II. BACKGROUND

Serverless architecture is a computing paradigm that replaces always-on servers with ephmeral computing

2332-5666/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCSW.2017.66

environments that are created in response to events and removed after executing their tasks. Linux containers are a key enablers of this architecture due to their lighweight images, and fast startup times. Amazon AWS Lambda is well known example of a commercial implementation of the serverless architecture [8]. However Lambda is proprietary and platform specific. OpenWhisk on the other hand provides an open-source platform for implementing serverless services. In fact, OpenWhisk is an open source framework that implements a distributed, event-driven compute service. OpenWhisk runs application logic (actions) in response to events. Actions can be language specific functions or small custom binary code embedded in a Linux container. Application owners write stateless programs (called actions) and register these for invocation on specific event triggers. Event triggers are generated from feeds such as database writes or object store updates or through Web API invocations. Whenever an event is triggered, OpenWhisk instantly deploys and executes the appropriate actions. Each action runs in its own container, and once the action is completed the container is removed.

Kubernetes manages its containers [1] by using a master-worker architecture where a kubernetes agent runs on each cluster node (worker) and reports back to the kubernetes master. The master exposes a management API that can be leveraged by OpenWhisk to execute the remediation actions.

III. Architecture

Our goal is to introduce functionality to kubernetes that enables the system to react to threats in an automated manner according to user-defined policies. OpenWhisk is a key enabler for this functionality since it allows the users to create their own policies in a generic way. In the context of securing a compromised container, the security enforcement service should support abrupt termination, graceful termination, or quarantining of containers. Graceful termination differs from the an abrupt one in that it preserves the container logs and the state of the container filesystem for future investigation after container termination. Quarantine is enforced by blocking any communication into and out of the compromised container.

A. Overall Architecture

Our architecture is composed of four main components. The vulnerability scanner (VS) is our threat detection component. When users deploy new kubernetes clusters, they register the clusters with the vulnerability scanner and install scanner agents in the clusters. When the scanner detects a threat, it posts a notification to the users' registered OpenWhisk action which will determine the appropriate actions to take in order to mitigate the threat. The notification contains the identity of the container pods impacted by the threat. The OpenWhisk policy action invokes the appropriate Kubernetes API extension which will relay the operation to the Security Enforcement Operator (SEO). Figure 1 shows the overall architecture of our security enforcement platform that leverages the capabilities of the vulnerability scanner, third party components like OpenWhisk, and extensions to Kubernetes APIs encoded in the SEO.

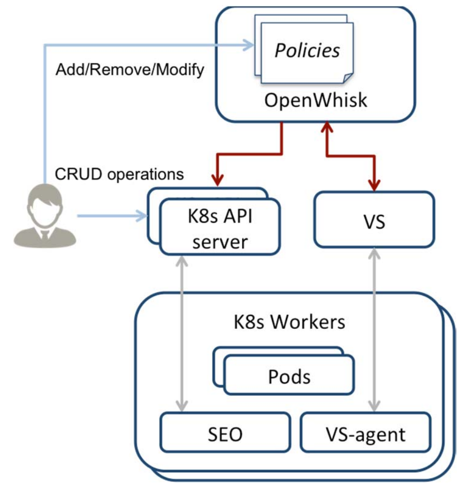

Figure 1. Overall automated threat mitigation architecture.

B. Vulnerability Scanning

Container vulnerability scanning tools such as Vulnerability Advisor [9], Docker Security Scanning [12] and CoreOS Clair [13] leverage online sources of vulnerabilities such as the Common Vulnerability Exposures database [14] and proprietary sources of threat intelligence to identify software vulnerabilities and insecure configurations. These knowledge bases evolve daily with new vulnerability discoveries which can lead containers that were previously deemed secure to become vulnerable.

Our vulnerability scanner leverages the capabilities of the Vulnerability Advisor to periodically scan running containers to discover vulnerabilities at creation time, new vulnerabilities disclosed during the containers life, and vulnerabilities introduced as a result of changes made to the running containers. This scan is performed by local vulnerability scanning agents that send container configuration information to the scanning service. Agents are deployed across all kubernetes cluster nodes. VS periodically ingests threat data from industry knowledge bases and evaluates them against the packages, files and configurations found in scanned containers. The scanner produces reports that identify specific software package versions in the container with disclosed vulnerabilities, and specific issues with the container configurations (e.g. insecure remote shell set up in the container.)

VS supports scans for multiple registered kubernetes clusters with installed agents and uses authentication tokens to restrict access to cluster data at the granularity of kubernetes namespaces. The scanner exposes RESTful APIs for access to vulerability reports for each container and implements a caller for OpenWhisk triggers exposed as Web APIs.

Scans produce new vulnerability findings and may trigger action invocations to the OpenWhisk API endpoints registered for the kubernetes cluster. These triggers carry small JSON documents that contain identifiers for the cluster and pods

[1] In Kubernetes (K8s), pods abstract containers. A pod is a grouping of one or multiple containers that interact closely and share the same fate.

impacted and a summary of the vulnerability findings. Figure 2 illustrates an example sequence of interactions between the various components used by our automated threat mitigation framework.

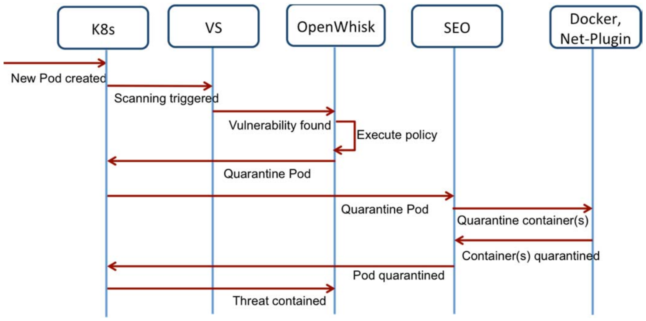

Figure 2. Example flow for threat mitigation.

C. Serverless Policies

OpenWhisk provides a platform on which to implement ad-hoc policies that automate remediation of newly discovered vulnerabilities or security issues in containers.

To implement ad-hoc VS security policies in OpenWhisk, users start by installing the security notices package in their OpenWhisk namespace. The security notices package sets up a trigger that policy actions can subscribe to and externalizes a Web API endpoint that acts as an event feed for the trigger. When VS scans a new container or image, it invokes the OpenWhisk Web API endpoint with the trigger belonging to the Kubernetes cluster owner.

```
import vs
import kubernetes

def main(params):
    findings = vs.get_findings(pod_id, timestamp)
    vulnerable_packages = findings['vulnerable_packages']
    insecure_configs = findings['insecure_configurations']

    if len(vulnerable_packages) > 0:
        kubernetes.snapshot(pod_id)
        kubernetes.terminate_graceful(pod_id)
        return {'text': 'Deleted pod ' + pod_id }

    if 'remote_shell_installed' in insecure_configs:
       kubernetes.quarantine(pod_id)
       return {'text': 'Quarantined pod ' + pod_id}

    return {'text': 'Container was not modified ' + pod_id}
```

Listing 1. A security policy that terminates or quarantines pods according to the severity of security issues.

Listing 1 shows a policy that terminates a pod gracefully when software package vulnerabilities are found and quarantines the pod if the scan finding reveals that a remote shell server is installed. The action uses two modules for communication with the VS and the kubernetes cluster. The *vs* module invokes the get_findings API of VS to obtain a list of vulnerabilities and security findings for the pod. The *kubernetes* module invokes the Kubernetes Web APIs to terminate or quarantine a pod.

D. The Kubernetes Security Enforcement Operator

In order to enforce the security recommendations sepecfied in the OpenWhisk policies, we need an enforcement system with the ability to:

- communicate with every container engine in our cluster;
- isolate the containers beyond the scope of the container engine and Kubernetes;
- expose an API endpoint through which OpenWhisk can communicate its recommendations based on users' policies.

Presently, Kubernetes does not support any feature for container quarantine, or termination based on security recommendations. Therefore we had to introduce this feature through a security enforcement system that extends Kubernetes APIs. Kubernetes supports the notion of third party resources (TPR) to extend its API with endpoints that can be monitoried by external components refered to as Operators. We designed our Security Enforcement Operator (SEO) by leveraging Kubernetes TPR. SEO is deployed on each cluster node where it constantly monitors the TPR API extension for new events. When OpenWhisk invokes the extension APIs to create a security recommendation Kubernetes event, the Operator executes the actions specified in the event, and reports back the execution results. Our Operator can terminate (abruptly or gracefully) or quarantine any given container in the Kubernetes cluster.

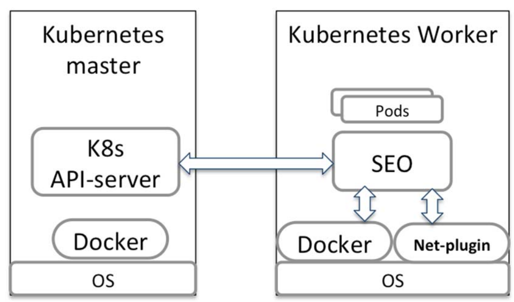

Figure 3. Security Enforcement Operator.

We deploy our SEO as a Kuberntes daemonset. A daemonset deployment ensures that every node in the Kubernetes cluster has at least one instance of the Operator running. Our Operator implements three interfaces as shown in Figure 3. The first interface interacts with the Kubernetes API-server by listening to the third party resources endpoint events (creation, deletion, update) invoked by the policy actions (executed by OpenWhisk). The second interface interacts with our container engine (Docker) in order to commit the container filesystem (thus persisting its content) and abruptly terminate containers running in pods. The third interface interacts directly with the networking plugin used by Kubernetes to manage the containers network configuration. Our SEO currently interacts with Calico, which isolates containers through a lower level mechanism (namely IP tables) in order to block any incoming/outgoing traffic related to an isolated environment.

IV. RELATED WORK

The serverless architecture is still in its infancy, and therefore the related work on using serverless in the context of securing containers is rather scarce. However the research on securing containers is an actively growing area. Barlev et al. [3] present a system protection tool called *Starlight* which implements a kernel module that intercepts local operations on each host and passes them to a local agent which in turn passes them to an event processor that analyzes the event and determines whether or not to alert the admin. Similar to our approach, this work uses a centralized approach for threat analysis, and can learn with time what rules to enforce. It targets applications like the container engine to detect host-escape attacks. However, this work does not perform container scanning to detect threats, hence a dormant threat will remain unnoticed until it is activated. Moreover that work does not leverage the use of serverless architecture to enhance efficiency nor does it implement actions to quarantine compromised containers. Mattetti et al. [4] present LiCShield which generates AppArmor profiles by tracing the container engine (Docker daemon) during the build and the execution of the containers. The profile generation process starts with a tracing phase by creating the container and tracing the kernel operations. The second phase consists of compiling the traces into AppArmor profiles for the container engine and the containers based on their images. LiCShield relates to our work from the aspect of targeting the same problem of securing Linux containers and the container engine. Nevertheless, LiCShield has the same limitations as Starlight, where it does not scan the containers for vulnerabilities, and therefore dormant threats will not be shielded against during the tracing phase. In fact we consider LiCShield and Starlight to be complementary to our work, since they also create security profiles for the container engine itself. We also believe that both works can leverage the serverless architecture to reduce their resource utilization. Catuogno and Galdi [10] present two models for defining the security properties of container-based systems, and argue that the two models are equivalent from the point of view of security threats. The authors also reached a conclusion that the security model adopted in Docker, if fully implemented, is adequate according to the criteria defined in the Common Criteria security certification. This work addresses broadly the criteria to secure containerized systems, which should be followed to ensure that the container host and docker engine are properly secured. The models compared in this paper provide a set of criteria to evaluate the security of a containerized system, however, these models provide general guidelines for configuring the system but they do not address services to detect and react to security vulnerabilities at runtime, which is the focus of our work.

In terms of using serverless-like architecture to introduce new functionalities in the cloud, McGrath et al. [11] present two cloud event applications: one applying Lambdefy framework to demonstrate the differing requirements between applications deployed to IaaS and applications deployed as a cloud event, and Media Management System for showing high scalability of image resizing tasks on Lambda. Similar to our work, this paper addresses use of cloud event technology. However the focus of this paper is performance metrics and optimization aspect of cloud event application using Lambda, and its target is high-performance media management system. It does not address cloud event application to security vulnerability detection nor security for container-based systems.

V. CONCLUSION AND FUTURE WORK

In this work we presented our serverless architecture for securing Linux containers. Our approach provides continous scanning for containers. Upon the detection of vulnerabilities, our automated threat mitigation framework analyzes the vulnerability reports, and based on user defined polycies, triggers the proper action needed to mitigate and contain the threat within the compromised container. In our future work we will investigate the use of artificial intelligence (e.g. machine learning) to automatically generate the security policies based on the severity of the detected threats.

REFERENCES

[1] R. John and J. F. Ransome. *Cloud computing: implementation, management, and security*. CRC press, 2016.

[2] M. Dirk. "Docker: lightweight linux containers for consistent development and deployment." *Linux Journal* 2014.239 (2014)

[3] S. Barlev, Z. Basil, S. Kohanim, R. Peleg, S. Regev and A. Shulman-Peleg, "Secure yet usable: Protecting servers and Linux containers," in IBM Journal of Research and Development, vol. 60, no. 4, pp. 12:1-12:10, July-Aug. 2016. doi: 10.1147/JRD.2016.2574138 keywords

[4] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev and L. Foschini, "Securing the infrastructure and the workloads of linux containers," 2015 IEEE Conference on Communications and Network Security (CNS), Florence, 2015, pp. 559-567.

[5] Docker Swarm, online: https://docs.docker.com/engine/swarm/ accessed on March 27, 2017

[6] Kubernetes, online: https://kubernetes.io/ accessed on March 27, 2017

[7] OpenWhisk, online: https://developer.ibm.com/openwhisk/ accessed on March 27, 2017

[8] D. Poccia, "AWS Lambda in Action: Event-Driven Serverless Applications", Manning Pubn, 2016. ISBN 1617293717, 9781617293719

[9] R. Koller, "Vulnerability Advisor – Secure your Dev + Ops across containers" https://www.ibm.com/blogs/bluemix/2016/11/vulnerability-advisor-secure-your-dev-ops-across-containers/

[10] L. Catuogno and C. Galdi, "On the Evaluation of Security Properties of Containerized Systems," 2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), Granada, 2016, pp. 69-76.

[11] G. McGrath, J. Short, S. Ennis, B. Judson and P. Brenner, "Cloud Event Programming Paradigms: Applications and Analysis," 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2016, pp. 400-406.

[12] Docker Security Scanning, online: https://docs.docker.com/docker-cloud/builds/image-scan/ accessed on March 27, 2017.

[13] CoreOS Clair, online: https://github.com/coreos/clair accessed on March 27, 2017.

[14] Common Vulnerabilities Exposures, online: https://cve.mitre.org acessed on March 27, 2017.