Securing CHEESEHub: A Cloud-based, Containerized Cybersecurity Education Platform

Mike Lambert
lambert8@illinois.edu
National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign
Illinois, USA

Rajesh Kalyanam
rkalyana@purdue.edu
Research Computing, Purdue University
Indiana, USA

Rob Kooper
kooper@illinois.edu
National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign
Illinois, USA

Baijian Yang
byang@purdue.edu
Department of Computer and Information Technology,
Purdue University
Indiana, USA

ABSTRACT

The Cyber Human Ecosystem for Engaged Security Education (CHEESEHub) is an open web platform that hosts community-contributed containerized demonstrations of cybersecurity concepts. In order to maximize flexibility, scalability, and utilization, CHEESEHub is currently hosted in a Kubernetes cluster on the Jetstream academic cloud. In this short paper, we describe the security model of CHEESEHub and specifically the various Kubernetes security features that have been leveraged to secure CHEESEHub. This ensures that the various cybersecurity exploits hosted in the containers cannot be misused, and that potential malicious users of the platform are cordoned off from impacting not just other legitimate users, but also the underlying hosting cloud. More generally, we hope that this article will provide useful information to the research computing community on a less discussed aspect of cloud deployment: the various security features of Kubernetes and their application in practice.

CCS CONCEPTS

• **Information systems** → **Computing platforms**; • **Security and privacy** → **Web application security**; *Vulnerability management*.

KEYWORDS

cybersecurity, containers, cloud computing, Kubernetes

ACM Reference Format:
Mike Lambert, Rajesh Kalyanam, Rob Kooper, and Baijian Yang. 2021. Securing CHEESEHub: A Cloud-based, Containerized Cybersecurity Education Platform. In *Practice and Experience in Advanced Research Computing (PEARC '21), July 18–22, 2021, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3437359.3465584

PEARC '21, July 18–22, 2021, Boston, MA, USA

ACM ISBN 978-1-4503-8292-2/21/07.
https://doi.org/10.1145/3437359.3465584

1 INTRODUCTION

With the pervasive role of computer applications in everyday life, the need to ensure the security of these applications and any associated privileged user information is critical. However, these applications are rarely built from the ground up and often rely on a variety of software libraries that have been designed and built by others. The HeartBleed bug [1] and SQL injection [2] attacks are examples of how missing validation steps in key software libraries can have far reaching impacts on far removed modern web applications. Consequently, practical cybersecurity training should form an integral component of the education of the next generation software engineering workforce.

CHEESEHub [9] is designed for this exact goal, and seeks to impart knowledge of common cybersecurity flaws and solutions through demonstrations of the real-world impacts of these flaws. At its core, CHEESEHub is a public web platform that users can access via their browser, launch these demonstrations on-demand, and follow the accompanying self-paced lessons for reproducing the flaws and evaluating the solutions. In order to simplify contribution, scaling, and deployment to support a variety of educational and training needs, CHEESEHub is designed to be deployable on a Kubernetes cluster with a configurable catalog of applications. While the system design and usage of CHEESEHub have been described elsewhere [9], we focus here on a previously un-discussed aspect: the security model of CHEESEHub, specifically the infrastructure, application, and user level security considerations and our security policies for each. To avoid reinventing the wheel, these security policies heavily leverage various features of the Docker container engine, Kubernetes, and its related plugins. With the growing popularity of cloud-based applications, we hope that this practical account of security features at developers' disposal will foster best practices in the development and deployment of such applications.

The rest of this paper is organized as follows: in Section 2 we provide a brief overview of the CHEESEHub system design and the corresponding infrastructure, application, and user security concerns; in Section 3 we describe the various Docker and Kubernetes security features that are leveraged in handling these concerns; in Section 4 we introduce some additional unique challenges that

arise when using Kubernetes to deploy applications as distributed containers in the cloud, and the corresponding Kubernetes orchestration capabilities utilized in addressing these considerations; and finally in Section 5 we provide some key takeaways, best practices, as well as additional reference resources for securing cloud-based applications.

2 CHEESEHUB DESIGN AND SECURITY CONCERNS

CHEESEHub is built on the National Data Service (NDS) Labs Workbench [8], a framework for deploying web-accessible, container-based analysis environments for a variety of research domains. Labs Workbench is designed for ease of deployment across various public and commercial cloud services and utilizes JSON files for configuring and customizing the set of deployed containerized applications. Figure 1 illustrates the high level system architecture of CHEESEHub with examples of the set of containers that are deployed for a few demonstrations: SQL Injection, ARP poisoning, and HeartBleed.

We focus here on the three highlighted components in Figure 1: the infrastructure, user space, and application containers. The specific security concerns associated with each of these components are as follows:

Infrastructure: This includes the underlying compute cloud infrastructure and the cloud virtual machines that comprise the Kubernetes cluster, as well as the various Kubernetes resources used to deploy CHEESEHub. The primary security concern here is the ability of a malicious user to break out of the *sandbox* environment of the hosted applications and target the host machine or Kubernetes resources, with or without the use of the cybersecurity tools or vulnerable software installed in the hosted applications.

User Space: A given user may launch one or more applications, each of which may include several related containers. The key security concern here is the ability of a malicious user to identify (either via brute force or some other means) the IP addresses of other users' containers and then exploit the security vulnerabilities installed in these containers to deny access or disrupt their use.

Application Containers: While the security implications of potential malicious uses of the application containers have been listed above, the applications themselves also present some inadvertent security risks. Specifically, runaway processes in the containers may consume system resources, preventing the launching or usage of other applications orchestrated on the same host machine. Furthermore, certain applications that involve sets of containers (i.e. client, hacker, and server) typically require unrestricted network traffic between each other or further still, to be on the same "local network". An implementation that allows any Kubernetes pod (i.e. a resource referring to a single or group of deployed container(s)) on CHEESEHub to communicate with any other pod would open the system up to malicious user attacks.

Given these aforementioned security concerns, we next describe the CHEESEHub security model that addresses each of these concerns through various out-of-the-box security features at our disposal.

3 CHEESEHUB SECURITY MODEL

We provide here an overview of the security model inherent in Docker and Kubernetes that make up the full security model presented by the CHEESEHub platform.

Docker Security: While the decision to use containers as the underlying execution mechanism was originally based on portability and reproducibility, containers isolate the vulnerable code and prevent it from directly affecting other processes running on the host machine. By providing a more intuitive interface for creating long-used security mechanisms within Linux, Docker creates multiple *namespaces* that are unique to each container (PID, MNT, NET, UTS, IPC) to prevent cross-container communication across these channels. Linux control groups (*cgroups*) are used to secure hardware utilization and to prevent applications from consuming too many resources on the host, such as CPU or memory. Finally, Docker integrates with system-level kernel security tools such as *SELinux* or *AppArmor* and uses Linux security context capabilities to only grant higher permissions when absolutely necessary [3].

Kubernetes Security: Though its main focus is to handle the orchestration of containers across multiple virtual machines, Kubernetes adds its own set of available security features and configurations on top. It allows the cluster administrator(s) to control access to its own API using role-based access control, and offers several configuration options for network plugins. Choosing particular network plugins, such as *WeaveNet* or *Calico*, can allow the administrator *NetworkPolicy* resources to restrict network egress/ingress traffic to an application. Furthermore, Kubernetes offers mechanisms for easily overriding particular Linux security capabilities mentioned above on an as-needed basis to demonstrate security vulnerabilities. For example: to demonstrate an ArpSpoof and subsequent SSL stripping man-in-the-middle attack, the user must be able to inspect and modify network settings. Adding *NET_ADMIN* to the security context allows us to demonstrate the hack, without providing the user with the broader capabilities that running a container in *privileged* mode allows.

These security features are vital to CHEESEHub to ensure that malicious code from one user's application cannot affect the underlying host machine or the performance of an application run by another user, whether directly or indirectly.

4 ORCHESTRATION CONSIDERATIONS

While the benefits certainly outweigh any deficits, there can be a bit of a learning curve when applying Kubernetes for multi-user container orchestration. The problems mentioned below are inevitable difficulties when distributing an application across multiple virtual machines that Kubernetes helps to address.

Container Networking: Providing network access to an application in a distributed container environment has some slight differences compared to a single host. With Docker, one simply needs to specify the container application ports that they would like to open, and then it is open to the entire host machine. Docker leaves it entirely up to the admin to configure the firewall that will determine whether or not that port is accessible from outside of

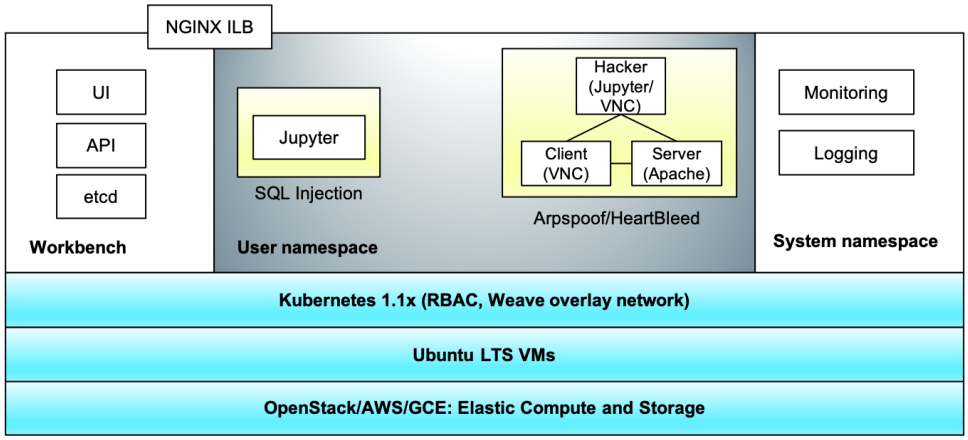

Figure 1: CHEESEHub system architecture illustrating the underlying compute and networking infrastructure, Labs Workbench components, and user facing containerized applications. The specific logical components of interest are: infrastructure (cyan), user space (grey), and containers (yellow).

the host machine. Kubernetes, on the other hand, provides additional methods for network traffic control by expanding the concept of "ports" into two distinct categories: "services" and "ingress" rules. The containers running within a single Pod can communicate openly to each other on a shared network namespace (NET), but by default their ports are not accessible from outside of the Pod. To allow Pod-to-Pod communication, we must create what is called a *Service* that will assign and expose a single static IP that routes traffic to a Pod or group of Pods. Each Pod has its own unique IP, but this IP can change if the Pod is deleted and recreated. Using the Service, other Pods can talk to the static Service IP to route requests to the containers within the exposed Pod without needing to know their actual IP if they ever change. In addition, to expose an application to the public internet and provide users with a way to conveniently access the application from their browser, we also need to set *Ingress* rules for the application. This involves choosing and running an Ingress Controller in the cluster (*NGINX*, *Traefik*, *HAProxy*, etc.), and then creating a set of rules that will route traffic designated for a particular hostname and path to a specific Service. It bears noting here that while most commercial cloud providers (Amazon Web Services, Google Cloud, etc.) provide Ingress controllers out-of-the-box, choosing, deploying, and configuring a particular Ingress controller is necessary in the case of bare-metal cloud environments such as XSEDE's Jetstream [6] which is used to deploy CHEESEHub.

Volume Provisioning: Data that lives only on a single host cannot be shared with other containers. Kubernetes offers many different ways of mounting data from different sources for a range different scenarios. For CHEESEHub, in order to allow the user to have a "home" volume that contains their personal and potentially privileged data, such as SSH or encryption keys, choosing the right Volume Provisioner (such as the NFS Client Provisioner) allows us to mount the user's single volume into many Pods with read/write access (ReadWriteMany). In this case when a new user is created in CHEESEHub, Kubernetes creates an empty directory on the NFS - this is the user's "volume". When a user launches an application, their volume directory is mounted into the application containers. Only the user's subfolder is mounted into their applications, so users cannot access data that is outside their own volume.

Pod / Node Affinity: In some of the more complex application cases, more control is required over which node would execute particular Kubernetes Pods that comprise a user's running application. In ArpSpoof, for example, the Pods for this application (viz. hacker, client, and server) must reside on the same "local network" and therefore must run on the same host node. To assist with this unique scheduling concern, Kubernetes offers optional features for "pod affinity". This powerful feature allows us to define case-by-case rules that control where Kubernetes Pods will execute. For example, in the case of ArpSpoof we can effectively tell Kubernetes to schedule all of the Pods for this application in the same place for our desired network locality.

Application / Host Updates: Patches to the hosted applications and the underlying hosts themselves are often required to protect against newly discovered vulnerabilities. Both Kubernetes and Helm (the Kubernetes package manager) simplify the task of upgrading the deployed application containers via support for rolling updates after modifying configurations such as the desired container image versions. Kubernetes also supports graceful termination of pods running on a host machine (via *kubectl drain*) to allow for host upgrades and patching. The use of Kubernetes *Deployments* and *ReplicaSets* will ensure that the required number of pods are running (via automatic re-deployment to other nodes in the cluster) following such termination on a specific host.

5 DISCUSSION

Given that its goal is to enrich cybersecurity education with hands-on experience with vulnerable code, CHEESEHub has many unique security implications that require special attention. Docker and Kubernetes provide a flexible framework for launching secure applications in any environment, distributed or otherwise, even one with such security needs as CHEESEHub. In this paper we have

focused primarily on presenting these unique CHEESEHub security concerns and providing a practical account of our mitigation strategies. More comprehensive accounts of the security implications for each Kubernetes component/layer and recommended countermeasures are presented in [4, 5]. A formalized Kubernetes threat matrix has also recently been proposed in [7]. These resources can be used for a formal and thorough security audit of Kubernetes-orchestrated applications.

We conclude with a few general recommendations for secure, cloud-based application deployment:

(1) If the application does not already run in a container, then it might benefit from some of the isolation and sand-boxing that Docker provides.
(2) If the goal is to convert a single monolithic application into several smaller distributed microservices, then Kubernetes can provide various methods for making sure that each piece of your application is individually exposed or secured.
(3) If the application already runs within a container, is there a mechanism in place to prevent users from running unauthorized system calls or injecting custom kernel code?
(4) If not, make sure the host operating system runs a Linux kernel that supports some set of Linux security features, such as SELinux or AppArmor.
(5) Make sure to set reasonable resource limits for CPU/MEM on your containers to prevent abuse, and
(6) Consider using NetworkPolicies to restrict any unexpected communications between application pods.

ACKNOWLEDGMENTS

This work is funded by the National Science Foundation through awards 1820573 and 1820608. CHEESEHub platform development and operations are made possible by XSEDE Jetstream. We would also like to thank Craig Willis from the National Center for Supercomputing Applications (NCSA) for his help in developing the CHEESEHub security model, and Alex Withers (also from NCSA) for his help in reviewing the security model.

REFERENCES

[1] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. 2014. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*. 475–488.
[2] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. 2006. A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering*, Vol. 1. IEEE, 13–15.
[3] Docker Inc. 2016. *Introduction to Container Security.* Technical Report. https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf Accessed: 2021-03-26.
[4] Panagiotis Mytilinakis. 2020. *Attack methods and defenses on Kubernetes.* Master's thesis. University of Piraeus.
[5] Md Shazibul Islam Shamim, Farzana Ahamed Bhuiyan, and Akond Rahman. 2020. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. In *2020 IEEE Secure Development (SecDev)*. IEEE, 58–64.
[6] Craig A Stewart, Timothy M Cockerill, Ian Foster, David Hancock, Nirav Merchant, Edwin Skidmore, Daniel Stanzione, James Taylor, Steven Tuecke, George Turner, et al. 2015. Jetstream: a self-provisioned, scalable science and engineering cloud environment. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. 1–8.
[7] Yossi Weizman. 2020. Threat Matrix for Kubernetes. https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/. Accessed: 2021-03-26.
[8] Craig Willis, Mike Lambert, Kenton McHenry, and Christine Kirkpatrick. 2017. Container-based analysis environments for low-barrier access to research data. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. 1–4.
[9] Baijian Yang, Rajesh Kalyanam, Craig Willis, Mike Lambert, and Christine Kirkpatrick. 2019. Cheese: Cyber human ecosystem of engaged security education. In *Proceedings of the 20th Annual SIG Conference on Information Technology Education*. 189–190.