Privilege Escalation Attack Scenarios on the DevOps Pipeline Within a Kubernetes Environment

Nicholas Pecka
Iowa State University, USA

Lotfi ben Othmane
Iowa State University, USA

Altaz Valani
Security Compass, Canada

ABSTRACT

Companies are misled into thinking they solve their security issues by using tooling that is advertised as aligning with DevSecOps principles. This paper aims to answer the question: Could the misuse of the DevOps pipeline subject applications to malicious behavior? To answer the question, we designed a typical DevOps pipeline utilizing Kubernetes (K8s) as a case study environment and analyzed the applicable threats. Then, we developed four attack scenarios against the case study environment: maliciously abusing the user's privilege of deploying containers within the K8s cluster, abusing the Jenkins instance to modify files during the continuous integration, delivery, and deployment systems (CI/CD) build phase, modifying the K8s DNS layer to expose an internal IP to external traffic, and elevating privileges from an account with create, read, update, and delete (CRUD) privileges to root privileges. The attacks answer the research question positively: companies should design and use a secure DevOps pipeline and not expect that utilizing software "advertised as aligning" with DevSecOps principles alone is sufficient to deliver secure software.

CCS CONCEPTS

• **Software and its engineering - Software creation and management**;

KEYWORDS

DevSecOps, Security, Kubernetes, CI/CD

ACM Reference Format:
Nicholas Pecka, Lotfi ben Othmane, and Altaz Valani. 2022. Privilege Escalation Attack Scenarios on the DevOps Pipeline Within a Kubernetes Environment . In *Proceedings of International Conference on Software and Systems Processes (ICSSP).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Companies are adopting the DevOps paradigm [4] where development and operation teams coexist and focus on a consistent development and delivery process. DevSecOps [18, 19, 22], a section of DevOps, incorporates the benefits that DevOps has brought, and includes a security mindset. This mindset helps to evolve security maturity but does not ensure that the system under development is secure on the sole basis that DevSecOps principles are being followed. The integration of development and production environments encouraged companies to adopt DevSecOps by integrating secure software practices and activities into their DevOps systems. Companies adopt DevSecOps models believing it solves their security issues without additional input [18]. The automation of DevSecOps systems, might, however, introduce new security threat vectors despite the great deal of benefits it provides. The question is: *Could misuse of the DevOps pipeline subject applications to malicious behavior?* Companies are being misled into thinking they have solved their security issues simply by utilizing tooling that has the ability to get to a state that aligns with DevSecOps processes but accepting at face value that the tool already is secure based on the security it provides out of the box and how it is being sold to the company. Answering the question positively would demonstrate the need to raise awareness about the importance of the security of DevOps pipelines.

To answer the research question, we created, as a case study, a DevOps pipeline that includes a code repository (GitHub), a CI/CD system [10] (Jenkins), a repository for storing packaged images (DockerHub) and the underlying architecture serving the K8s components [3, 15, 25]–K8s allows a system to be better scaled, monitored, and maintained, including systems that use machine learning [13]. Then, we performed threat modeling [26] of the system. Threat modeling is a process where a system is analyzed for potential security attacks that take advantage of vulnerabilities, quantifying threats, and recommending appropriate remediation [26]. Along with those objectives, we aimed to acquire knowledge on pre-existing vulnerabilities and also potential areas we could exploit for testing. From that research we derived four attack scenarios: (1) retrieval of application data utilizing a custom app that leverages the K8s DNS, (2) manipulate the CI/CD application Jenkins and install a backdoor, (3) expose an internal cluster IP to external users, and (4) leverage a hostPath volume to escape a namespace and gain root access on the host. We look, specifically, at the concept of privilege escalation throughout these scenarios.

The contributions of the paper are:

(1) Developing a threat model of a DevOps environment utilizing Strimzi application as a case study.
(2) Designing four attacks scenarios that demonstrate four of the threats to the DevOps environment.
(3) Proposing mitigation techniques for the identified threats.

The tests show that DevOps pipeline weaknesses could create an insecure software supply chain system (SSCS). The resources for the project including the attack videos are shared [21].

The paper is organized as follows: Section 2 discusses related work, Section 3 provides information on the experimentation environment, Section 4 describes the penetration

ICSSP, May 20-22, 2022, Virtual
2022. ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

tests, Section 5 discusses the proposed mitigations to address the reported attacks, and Section 6 concludes the paper.

2 RELATED WORK

Understanding potential entry points to thwart attackers is vital information. Shamim et al. outlined and explained in Ref [24] the multiple levels of security including authentication, security policies, logging, network isolation, encryption, patching, SSL/TLS, and others in great details. They derived their findings from over 100 internet artifacts. The individual items outlined are not a comprehensive list but were found to be the most affected points of entry across the examined artifacts. Minna et al. extended Shamim et al.'s work [8] by outlining various network-security issues pertaining to a K8s cluster, such as Pod netns by a Pause Container, Container Network Interface (CNI) Plug-Ins Jeopardy, software isolation of resources, network policies limitations, multi tenant K8s clusters, dynamic nature of K8s objects, virtual network infrastructure, and not embedded distributed tracing. In addition, the authors mapped the security of K8s to the Microsoft K8s threat matrix [29]. Karamitsos et al. [13] discuss the impact of business manager in deciding on accepting risks of the DevOps systems due to associated time and cost.

Bertucio analyzed the security of software supply chain system (SSCS) [5]. They break down each component of the supply chain and provide a risk and remediation of each section. They outline a SSCS to depict the points highlighted throughout the blog provided by Google. The paper looks further in detail about specific elements of the supply chain and provides real world examples from an attackers point of view followed by mitigations to said attacks.

For a further look into various attack scenarios, a github user by the name of madhuakula has created an interactive playground called Kubernetes Goat [17]. Users can either follow the instructions on the page to setup their own vulnerable K8s environment or use the built in interactive playground to follow various penetration testing scenarios.

On a related topic, attackers realized that public Continuous Integration (CI) platforms are resource-rich but loosely protected free Internet services and started exploiting that for Cryptomining. For instance, Li et al.[16] discovered 1,974 Cijacking instances, 30 campaigns across 12 different cryptocurrencies on 11 mainstream CI platforms. Further, they unveils the evolution of cryptojacking attack strategies on the CI platforms in response to the protection put in place by these platforms, the duration of the mining jobs (as long as 33 months), and their life cycle. They also discovered that the revenue of the attack is over $20,000 per month.

3 CASE STUDY SETUP

This section describes a system that we setup to demonstrate the use of a SSCS to integrate and deploy a secure application (Strimzi) as an insecure software.

We developed a simple Web application to demonstrate the use of an application with certain privileges to access other components of the DevOps environment [20]. The application

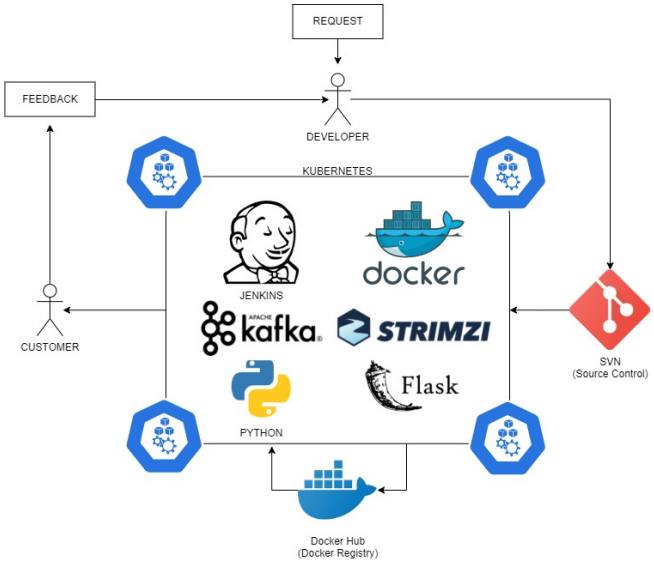

Figure 1: The DevOps environment case study.

is developed in Python [7] utilizing the micro web framework Flask [23] to provide the user/attacker a front-end UI when deployed to the K8s cluster. The application uses the Apache Kafka[2] library to interact with Strimzi [11]. Strimzi is used to streamline the deployment of Apache Kafka [2] on K8s. This application will serve as our secure application for our later attack scenarios.

ESXi [27] is used as the hypervisor [28] to manage four Virtual Machines (VMs)'s that host the experimentation environment. ESXi was built on a bare metal server with an i7-6700K CPU @ 4.00 GHz, 4 CPU cores, and 32 GB RAM. GitHub [9] is used as a code repository for the application. GitHub serves as the trigger point for the Jenkins [14] job. Jenkins is an industry standard for CI/CD [10].

The application components are deployed to K8s [15], an orchestration tool for docker [6] containers that allows a collection of containers to be monitored, managed, and sized at scale. The kube-APIserver is leveraged for the scheduler, controller-manager, and the etcd components to communicate so they can exist separately allowing them to be decoupled. There is then a kubelet on each of the worker nodes that will call back to the APIserver so the other components can manage the cluster properly. An important note is that K8s shrouds the containers in its own networking layer. K8s aligns closely with the DevSecOps mentality and these functionalities provide DevSecOps great tooling. Due to this, however, it might be assumed the applications within a K8s cluster are secure. This paper aims to prove that insider attackers can exploit security weaknesses of DevSecOps pipeline to transform a possibly secure software into an insecure one.

Figure 1 depicts the DevOps pipline case study. From developer to the customer, a developer begins by submitting their code commit to GitHub. A webhook listener from Jenkins triggers from the latest code push to Github and initiates the corresponding Jenkins build job. During the build job, Jenkins pulls the source repository from GitHub including the

Table 1: Outline of the attack scenarios.

ID	Prerequisite privileges	Attack	Component	Attacker Gains
1	Deploy applications with the same namespace as target	Deploy malicious application, leverage application to siphon data from target application	Strimzi	Potential to compromise sensitive data within kafka
2	Access to privileged Jenkins account	Authenticate to Jenkins, edit build job to input bad payload, once deployed access application through backdoor created from bad payload	Jenkins	Installation of potential backdoor into application being built
3	Add objects to desired target	Authenticate to K8s cluster, edit the networking of target applications service, input an ingress object to expose externally	K8s Networking	Ability to connect to app from external
4	create, read, update, and delete (CRUD) privileges	Deploy application in namespace with hostPath volume, leverage the volume by rooting to the host system, gain K8s credentials and has full access to cluster	K8s storage, Docker	Root access to Host box along with K8s cluster

new code changes. Next, it compiles and packages everything into a docker container utilizing built in docker functions to prepare the application for future deployment. Once built, the docker container is pushed to DockerHub [12] and tagged with the build version. Jenkins proceeds to log into the K8s cluster and utilizes docker commands to download the docker image on DockerHub onto the K8s cluster. The image is then deployed to the K8s cluster. The customer then evaluates the new version and provides user feedback that the development team then uses to enhance the DevOps system.

4 ATTACK DEMONSTRATION SCENARIOS ON THE DEVOPS CASE STUDY

The components of the research environment were broken down and analyzed for their inputs and outputs. Through this, we derive potential threats to be used against the system and selected four of them for validation. Table 1 outlines the attacks in the form of prerequisite, attack description, affected component, and attacker's gains from the attack. The remaining of this section describes the four attacks against the case study environment in details.

4.1 Retrieve Information in Topic

The first attack deals with an attacker having privileges to deploy containers to a K8s cluster. The goal is to compromise information from secure applications within the SSCS that exist within the cluster. This data is generally only available within the K8s cluster itself due to the K8s DNS layer. The attacker creates a custom application that, when deployed, will provide a front facing web UI. The custom app used was created as part of this research and is available at Ref. [20]. Once deployed within the cluster, the attacker connects to the front facing UI of the new application and leverages the application to compromise data by siphoning data from the Strimzi application using the custom application. This could lead to compromise secretive data depending on what the custom application is able to retrieve. Listing 1 lists the commands to perform the attack.

```
// Authenticate to Kubernetes cluster and
// confirm kubeconfig matches the desired K8s cluster
// Deploy Strimzi
kubectl apply -f name_of_strimzi.yaml
// Deploy Custom Application
kubectl apply -f name_of_custom_app.yaml
// Verify the applications are in ready state
kubectl get pods \
-n name_of_namespace_where_apps_are_located
// Locate Strimzi internal clusterIP
kubectl get services \
-n name_of_namespace_where_strimzi_is_located
// Locate URL of custom application
kubectl get services \
-n name_of_namespace_where_custom_app_is_located
// Navigate to URL from previous step
// Populate fields of custom application to
// connect to Strimzi
Plug in values clusterIP:Port of Strimzi
// Verify data is sent/received to/form Strimzi
Data will be displayed after executing command
```

Listing 1: Attack steps to Retrieve Information in Topic.

4.2 Manipulate CI/CD by Modifying the Files

The second attack deals with an attacker that has privileges to the Jenkins instance serving the CI/CD. The attacker selects a build step and then modifies the files prior to being packaged and deployed to an online container repository. Once the modified application is deployed, anything may trigger the malicious payload. This could put multiple systems at risk if the application modified is heavily used across a wide array of organizations (open source application such as Strimzi is a great example). Listing 2 lists the commands to perform the attack.

```
// Authenticate to Jenkins
Jenkins login - www.name_of_jenkins_url.com
input username/password
// Locate Build Step
Navigate to proper build step
```

```
// Modify Build Step
Select build step _for modification
// Edit build step by inputting malicious payload
malicious_payload_code
// Execute build job
Run Jenkins build
// Authenticate to K8s cluster
Confirm kubeconfig matches desired K8s cluster
// Pull newly created malicious docker image
docker pull repo_name/image_name/tag
// Deploy malicious docker image
kubectl apply -f name_of_image.yaml
// Trigger malicious payload
Perform triggering action
```

Listing 2: Attack steps to Manipulate CI/CD.

```
// Authenticate to K8s cluster and
// confirm kubeconfig matches desired K8s cluster
// Deploy Strimzi
kubectl apply -f name_of_strimzi.yaml
// Verify Strimzi is in ready state
kubectl get pods -n namespace_of_Strimzi
// Locate Strimzi internal clusterIP
kubectl get services -n namespace_of_Strimzi
// Add NodePort network object to Strimzi
kind: Service
apiVersion: v1
metadata:
  name: strimzi-service
spec:
  selector:
    app: strimzi_app
  ports:
  - protocol: TCP
    port: Strimzi_Port
    nodePort: (30000-32767) - # in _this range
  type: NodePort
// Verify Strimzi is exposed
Contact newly exposed IP
```

Listing 3: Attack steps to expose K8s clusterIP to external users.

4.3 Kubernetes Expose clusterIP to External Users

The third attack deals with an attacker that has access to the K8s cluster to manipulate networking protocols. K8s provides internal cluster IPs, nodeports, and other ingress type objects for K8s resources. These objects allow for internal applications to communicate across the cluster, and to external sources. Services in K8s start with an internal cluster IP that allows for communication with other services within the K8s cluster. The attacker can expose the cluster IP with another ingress object such as a nodeport. The nodeport will attach an external URL that will allow external applications to contact the internal K8s application via the nodeport. With this, an attacker can hook directly into the now insecure application (courtesy of the recent K8s configuration) and siphon secretive data. Listing 3 lists the commands to perform the attack.

```
// Authenticate to K8s cluster and
//confirm kubeconfig matches desired K8s cluster
// Ensure service account has CRUD privileges
// Launch attacker pod with hostPath volume attached
kubectl apply -f attacker_pod_name.yaml
// Exec into the attacker pod
kubectl -n crud_namespace \
exec -it attack_pod_name bash
// Verify account level is not admin
kubectl get secrets -n kube-system
// Verify pod creation in developer ns
Kubectl auth can-i create pod -n crud_namespace
// Check where at in host
echo ${uname -n}
// Execute command to escalate privileges
chroot /host/ bash
// Verify location by checking running containers
docker ps
// Locate kubecfg files and view K8s cluster
/location/to/kubectl \
--kubeconfig=/location/to/kubecfg-kube-node.yaml
// Check pods within K8s cluster
/location/to/kubectl \
--kubeconfig=/location/to/kubecfg-kube-node.yaml \
get pods -A
// Delete a pod within the K8s cluster
/location/to/kubectl \
--kubeconfig=/location/to/kubecfg-kube-node.yaml \
delete pod pod_name -n pod_namespace
```

Listing 4: Steps to perform the Kubernetes Namespace Breakout attack.

4.4 Kubernetes hostPath Namespace Breakout

The final attack deals with a hostPath namespace breakout [1]. A namespace in K8s allows for network segregation and to map deployments too when created. An attacker requires access to a service account with CRUD (create, retrieve, update, delete) privileges in any namespace within the cluster. The attacker deploys a pod within the allowed namespace, then proceeds to abuse the hostPath volume to mount an escape for privilege escalation. A hostPath volume is a storage object that mounts a file or directory from the host node's file system into the pod. Once the attacker deploys their malicious pod, they then exec into it and chroot to access the node's root file system due to the hostPath volume mount exploit. The attacker can then find the kubeconfig files on the host and gain cluster admin privileges. Through this, they can target our secure applications that may exist on other nodes and perform malicious actions against them including editing, deletion, and more. Listing 4 lists the commands to perform the attack.

5 PROPOSED PROTECTION MECHANISMS

The common theme of the attacks is privilege escalation. Thus, the first protection from the attacks is use of the

principle of least privilege when managing a SSCS within the utilized DevSecOps model K8s; limiting account access shrinks the attack vector, as attackers will have less victims to choose from, to perform the type of attacks described in this paper. This section proposes protection mechanisms against the reported attack scenarios of section 4.

Protection from deploying malicious application. The main protection from deploying malicious applications and disclosing confidential information is to limit users privileges. We recommend implementing service accounts that are tied to specific namespaces to prevent users from deploying containers outside their dedicated area.

Protection from the CI/CD manipulation. The main protection from manipulating the CI/CD pipeline is to restrict access to Jenkins instance. We recommend the use of a service account to trigger the Jenkins job and limit other uses to admin/super user to, for instance, override things if needed.

Cluster IP exposure mitigation. The main protection from exposing the IP address of an internal K8s resource externally is to establish service accounts in the K8s cluster. We recommend assigning specific service accounts to access specific resources within certain namespaces and focus the activities monitoring for malicious behavior to specific users that have access to the internal resources in question.

HostPath volume escalation mitigation. The main protection from the hostPath volume namespace breakout is to restrict the CRUD privileges to higher level accounts. Lower level accounts that needs CRUD privileges must authenticate as a high level user to perform their tasks, which focuses the activities monitoring to specific limited accounts. We recommend also to acquire dedicated storage so to prevent the need of hostPath volumes being deployed and instead hosting the storage on another machine that would not be part of the main cluster.

6 CONCLUSIONS

We developed a DevOps pipeline case study, and demonstrated four privilege elevation oriented malicious actions against the internal components of the pipeline that show the possibility to maliciously use the system to make a software insecure. The chosen attack scenarios are maliciously abusing the user's privilege of deploying containers within the K8s cluster, abusing the Jenkins instance to modify files during the CI/CD build phase, modifying the K8s DNS layer to expose an internal IP to external traffic, elevating privileges from a create, read, update, and delete (CRUD) privileges of a low-level account to root privileges.

For the outlined attack scenarios presented in this paper, abiding by the principle of least privilege will mitigate most issues when dealing with privilege escalation. Ensuring that lower level accounts do not possess any form of admin or root access will help reducing the potential attack landscape and enable the security organization to focus on monitoring the activities of the pool of accounts needing those types of privileges. This paper focuses on four specific attack scenarios, but there are numerous other potential attacks not only in the privilege escalation space but at various other levels of the system. Future work can include work in the cloud spaces that utilize K8s rather then the on premise view this paper examines. Also looking into other attack vectors outside of privilege escalation would be good research in the future as well.

REFERENCES

[1] D. Abhisek. 2020. Kubernetes Namespace Breakout using Insecure Host Path Volume - Part 1. https://blog.appsecco.com/kubernetes-namespace-breakout-using-insecure-host-path-volume-part-1-b382f2a6e216. Accessed on Oct. 2021.

[2] Apache. 2021. Apache Kafka. https://kafka.apache.org. Accessed on Sep. 2021.

[3] C. Artur, I. Iustin-Alexandru, B. Robert, D. Virgil, and C. Ovidiu. 2020. Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes. *19th RoEduNet Conference: Networking in Education and Research (RoEduNet)* (7 2020), 6.

[4] AWS. 2021. What is DevOps. https://aws.amazon.com/devops/what-is-devops/. Accessed on Sep. 2021.

[5] A. Bertucio. 2021. Protect your open source project from supply chain attacks. https://opensource.googleblog.com/2021/10/protect-your-open-source-project-from-supply-chain-attacks.html?m=1. Accessed on Nov. 2021.

[6] Docker. 2013. Developers Love Docker. Businesses Trust It. https://docker.com. Accessed on Oct. 2021.

[7] Python Software Foundation. 2001. Python. https://python.org. Accessed on Oct. 2021.

[8] M. Francesco, C. Balakrishnan, B. Agathe, R. Filippo, and M. Fabio. 2021. Understanding the Security Implications of Kubernetes Networking. *IEEE Computer and Reliability Societies* (09 2021), 11.

[9] Inc. GitHub. 2008. GitHub. https://github.com. Accessed on Oct. 2021.

[10] Red Hat. 2018. What is CI/CD? https://www.redhat.com/en/topics/devops/what-is-ci-cd. Accessed on Oct. 2021.

[11] Red Hat. 2021. Apache Kafka on Kubernetes. https://github.com/strimzi. Accessed on Sep. 2021.

[12] Docker Inc. 2011. Build and Ship any Application Anywhere. https://hub.docker.com. Accessed on Oct. 2021.

[13] K. Ioannis, A. Saeed, and A. Charalampos. 2020. Understanding the Security Implications of Kubernetes Networking. *Information 2020, 11, 362* (11 2020), 15.

[14] Jenkins. 2011. Jenkins. https://jenkins.io. Accessed on Oct. 2021.

[15] Kubernetes. 2021. Production-Grade Container Orchestration. https://kubernetes.io. Accessed on Sep. 2021.

[16] Z. Li, W. Liu, H. Chen, X. Wang, X. Liao, L. Xing, M. Zha, H. Jin, and D. Zou. 2022. Robbery on DevOps: Understanding and Mitigating Illicit Cryptomining on Continuous Integration Service Platforms. In *2022 2022 IEEE Symposium on Security and Privacy (SP) (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 363–378. https://doi.org/10.1109/SP46214.2022.00022

[17] madhuakula. 2020. Kubernetes Goat. https://github.com/madhuakula/kubernetes-goat. Accessed on Nov. 2021.

[18] V. Mohan and L. Ben Othmane. 2016. SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*. 542–547. https://doi.org/10.1109/ARES.2016.92

[19] V. Mohan, L. ben Othmane, and A. Kres. 2018. BP: Security Concerns and Best Practices for Automation of Software Deployment Processes: An Industrial Case Study. In *2018 IEEE Cybersecurity Development (SecDev)*. 21–28.

[20] N. Pecka. 2021. py-producer-consumer. https://github.com/npecka/py-producer-consumer. Accessed on Oct. 2021.

[21] N. Pecka and L. Ben Othmane. 2022. Insider Attacks on the DevOps Pipeline. https://github.com/npecka/InsiderAttacksontheDevOpsPipeline.

[22] A. Quintessence. 2021. The DevSecOps Cultural Transformation. https://www.pagerduty.com/blog/devsecops-ops-guide/. Accessed on Jan. 2022.

[23] A. Ronacher. 2010. Flask web development, one drop at a time. https://flask.palletsprojects.com/en/2.0.x/. Accessed on Oct. 2021.
[24] S. Islam Shazibul, B. Ahamed Farzana, and R. Akond. 2020. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. *arXiv:2006.15275v1 [cs.CR]* (06 2020), 7.
[25] G. Somya and G. Satvik. 2019. Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security. In *IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. San Jose, CA, 5.
[26] Synopsys. 2021. Threat Modeling. https://https://www.synopsys.com/glossary/what-is-threat-modeling.html. Accessed on Nov. 2021.
[27] VMWare. 2001. ESXi. https://www.vmware.com/products/esxi-and-esx.html. Accessed on Oct. 2021.
[28] VMWare. 2001. What is a hypervisor? https://www.vmware.com/topics/glossary. Accessed on Oct. 2021.
[29] Y. Weizman. 2020. Threat matrix for Kubernetes. https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/. Accessed on Oct. 2021.