Take Over the Whole Cluster: Attacking Kubernetes via Excessive Permissions of Third-party Applications

Nanzi Yang
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Wenbo Shen*
Zhejiang University
ZJU-Hangzhou Global Scientific and
Technological Innovation Center
Hangzhou, China

Jinku Li*
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Xunqi Liu
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Xin Guo
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Jianfeng Ma
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

ABSTRACT

As the dominant container orchestration system, Kubernetes is widely used by many companies and cloud vendors. It runs third-party add-ons and applications (termed *third-party apps*) on its control plane to manage the whole cluster. The security of these third-party apps is critical to the whole cluster but has not been systematically studied so far.

Therefore, this paper analyzes the security of third-party apps and reveals that third-party apps are granted excessive critical permissions, which can be exploited by an attacker to escape from the worker node and take over the whole Kubernetes cluster. Even worse, excessive permissions of different third-party apps can be chained together to turn non-critical issues into severe attack vectors. To systematically analyze the exploitability of excessive permissions, we design three strategies based on different attacking paths. These three strategies can steal the cluster admin permission with the DaemonSet of a third-party app directly, or via the same app's or another app's critical component indirectly.

We investigate the security impact of excessive permission attacks in real production environments. We analyze all third-party apps in CNCF and show that 51 of 153 (33.3%) ones have potential security risks. We further scan Kubernetes services provided by the top four cloud vendors. The results show that all of them are vulnerable to excessive permission attacks. We report all our findings to the corresponding teams and get eight new CVEs from communities and a security bounty from Google.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**.

---

*Co-corresponding authors.

KEYWORDS

1 INTRODUCTION

Kubernetes is a container orchestration system for automating deployment, scaling, and management of containerized applications [6]. In recent years, it has been widely adopted by numerous companies to manage their workloads [7] and used by cloud vendors as their container orchestrators [18, 59, 61, 69]. According to the CNCF (Cloud Native Computing Foundation), 89% of CNCF end users are either using or evaluating Kubernetes [34], which makes it a dominant player in the container orchestration market. Sysdig report shows that Kubernetes has a commanding 75% share of the container orchestrator market [68]. Therefore, Kubernetes has become the de-facto standard for container orchestration.

A Kubernetes cluster usually consists of a control plane and a data plane [8]. The control plane acts as the central controller of the whole cluster and deploys the workload of cluster users to worker nodes in the data plane. The control plane contains the necessary components of Kubernetes, including kube-apiserver, kube-scheduler, etcd, and controller-manager. Moreover, it also runs add-ons or applications from third parties. Here we term these third-party add-ons and applications running on Kubernetes control plane as *third-party apps*. These third-party apps are used to extend the control functionality of Kubernetes and are granted critical permissions for cluster management. Therefore, their security is critical to the whole cluster.

Unfortunately, the security of third-party apps has not been systematically studied before. Existing research on Kubernetes security is mainly on the co-residency attacks [64], the DDoS attacks against the auto-scaling [26], and misconfigurations [42, 62].

Therefore, in this paper, we analyze the security of third-party apps and reveal a new attack surface called *excessive permission attack*, which can be exploited to attack the whole Kubernetes cluster. Our key observation is that many third-party apps are granted additional critical permissions, called *excessive permissions*, which can be abused to make multiple attacks in Kubernetes, named *excessive permission attacks*. For example, by abusing excessive permissions, an attacker can escape from the worker node and gain control over the whole Kubernetes cluster, resulting in privilege escalation in Kubernetes.

Even worse, our study demonstrates that excessive permissions of different third-party apps can be chained together to turn noncritical issues into severe attack vectors.

To systematically analyze the exploitability of third-party apps' excessive permissions, we propose three strategies based on different attacking paths. First, the attacker on a worker node can exploit excessive permissions of third-party apps' DaemonSet [9] to directly steal the cluster admin permission, thus it escalates its privilege in Kubernetes. Second, an attacker can leverage the DaemonSet's excessive permissions to hijack the same app's critical component and then use their excessive permissions to indirectly steal the cluster admin permission. Third, an attacker can utilize the excessive permissions from the DaemonSet of one third-party app to hijack the critical components of another third-party app and then use them to indirectly steal the cluster admin permission. As a result, in all three strategies, an attacker who controls a worker node can escalate to the cluster admin and take over the whole cluster.

To fully investigate the security impact of excessive permission attacks in real production environments, we develop a tool following our strategies and use it to scan third-party apps in CNCF projects as well as third-party apps used in top four public clouds. Specifically, we scan all the 153 projects in the CNCF project lists and find that 51 (33.3%) of them have potential security risks. For the top four public clouds, we select Google Kubernetes Engine (GKE) [73], Amazon Elastic Kubernetes Service (Amazon EKS) [21], Azure Kubernetes Service (AKS) [23], and Alibaba Cloud Container Service for Kubernetes (Alibaba Cloud ACK) [32]. Our results show that all of them are vulnerable to excessive permission attacks. In particular, three third-party apps in Google GKE, three third-party apps in Amazon EKS, two third-party apps in Azure AKS, and nine third-party apps in Alibaba Cloud ACK have potential security risks.

For the 51 vulnerable CNCF projects, we report the identified issues to the related communities through the proper channels. For the top four public clouds, we also report our findings to their security teams. Our findings are confirmed and *eight new CVEs have been assigned to us*. Moreover, Google has awarded us *a bounty of $1337.00* for our findings.

In summary, the major contributions of this paper are summarized as follows.

New Attack Surface. We reveal that third-party apps are granted excessive permissions, which can be abused to make attacks and cause severe consequences (e.g., getting cluster admin permission and taking over the whole cluster). We term these attacks as *excessive permission attacks*.
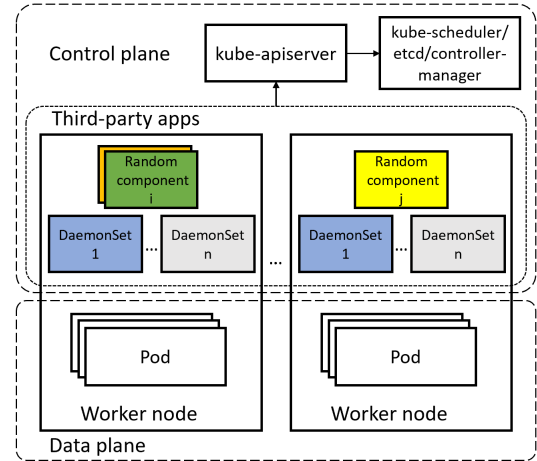

Figure 1: The overview of Kubernetes architecture. "Random component" means a randomly-deployed component.

Attacking Strategies. We analyze the attack paths systematically by designing three attacking strategies. These three strategies can steal the cluster admin permission from the DaemonSet of a third-party app directly, or via the same app's or another app's critical component indirectly.

Practicality Evaluation. We analyze all third-party apps in CNCF and show that 51 of 153 (33.3%) ones have potential security risks. We further scan Kubernetes services provided by the top four public clouds. The results show that all of them are vulnerable to excessive permission attacks.

Community Impact. We report all our findings to the corresponding teams and get eight new CVEs from communities and a security bounty from Google. We also plan to open-source our tool at `https://github.com/XDU-SysSec/ExcessivePermissionAttack` so that it can help researchers and developers pinpoint the weak points of their third-party apps.

The rest of this paper is structured as follows. We introduce the necessary background knowledge in §2. We use a real example as a motivation to illustrate the details of the excessive permission attack in §3. We give out three strategies for making excessive permission attacks via third-party apps in §4. We scan excessive permission risks in CNCF projects and cloud vendors in §5. We discuss the measurements to mitigate these issues in §6. We summarize the related work in §7 and conclude our paper in §8.

2 BACKGROUND

In this section, we present the necessary background knowledge of our work, including the Kubernetes architecture, third-party apps of Kubernetes, and role-based access control.

2.1 Kubernetes Architecture

Kubernetes is a container orchestration system that provides a framework for creating and managing containers at scale. The overview of Kubernetes architecture is shown in Figure 1. Particularly, a Kubernetes cluster comprises a control plane and a data

plane. The control plane is responsible for managing and maintaining the cluster, which contains the Kubernetes components and third-party apps. The data plane is responsible for running workloads of different users, and it contains all pods of different users that are running on multiple worker nodes.

In the control plane, several Kubernetes components (including kube-apiserver, kube-scheduler, etcd, and controller-manager) maintain the cluster's functionalities. For example, kube-apiserver provides the cluster's Restful API front end, through which containers can interact with multiple Kubernetes resources. In addition, there are usually a number of third-party apps in the control plane to extend the cluster's functionalities and serve the entire cluster. A typical third-party app contains two types of components according to their different running locations, i.e., the DaemonSet and one or more randomly-deployed components (*random components* for short). Specifically, the DaemonSet is a set of pods and each of them runs on a separate worker node. The random components of a third-party app are sets of pods that run on worker nodes randomly, which could be Kubernetes Deployments [10], StatefulSets [16], and so on. As shown in Figure 1, there are multiple third-party apps in a Kubernetes cluster. The DaemonSets of third-party apps are running on each worker node (i.e., DaemonSet 1, ..., DaemonSet n). While the random components of different apps are running on worker nodes randomly (e.g., the random component *i* is running on the first worker node, and *j* is running on the last worker node).

In the data plane, a pod is a group of one or more containers that share storage and network resources, which is the basic unit created and managed by Kubernetes. The worker nodes, which reside on physical or virtual machines, are responsible for running containers of different users. For security purposes, cloud vendors typically recommend running different users' containers on separate worker nodes to prevent container breakouts [70], isolate workloads [20] and tenants, and enforce access controls [24].

2.2 Third-party Apps of Kubernetes

Third-party apps of Kubernetes are a series of software that can be installed and run in the Kubernetes cluster. They provide external functionalities for interacting with underlying resources and improve the effectiveness of the Kubernetes cluster.

There are lots of third-party apps in the real world, which are maintained by different vendors and communities. Specifically, there are 153 projects in the CNCF project lists, which are divided into three groups, i.e., graduated, incubating, and sandbox. For example, Rook[1] is a graduated project (or app) to provide cloud-native storage for Kubernetes. And Kubevirt[2] is an incubating project for managing virtualization workloads inside a Kubernetes cluster. While Kubewarden[3] is a sandbox project which offers a policy engine for Kubernetes.

Besides, cloud vendors use a number of third-party apps in their business environments. For example, Google GKE [73] uses third-party apps for extending its features [71]. Amazon EKS has multiple third-party apps for managing underlying AWS resources such as networking, computing, and storage [19]. Azure AKS enables

---

[1]https://rook.io/

[2]https://kubevirt.io/

[3]https://www.kubewarden.io/

```
1  Name:           kubevirt-operator
2  Labels:         kubevirt.io=
3  Annotations:    <none>
4  Role:
5    Kind:  ClusterRole
6    Name:  kubevirt-operator
7  Subjects:
8    Kind            Name              Namespace
9    ----            ----              ---------
10   ServiceAccount  kubevirt-operator  kubevirt
11
```

Figure 2: The `kubevirt-operator` ClusterRoleBinding. It binds the `kubevirt-operator` ClusterRole to the `kubevirt-operator` service account.

```
1  Name:           kubevirt-operator
2  Labels:         kubevirt.io=
3  Annotations:    <none>
4  PolicyRule:
5    Resources Non-Resource URLs Resource Names Verbs
6    --------- ----------------- -------------- -----
7    ...
8    secrets   []                []             [create list get watch]
9    ...
```

Figure 3: The `kubevirt-operator` ClusterRole. It has the "list" verb of the "secrets" resource.

```
1  Name:           kubevirt-handler
2  ...
3  PolicyRule:
4    Resources Non-Resource URLs Resource Names Verbs
5    --------- ----------------- -------------- -----
6    ...
7    nodes     []                []             [patch list watch get]
8    ...
```

Figure 4: The `kubevirt-handler` ClusterRole. It has the "patch" verb of the "nodes" resource.

several third-party apps to provide extra capabilities for Kubernetes clusters [22]. And Alibaba Cloud ACK leverages third-party apps to enable multi-host networking and other extra functionalities [33].

However, third-party apps of Kubernetes are usually developed and maintained by different third-party software vendors. Among these vendors, a number of them may lack security experts to review the source code of their apps. Thus, such apps possibly introduce potential security risks, e.g., applying for excessive permissions unnecessary for apps to work properly, which motivates us in this work.

2.3 Role-based Access Control

Kubernetes enables role-based access control (RBAC) by default to restrain the container's access permissions to Kubernetes resources [17]. In the RBAC mechanism, permissions are grouped into Roles and ClusterRoles [12], which can be granted to the pod's service account [15] through the RoleBinding and ClusterRoleBinding operations. The permissions granted by ClusterRoleBinding apply to the entire cluster, and the permissions granted by RoleBinding are limited to a specific Kubernetes namespace [11]. When containers make resource access requests to the kube-apiserver, only requests that comply with the RBAC permissions will be allowed.

For example, the third-party app Kubevirt has a random component named `virt-operator`, which has a service account called `kubevirt-operator`. As shown in Figure 2, the `kubevirt-operator` service account has a ClusterRoleBinding called `kubevirt-operator` (line 1), which binds the `kubevirt-operator` ClusterRole (line 6) to the `kubevirt-operator` service account of the `kubevirt` namespace (line 10). As shown in Figure 3, the `kubevirt-operator` ClusterRole (line 1) has a "list" verb of the "secrets" resource (line 8), which enables the `virt-operator` component of the app to list all the secrets in the entire cluster. This permission is excessive for the `virt-operator` component, as it only needs to retrieve several specific secrets inside the `kubevirt` namespace (e.g., `kubevirt-ca`, `kubevirt-export-ca`, etc.), but not all the secrets in the whole cluster. Secrets are objects that contain sensitive data such as passwords, tokens (e.g., the cluster admin token), or keys [13]. Note that multiple service account tokens, including the cluster admin token if created, are stored as "secrets" resources inside the Kubernetes cluster. Retrieving all secrets allows the attacker to acquire all user tokens across the whole cluster. In other words, the `virt-operator` component of the app can leverage this excessive permission to obtain the cluster admin token, which provides superuser access to the Kubernetes cluster.

Besides the random component `virt-operator`, Kubevirt has a DaemonSet called `virt-handler`, which has a set of pods running on each node. The service account of `virt-handler` is bonded with a ClusterRole named `kubevirt-handler` via ClusterRoleBinding. As shown in Figure 4, this ClusterRole has a "patch" verb of the "nodes" resource, which is able to update field(s) of the "nodes" resource using strategic merge. This permission is excessive for the `virt-handler` DaemonSet, as it makes the `virt-handler` on a single node modify all the worker nodes in the entire cluster. As a result, the `virt-handler` running on a worker node can abuse this excessive permission to patch (or modify) all other nodes in the cluster and force the `virt-operator` to run on its own node.

In a real scenario, a malicious user can chain these excessive permissions together to build attack vectors. More specifically, an attacker can first abuse the excessive permission of `virt-handler` to force the `virt-operator` to run on its own node. After that, the attacker can exploit the excessive permission of `virt-operator` to steal the cluster admin token. As a result, the attacker gains full control over the whole cluster, which is similar to obtaining root user privileges in Linux/Unix. We present more details in §3.2.

3 MOTIVATION

In this section, we first describe the threat model and assumptions of our work. Then we present the motivation of this paper and use a real example to illustrate the details.

3.1 Threat Model and Assumptions

In this paper, we assume that an attacker controls one worker node of Kubernetes (a.k.a, attacker-controlled worker node), and aims to take over the whole cluster. Note that in a real scenario, if an attacker has access to the applications in the cluster, the attacker can: (1) compromise the applications running inside the container, (2) perform a container escape to compromise a worker node, and

(3) take control of the whole cluster. Our paper concentrates on the third step. This assumption is based on several factors. First, application compromising and container escapes are prevalent in real-world scenarios [29, 49, 51, 55, 58]. A malicious user who has access to applications inside a container can first leverage vulnerabilities of applications to compromise the container [29, 55]. After that, the attacker is able to exploit multiple existing vulnerabilities to break out the container and compromise a worker node [65]. Second, in a Kubernetes scenario, what happens after compromising a node is unknown. More specifically, a container escape only compromises a single node without affecting the workloads on other nodes. Third, there is a commonly held belief that worker nodes provide strong isolation [20, 24, 70]. For example, Google GKE suggests isolating workloads in separate nodes to reduce the risk of privilege escalation [70]. Amazon EKS recommends isolating tenant workloads to specific nodes to increase isolation in the soft multi-tenancy model [20]. Azure AKS proposes to use dedicated nodes for isolating teams and workloads [24]. In contrast, our research shows that the excessive permissions of third-party apps can be abused to break the isolation of worker nodes and take over the whole cluster, showing the cases that which the common belief does not hold.

On the restriction side, the third-party apps are deployed following their official documents. In addition, other Kubernetes restrictions work properly to harden the cluster following the Kubernetes best practices [14]. Specifically, these restrictions include controlling accesses to the Kubernetes APIs, controlling accesses to the Kubelet, controlling the capabilities of a workload or user at runtime, and protecting cluster components from being compromised.

In the following, we use a real example to show that even if the aforementioned restrictions are in place, an attacker who controls one worker node can still abuse the excessive permissions of third-party apps to take over the whole cluster, which poses a high impact on the cluster's confidentiality, integrity, and availability.

3.2 Motivating Example: Excessive Permission Attack via Kubevirt

Kubevirt[2] is a Kubernetes third-party app that manages virtual machines inside a Kubernetes cluster, and it is widely adopted by multiple companies (e.g., ARM, Nvidia, and RedHat) [45]. Specifically, Kubervirt has two components. One component is the `virt-handler`, which is the DaemonSet [9] that runs a pod on each worker node. The other component is the `virt-operator`, which is a random component running as a Kubernetes Deployment [10] on worker nodes randomly.

Attack Definition. The excessive permissions refer to those permissions unrelated to the app's normal functionalities. In other words, these permissions are granted to the app's components even though they are unnecessary for the proper functioning of the app. An attacker can abuse these excessive permissions to launch attacks, which we call *excessive permission attacks*.

For example, in Kubevirt, the `virt-handler` is a DaemonSet with excessive permissions that can be exploited to update the properties of all worker nodes, and we call such DaemonSet *critical*

DaemonSet. The `virt-operator` is a random component with excessive permissions that can be used to obtain all secrets inside the entire cluster, including the cluster admin's secret if created, and we call such component *critical component*. By combining the excessive permissions of the critical DaemonSet and the critical component, we are able to launch an excessive permission attack that results in privilege escalation to take over the whole cluster.

It is worth pointing out that any permissions granted to the app beyond what is essential for its normal functionalities can be considered excessive. In other words, the concept of excessive permissions can vary depending on the runtime semantics of a specific application. Furthermore, the consequences of excessive permission misuse can raise issues beyond taking over the whole cluster. For instance, third-party apps are able to acquire unrelated permissions from the current app, which may also lead to other various issues.

Attack Vector Analysis. The excessive permission attack via Kubevirt is shown in Figure 5. Specifically, to launch an attack, it requires two steps. First, the attacker abuses the critical DaemonSet (i.e., `virt-handler`) to force the critical component (i.e., `virt-operator`) to run on the attacker-controlled worker node. Particularly, the service account of `virt-handler` is `kubevirt-handler`, which has a ClusterRole named `kubevirt-handler` via the `kubevirt-handler` ClusterRoleBinding. The `kubevirt-handler` ClusterRole has a "patch" verb of the "nodes" resource, which is able to patch (or modify) all the worker nodes in the entire cluster. Thus, the attacker can abuse the "patch" excessive permission to patch all other worker nodes with "node.kubernetes.io/unschedulable: NoExecute" taint. Note that the taint is a property of worker nodes that allows a worker node to repel a set of pods. As a result, by patching all other worker nodes with this taint, the `virt-operator` component will be evicted from all other worker nodes and forced to run on the attacker-controlled worker node (❶ in Figure 5).

Second, the attacker can abuse the excessive permission of the critical component (i.e., `virt-operator`) to make a privilege escalation. As described in §2.3, the service account of the `virt-operator` is `kubevirt-operator`, which has a `kubevirt-operator` ClusterRole via `kubevirt-operator` ClusterRoleBinding. The `kubevirt-operator` ClusterRole has the "list" verb of the "secrets" resource. Thus, the attacker can use the service account token of `virt-operator` to obtain all secrets in the whole cluster, including the cluster admin's secret (❷ in Figure 5). This attack leads to a privilege escalation that takes over the whole cluster.

Note that in the above process, a typical third-party app like Kubevirt has a critical DaemonSet and a critical component with excessive permissions, and these permissions are not needed by the app to exhibit its normal functionalities. Thus, if an attacker controls a worker node, he/she can abuse these excessive permissions to steal the cluster admin permission and take over the whole cluster, which results in a privilege escalation.

We report this vulnerability to the Kubevirt community. They confirm the issue and fix it by adding the resource name to restrain what secrets can be accessed by the `kubevirt-operator` service account. Further, a new CVE (i.e., CVE-2023-26484) has been assigned to us.
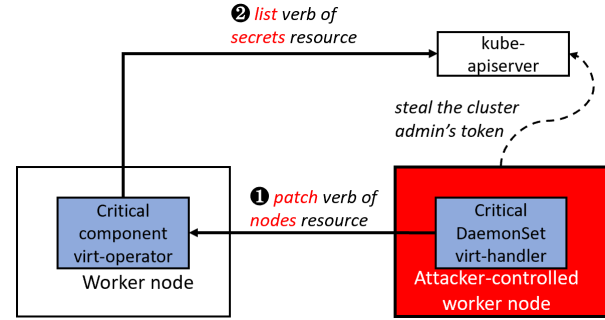

Figure 5: Excessive permission attack via Kubevirt. The `virt-operator` component is forced to run on the attacker-controlled worker node and it lists all secrets inside the entire cluster.

4 EXCESSIVE PERMISSION ATTACKING STRATEGY

To systematically identify ways to launch excessive permission attacks via third-party apps, we design three strategies based on different exploiting paths of excessive permissions, as shown in Figure 6. Particularly, there are two types of components that have excessive permissions from a third-party app, i.e., critical DaemonSet and critical component, which run on worker nodes. Furthermore, one or more third-party apps run in the same cluster. Our main attacking strategy is that an attacker can abuse the excessive permissions of the critical DaemonSets to steal the cluster admin permission directly, or chain excessive permissions of critical Daemonsets with excessive permissions of critical components from the same or another third-party app to obtain the cluster admin permission indirectly.

The attacking strategies to make excessive permission attacks via third-party apps are shown in Figure 6. Specifically, an attacker can abuse the excessive permissions of the critical DaemonSet from one third-party app to steal the cluster admin permission directly (strategy ❶). As well, an attacker can hijack the critical component from the same app (strategy ❷) or another app (strategy ❸) to force the (critical) component to run on the attacker-controlled worker node, and then he/she combines the excessive permissions of the critical DaemonSet and the critical component to steal the cluster admin permission. For each strategy, we present an example of carrying out the attacks.

4.1 Obtaining the Cluster Admin Permission Directly

Our first strategy (i.e., the strategy ❶ as shown in Figure 6) is to leverage the third-party app's critical DaemonSet to steal the cluster admin permission. Specifically, the attacker-controlled worker node has pods of DaemonSets for multiple third-party apps. If one critical DaemonSet has excessive permissions that can obtain the cluster admin permission (e.g., the "get" verb of the "secrets" resource), then the attacker can abuse the critical DaemonSet's service account to steal the cluster admin permission directly, which results in a privilege escalation.
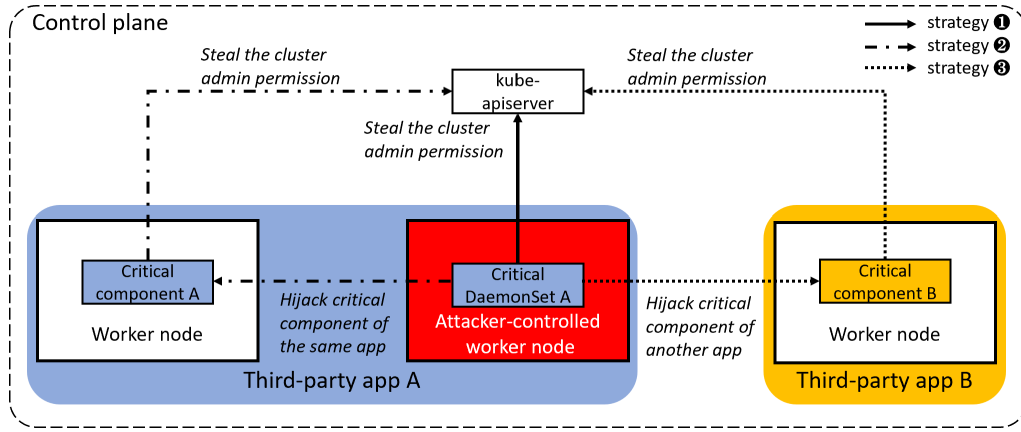
Figure 6: The attacking strategies to make excessive permission attacks via third-party apps.

For example, CubeFS[4] is a third-party app that has a critical DaemonSet `cfs-csi-node`. This critical DaemonSet's service account `cfs-csi-service-account` has a `cfs-csi-cluster-role` ClusterRole via the `cfs-csi-cluster-role-binding` ClusterRoleBinding, and this ClusterRole has the "get" verb of the "secrets" resource. Hence, an attacker-controlled worker node can use the excessive permission of the `cfs-csi-node` to obtain the cluster admin secret directly, which leads to a privilege escalation. We provide further details in §5.3.

4.2 Hijacking Critical Components of the Same App

Our second strategy (i.e., the strategy ❷ as shown in Figure 6) is to leverage the third-party app's critical DaemonSet to hijack one certain critical component of the app, and then use the critical component to get the cluster admin permission. Specifically, the attacker can abuse critical DaemonSet's excessive permissions to force the critical component of the same app to run on its own worker node. After that, it leverages the excessive permissions of the critical component to steal the cluster's admin permission indirectly, which makes an excessive permission attack.

For example, as described in §3, Kubevirt's critical DaemonSet (i.e., `virt-handler`) is able to force its critical component (i.e., `virt-operator`) to run on the attacker-controlled worker node. After that, the critical component's service account can be leveraged to get the cluster admin's token, thus it causes a privilege escalation. Besides Kubevirt, other third-party apps also have critical DaemonSets and critical components which can be combined to make similar attacks. We present more attacks in §5.3.

4.3 Hijacking Critical Components of Other Apps

In real-world scenarios, multiple third-party apps often run simultaneously in the same Kubernetes cluster. As the strategy ❸ shown in Figure 6, if excessive permissions of one third-party app are not sufficient to launch an attack, the attacker can hijack another app's critical component to force it to run on the attacker-controlled

---

[4]https://cubefs.io/

worker node, and then he/she chains the excessive permissions of different third-party apps together to perform a real attack.

For example, as described in §3, Kubevirt fixes its problem by restraining the secrets that can be accessed by the `virt-operator`. However, the critical DaemonSet of Kubevirt, i.e., `virt-handler`, can still patch other worker nodes in the cluster. If another third-party app's critical component has "get/list" verbs of the "secrets" resource, the attacker can force the app's critical component to run on the attacker-controlled worker node and exploit its excessive permissions to steal the cluster admin's secret, which results in a privilege escalation. We provide more attacks in §5.3.

5 PRACTICAL EXCESSIVE PERMISSION ATTACKS

In this section, we first present our approach to identifying attack vectors. Then we use real cases to validate the identified attack vectors and give out detailed steps for making excessive permission attacks with third-party apps. We measure all the third-party apps from the CNCF projects in our local environment, and it indicates that 51 of 153 (33.3%) CNCF projects have vulnerabilities. We measure third-party apps used by the Kubernetes services from the top four cloud vendors, including Google GKE, Amazon EKS, Azure AKS, and Alibaba Cloud ACK, and all of them have vulnerabilities in their third-party apps. The results show that our attack surface is practical and has severe consequences.

5.1 Identifying Attack Vectors

We implement our attacking strategies into a semi-automatic approach to identify attack vectors systematically, which consists of three steps: (1) installing each third-party app manually, (2) identifying critical applications automatically, and (3) launching attacks manually.

For the first step, we install each third-party app in a Kubernetes cluster manually. For the second step, we develop a script to scan and identify critical apps automatically. This script is implemented based on our strategies discussed in §4. Note that we regard an app as critical if the app has a critical DaemonSet or critical component. To make the collection, we identify critical DaemonSet by checking

Table 1: Experiment environments setup. "Distribution" means the version of Kubernetes in different environments, "VM" means the hardware configurations (including the number of CPU cores and the space of memory) for virtual machines, "Containerd" means the version of contained runtime, and "Dataset selection" means the selection source of third-party apps.

Environments	Distributions	Topology	VM	OS	Containerd	Datset selection
Local	v1.25.4	1 control plane, 2 worker nodes	4 cores, 8GB	Ubuntu 20.04	1.6.8	CNCF project lists
Google GKE	v1.24.9-gke.3200	3 worker nodes	2 cores, 4GB	Container-Optimized	1.6.20	GKE startup UI
Amazon EKS	v1.24.10-eks-48e63af	3 worker nodes	2 cores, 4GB	Amazon Linux 2 (AL2_x86_64)	1.6.19	EKS startup UI
	v1.23.16-eks-48e63af					
Azure AKS	v1.24.9	2 worker nodes	4 cores, 16GB	AKSUbuntu 18.04	1.6.18	AKS startup UI
Alibaba Cloud ACK	v1.24.6-aliyun.1	3 control planes, 3 worker nodes	4 cores, 8GB	Alibaba Cloud Linux 2.1903	1.5.13	ACK startup UI

if the DaemonSet of a third-party app has excessive permissions to steal the cluster admin permission directly, or if it can force other components to run on the attacker-controlled worker node. Further, we identify critical components by looking for components with excessive permissions that can be used to steal the cluster admin permission. For the third step, we manually launch real attacks with these identified third-party apps in the cluster and find real vulnerabilities.

For example, for the attack presented in §3.2, we install the Kubevirt app in our local cluster and run the script tool. As a result, we find that the `virt-handler` DaemonSet of the app has a "patch" verb for the "nodes" resource, and the `virt-operator` component has a "get/list" verb for the "secrets" resource. Thus, following the strategy ❷ as shown in Figure 6, we launch an excessive permission attack with the Kubevirt app, and it results in a privilege escalation. We then report this vulnerability to the Kubevirt community and they assign us a new CVE.

In summary, as for our manual efforts, it takes two weeks to design and develop our script. Subsequently, for each app, it takes about one hour to install and set up environments, followed by about three hours to manually trigger the attacks. We invest two months in completing all experiments presented in this paper.

5.2 Setting up Third-party Apps

Ethical Considerations. For the cloud environments, we intend to evaluate the attacks without causing real-world impact. The excessive permission attacks can result in privilege escalations and generate cluster-level implications in Kubernetes, which makes it unethical to conduct such attacks in multi-tenant cloud-vendor environments. Therefore, we choose to evaluate our attacks on dedicated Kubernetes clusters provided by cloud vendors such as Google GKE [73], Amazon EKS [21], Azure AKS [23], and Alibaba Cloud ACK [32]. As we are the only tenant in such clusters, we can conduct our attacks without compromising the security and privacy of other tenants.

Third-party apps setup of the local environment. In our local environment, we select the CNCF projects [37, 38] as the dataset of third-party apps to evaluate the impact of excessive permission attacks. As described in §2.2, there are three types of CNCF projects. Specifically, at the time of writing this paper, there are 20 graduated, 38 incubating, and 95 sandbox projects, respectively. We

install these projects following their official documents. The other configurations are summarized in Table 1.

Third-party apps setup of Google GKE. In Google GKE, we only select those third-party apps which can be enabled in the GKE cluster startup UI to evaluate the impact of excessive permission attacks. We select these apps instead of ones from the GKE marketplace [72] or other resources because the apps from the start-up UI have a higher probability of usage. Thus, identifying the vulnerabilities of these applications holds greater practical significance for cloud vendors.

In summary, the GKE enables 14 third-party apps, including the Config Connector, Managed Service for Prometheus, Dataplane V2, Filestore CSI Driver, Logging, HTTP load balancing, Workload Identity, Cloud Monitoring, Kube-DNS, Cloud DNS, NodeLocal DNSCache, Compute Engine Persistent Disk CSI Driver, Calico Network policy, and Anthos. We summarize other installation configurations in Table 1.

Third-party apps setup of Amazon EKS. In Amazon EKS [21], for a similar reason, we also only select those third-party apps which can be enabled in the EKS startup UI. In total, the EKS enables 13 third-party apps, including Amazon VPC CNI, CoreDNS, Kube-proxy, Amazon EBS CNI driver, ADOT, Amazon GuardDuty agent, Kubecost, Dynatrace Container Agent, Teleport OSS, Tetrate Istio Distro (TID), Upbound Universal Crossplane (UXP), Datree, and Kpow for Apache Kafka. Note that the TID can only be enabled in the EKS version up to 1.24, and the UXP can only be enabled in the EKS version up to 1.23. So we run an EKS cluster with version 1.23 to install and analyze the UXP. For TID and other third-party apps, we enable them in an EKS cluster with version 1.24. Other installation configurations are summarized in Table 1.

Third-party apps setup of Azure AKS. In Azure AKS [23], we only select those third-party apps which can be enabled in the AKS cluster startup UI. In total, AKS enables 7 third-party apps, including Network configuration Kubenet, Network configuration Azure CNI, Network policy Calico, Network policy Azure, Container Logs, Azure Policy, and Secret store CSI driver.the other setups are also summarized in Table 1.

Third-party apps setup of Alibaba Cloud ACK. The Alibaba Cloud ACK also uses third-party apps to extend the cluster's capabilities (e.g., network connection, resource monitoring, etc.). Similar to the former three vendors, we only select those third-party apps through the ACK cluster startup UI. As a result, ACK enables

Table 2: Result summary of CNCF projects. "No." means the number of identified projects.

Project type	Strategy❶ No.	Strategy❷ No.	Strategy ❸ No.	Identified/Total
Graduated	1	-	1	2/20
Incubating	2	1	13	16/38
Sandbox	2	5	26	33/95
Summary	5	6	40	51/153

13 third-party apps, i.e., Prometheus Monitoring, node-problem-detector, CSI Volume Plugin, Flannel, Terway, Log Service, NodeLocal DNSCache, CloudMonitor Agent, Nginx Ingress, MSE Ingress, ALB Ingress, Cluster Inspect, and the Dynamically Provision Volumes. The other startup configurations are listed in Table 1.

Usages of third-party apps. Note that third-party apps which assign us CVEs are widely adopted or supported by multiple companies. More specifically, Kubevirt has multiple adopters such as ARM, Nvidia, and RedHat [45]. CubeFS is used by multiple companies such as JD.com, NetEase, OPPO [54]. OpenKruise is adopted by multiple companies such as Alibaba Group, Ant Group, LinkedIn, and so on [54]. Fluid is adopted by multiple companies such as Alibaba Cloud, Tencent Cloud, Baidu AI Cloud, and so on [36]. Kubewarden is adopted by SUSE, Firesight Mining Software Corporation, and so on [46]. Open Cluster Management is supported by Alibaba Cloud, Ant Group, Redhat, and so on[7]. Clusternet is modified and adopted by Tencent Cloud [30, 31]. OpenFeature is adopted by multiple organizations such as Dynatrace, Ebay, Proofpoint, and so on [53].

Additionally, all other third-party apps in our experiments are sourced from CNCF project lists and cloud environments, which are widely used and supported by big companies. We do not disclose them due to ethical considerations.

5.3 Identifying Vulnerabilities of CNCF Projects

To demonstrate the effectiveness of excessive permission attacks on open-source projects, we use our tool to scan third-party apps from the CNCF project lists. Table 2 presents the result summary of CNCF projects. More specifically, we identify 2, 16, and 33 third-party apps in graduated, incubating, and sandbox projects, respectively. Overall, there are 51 of all 153 (33.3%) CNCF projects that have potential security risks of making excessive permission attacks. Among them, 5 projects involve strategy ❶ in Figure 6, 6 projects involve strategy ❷ in Figure 6, and 40 projects involve strategy ❸ in Figure 6. In addition, the identified projects cover all three attacking strategies presented in §4.

In the following, to present how to carry out excessive permission attacks following our strategies in a real scenario, we launch attacks with typical apps which assign us CVEs to illustrate the details. Note that eight CVEs are assigned to us and we have discussed the attack via Kubevirt in §3.2. Next, we present the attacks via other seven third-party apps. The result summary of excessive permission attacks via typical third-party apps in our local environment is shown in Table 3.

5.3.1 Excessive Permission Attack with CubeFS. The first attack involves exploiting the excessive permissions of the CubeFS' critical DaemonSet via the strategy ❶ in Figure 6. This allows an attacker

to obtain the cluster admin's token, thereby it carries out a privilege escalation via CubeFS.

CubeFS[4] is an open-source cloud-native file storage system, which is hosted by the CNCF as an incubating project. As shown in Table 3, the CubeFS has a `cfs-csi-node` critical DaemonSet, which runs a pod on each node of a Kubernetes cluster. The critical DaemonSet uses a service account `cfs-csi-service-account`, which is assigned a `cfs-csi-cluster-role` ClusterRole via a ClusterRoleBinding named `cfs-csi-cluster-role-binding`. This ClusterRole has the "get/list" verbs of the "secrets" resource. Thus, on an attacker-controlled worker node, the attacker can abuse the excessive permissions of the critical DaemonSet to obtain all secrets directly, including the cluster admin's secret if created, and escape from a worker node to take over the whole cluster. We report this vulnerability to the CubeFS community, and they fix this issue and assigned us a new CVE (i.e., CVE-2023-30512).

5.3.2 Excessive Permission Attack via OpenKruise. The second attack involves exploiting excessive permissions in OpenKruise's critical DaemonSet via the strategy ❶ in Figure 6. Similar to the CubeFS, OpenKruise has one critical DaemonSet, which can be leveraged by the attacker to get the cluster admin's token and result in a privilege escalation.

OpenKruise[5] is a CNCF incubating project, which provides open-source automated management of large-scale applications in Kubernetes. As shown in Table 3, the OpenKruise has a `kruise-daemon` critical DaemonSet, which runs a pod on each node of a Kubernetes cluster. The `kruise-daemon` DaemonSet uses the `kruise-daemon` service account, which is assigned the `kruise-daemon-role` ClusterRole via the `kruise-daemon-rolebinding` ClusterRoleBinding. This ClusterRole has "get/list" verbs of the "secrets" resource. Thus, similar to the excessive permission attacks via CubeFS, the attacker can leverage these excessive permissions to get/list all secrets in the whole cluster (including the cluster admin's token if created), and thus it makes a privilege escalation. We report this vulnerability to the OpenKruise community and they fix this issue. A new CVE CVE-2023-30617 has been assigned to us.

5.3.3 Excessive Permission Attack with Fluid. The third attack involves an excessive permission vulnerability in Fluid via the strategy ❷ in Figure 6. An attacker can exploit the excessive permission of the Fluid's critical DaemonSet to hijack the critical component of Fluid to run on the attacker-controlled worker node. After that, he/she leverages the excessive permission of the critical component to retrieve all secrets in the whole cluster, including the cluster admin's token if it is created, which results in a privilege escalation.

Fluid[6] is a Kubernetes-native distributed dataset orchestrator, which is a CNCF sandbox project. As shown in Table 3, the Fluid has a `csi-nodeplugin-fluid` critical DaemonSet. Additionally, the Fluid has a `fluid-webhook` critical component, which is a Kubernetes Deployment that runs on worker nodes randomly.
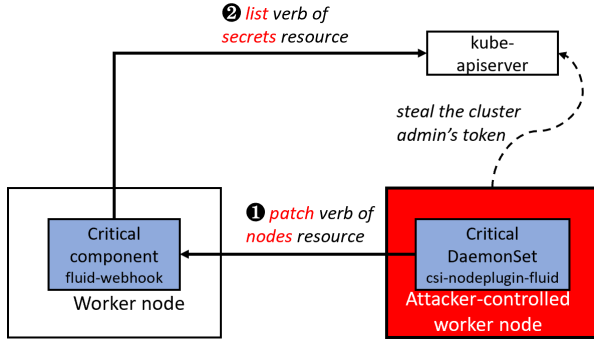
For the `csi-nodeplugin-fluid` critical DaemonSet, its service account is `fluid-csi`, which has a `fluid-csi-plugin` ClusterRole via `fluid-csi-plugin` ClusterRoleBinding. The `fluid-csi-plugin` ClusterRole has a "patch" verb of the "nodes" resource, and this excessive

---

[5]https://openkruise.io/
[6]https://fluid-cloudnative.github.io/

Table 3: Result summary of excessive permission attacks via typical third-party apps in our local environment. "Component name" means the name of critical DaemonSet or critical component, and "Type" means the component's type.

App name	CVE-ID	Component name	Type	Service account	ClusterRole	Excessive permission
CubeFS	2023-30512	cfs-csi-node	DaemonSet	cfs-csi-service-account	cfs-csi-cluster-role	get/list secrets
OpenKruise	2023-30617	kruise-daemon	DaemonSet	kruise-daemon	kruise-daemon-role	get/list secrets
Kubevirt	2023-26484	virt-handler	DaemonSet	kubevirt-handler	kubevirt-handler	list/patch nodes
		virt-operator	Deployment	kubevirt-operator	kubevirt-operator	get/list secrets
Fluid	2023-30840	csi-nodeplugin-fluid	DaemonSet	fluid-csi	fluid-csi-plugin	patch nodes
		fluid-webhook	Deployment	fluid-webhook	fluid-webhook	get/list secrets
Kubewarden	2023-22645	kubewarden-controller	Deployment	kubewarden-controller	kubewarden-controller-manager-cluster-role	get/list secrets
Open Cluster Management	2023-2250	cluster-manager-registration-controller	Deployment	cluster-manager-registration-controller-sa	open-cluster-management: cluster-manager-registration:controller	escalate/bind verbs of clusterroles
		cluster-manager	Deployment	cluster-manager	cluster-manager	escalate/bind verbs of clusterroles, get/list secrets
Clusternet	2023-30622	clusternet-hub	Deployment	clusternet-hub	clusternet:hub	* verbs of *.* resources
OpenFeature	2023-29018	open-feature-operator-controller-manager	Deployment	open-feature-operator-controller-manager	open-feature-operator-manager-role	list/update verbs of clusterrolebindings

Figure 7: Excessive permission attack via Fluid. The attacker can leverage the `cfs-nodeplugin-fluid` critical DaemonSet to force the `fluid-webhook` critical component to run on the attacker-controlled worker node, which can be leveraged to get the cluster admin's token.
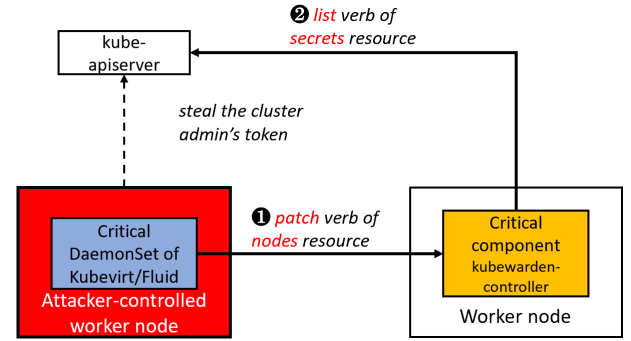
Figure 8: The potential risk of Kubewarden. The attacker can leverage the DaemonSet of other apps to force the Kubewarden's critical component to run on the attacker-controlled node, then he/she can use its excessive permission to get the cluster admin's token.

permission can be leveraged to patch other nodes with unschedulable taint. For the `fluid-webhook` critical component, its service account is `fluid-webhook`, and it has a `fluid-webhook` ClusterRole through the `fluid-webhook-clusterrolebinding` ClusterRoleBinding. The ClusterRole has "get/list" verbs of the "secrets" resource, and this excessive permission can be used to get all secrets in the whole cluster.

Therefore, as shown in Figure 7, similar to the Kubevirt attack we discussed in §3.2, an attacker can exploit the excessive permission of the `csi-nodeplugin-fluid` to patch other nodes with "node.kubernetes.io/unschedulable: NoExecute". As a result, the `fluid-webhook` will be evicted from its original node and forced to run on the attacker-controlled worker node (❶ in Figure 7). After that, the attacker can abuse the excessive permissions of `fluid-webhook` to obtain all secrets (including the cluster admin's

token) in the entire cluster (❷ in Figure 7). We report this vulnerability to the Fluid community. They fix this problem and assign us a new CVE (i.e., CVE-2023-30840).

5.3.4 A Potential Risk of Kubewarden. The fourth case is a potential risk of Kubewarden[3], which is a CNCF sandbox project. First, Kubewarden's critical component (i.e., `kubewarden-controller`) can be forced to run on the attacker-controlled worker node via other third-party apps. After that, the excessive permissions of this critical component can be abused to steal the cluster admin's token, which involves the strategy ❸ in Figure 6.

For the first step, as shown in Figure 8, an attacker can leverage other third-party apps' critical DaemonSets (e.g., the critical DaemonSets of Kubevirt or Fluid) to patch all other nodes and make them unschedulable. The Kubewarden's critical component will be evicted and forced to run on the attacker-controlled worker node (❶ in Figure 8).

For the second step, Kubewarden has a `kubewarden-controller` critical component, which is a Kubernetes Deployment and runs on worker nodes randomly. As shown in Table 3, this critical component has a `kubewarden-controller` service account, which is assigned a `kubewarden-controller-manager-cluster-role` Cluster-Role via the `kubewarden-controller-manager-cluster-role` Cluster-RoleBinding. This ClusterRole has "get/list" verbs of the "secrets" resource. Thus, an attacker can abuse these excessive permissions to obtain all secrets in the entire cluster, which results in a privilege escalation (❷ in Figure 8). We report this vulnerability to the Kubewarden community. They fix this problem and give us a new CVE (i.e., CVE-2023-22645).

5.3.5 A Potential Risk of Open Cluster Management. The fifth case is a potential risk of Open Cluster Management (OCM for short). OCM[7] is a CNCF sandbox project which focuses on multi-cluster and multi-cloud scenarios for Kubernetes apps. Similar to the Kubewarden attack, it also needs other third-party apps to force OCM's critical components to run on the attacker-controlled worker node. After that, the two critical components of OCM can be leveraged to steal high-critical permissions in the Kubernetes cluster, and it involves the strategy ❸ in Figure 6.

The first step of this attack is the same as the Kubewarden. After the first step, there are two OCM critical components, one of them is `cluster-manager-registration-controller` Deployment, and the other is `cluster-manager` Deployment. They both have excessive permissions.

For the `cluster-manager-registration-controller` critical component, it has a `cluster-manager-registration-controller-sa` service account. This service account is bounded with a ClusterRole `open-cluster-management:cluster-manager-registration:controller` via a same-name ClusterRoleBinding. The ClusterRole has the "escalate/bind" verbs of the "clusterroles" resource. The "escalate" verb makes the attacker can edit a ClusterRole that he/she is already bounded to, including granting high-critical permissions to his/her own ClusterRole. The "bind" verb allows the attacker to create a ClusterRoleBinding resource even if they do not have the permissions for the targeted ClusterRole. Thus, these excessive permissions can both allow an attacker to grant the cluster admin to his/her own account in the Kubernetes cluster, and it leads to privilege escalations.

For the critical component `cluster-manager`, besides the "escalate/bind" verbs of the "clusterroles" resource, it also has "get/list" verbs of the "secrets" resource, which can be used to retrieve all secrets in the whole cluster and also lead to a privilege escalation. We report these vulnerabilities to the OCM community. They fix these problems and assign us a new CVE (i.e., CVE-2023-2250).

5.3.6 A Potential Risk of Clusternet. The sixth case is a potential risk of Clusternet. Clusternet[8] is a CNCF sandbox project which helps users manage multiple Kubernetes clusters. Similar to the former two attacks, it also needs other third-party apps to force Clusternet's critical component to run on the attacker-controlled worker node. After that, its excessive permissions can be abused to

steal the cluster admin's secrets in the Kubernetes cluster, and it involves the strategy ❸ in Figure 6.

The first step of this attack is the same as the former two attacks. For the second step, Clusternet has a `clusternet-hub` critical component. As shown in Table 3, the `clusternet-hub` has a `clusternet-hub` service account. This service account has a `clusternet:hub` Cluster-Role via the `clusternet:hub` ClusterRoleBinding. The ClusterRole has "*" verbs of "*.*" resources. The "*" verb means all Kubernetes verbs of resources, including the "list" verb. The "*.*" resources contain multiple Kubernetes resources, including the "secrets" resource. Thus, an attacker can abuse this excessive permission to list all secrets in the whole cluster, including the cluster admin's token if created, which causes a privilege escalation. We report this problem to the Clusternet community. They fix it and assign us a new CVE (i.e., CVE-2023-30622).

5.3.7 A Potential Risk of OpenFeature. The seventh case represents a potential risk in OpenFeature. OpenFeature[9] is a CNCF sandbox project that enables a robust feature flag ecosystem using cloud-native technologies. Similar to the former three attacks, the OpenFeature's critical component can be forced to run on the attacker-controlled worker node, and the critical component has excessive permissions to escalate the privileges of any service account in the whole cluster, which involves strategy ❸ in Figure 6.

The first step of this attack is the same as the former three attacks. For the second step, the OpenFeature has a critical component named `open-feature-operator-controller-manager`. This critical component has a `open-feature-operator-controller-manager` service account. This service account is granted a ClusterRole called `open-feature-operator-manager-role` via a ClusterRoleBinding called `open-feature-operator-manager-rolebinding`. This ClusterRole has "list/update" verbs for the "clusterrolebindings" resource. Thus, an attacker can modify all existing ClusterRoleBindings with any service accounts in the whole cluster, which makes escalate the privileges of any service account. We report this vulnerability to the OpenFeature community. They fix it and give us a new CVE (i.e., CVE-2023-29018).

5.4 Identifying Vulnerabilities of Cloud Vendors

To further understand the prevalence of excessive permission attacks in business environments, we analyze the third-party apps used by Google GKE, Amazon EKS, Azure AKS, and Alibaba Cloud ACK. All of them have vulnerabilities of making excessive permission attacks. The result summary of excessive permissions in cloud environments is shown in Table 4.

5.4.1 Excessive Permissions in Google GKE. Google GKE can install 14 third-party apps during its startup, and three of them (i.e., Calico Network Policy, Config Connector, and Anthos) are vulnerable to excessive permission attacks. The results are summarized in Table 4.

The Calico Network Policy app has a `calico-node` critical DaemonSet, which has an excessive permission that allows the "patch" verb of the "nodes/status" resource. This excessive permission can be exploited to force critical components of other third-party apps to run on the attacker-controlled worker node. For example, the `calico-node` critical DaemonSet can patch other nodes/status with

---

[7]https://open-cluster-management.io/
[8]https://clusternet.io/

[9]https://openfeature.dev/

Table 4: Result summary of excessive permissions in cloud environments. "Vendor" means the name of cloud vendors, "App" means the name of third-party apps, and "Type" means the type of components.

Vendor	App	Component	Type	Excessive permission	Report channel	Report status
Google GKE	Calico Network Policy	calico-node	DaemonSet	patch nodes/status	Google Bug Hunters	Confirmed
	Config Connector	configconnector-operator	StatefulSet	get/list secrets		
	Anthos	istio-cni-node	DaemonSet	delete pods		
		activator	Deployment	get/list secrets		
		autoscaler				
		controller				
		metrics				
		webhook				
Amazon EKS	Amazon VPC CNI	aws-node	DaemonSet	update nodes	AWS Security	Confirmed
	Tetrate Istio Distro	istiod	Deployment	get/list secrets		
	Upbound Universal Crossplane	crossplane	Deployment	get/list secrets		
Azure AKS	Secret store CSI driver	aks-secrets-store-csi-driver	DaemonSet	get/list secrets	Microsoft Security Response Center (MSRC)	Confirmed
	Azure Policy	gatekeeper-audit	Deployment	get/list *.* resources		
		gatekeeper-controller				
Alibaba Cloud ACK	Prometheus Monitoring	node-exporter	DaemonSet	get/list secrets	Alibaba Security Response Center (ASRC)	Confirmed
		arms-prometheus-ack-arms-prometheus	Deployment	get/list secrets		
		kube-state-metrics	Deployment	get/list secrets		
	CSI Volume Plugin	csi-plugin	DaemonSet	get/list secrets		
	Node-problem-detector	ack-node-problem-detector-daemonset	DaemonSet	* verbs of nodes		
	Flannel	kube-flannel-ds	DaemonSet	patch nodes/status		
	Terway	terway-eniip	DaemonSet	update/patch nodes		
	Nginx Ingress	nginx-ingress-controller	Deployment	list secrets		
	ALB Ingress	alb-ingress-controller	Deployment	get/list secrets		
	MSE Ingress	ack-mse-ingress-controller	Deployment	get/list secrets		
	CloudMonitor Agent	alicloud-monitor-controller	Deployment	get/list * resources		

"capacity: CPU 0", which means other nodes have no CPU resources. Thus, the critical components of other apps will be forced to run on the attacker-controlled worker node after being deleted, which involves the first step of the strategy ❸ in Figure 6.

Regarding the Config Connector app, its StatefulSet [16] (a set of pods each of which has a sticky ID) `configconnector-operator` has the "get/list" verbs of the "secrets" resource. These excessive permissions can be exploited to retrieve all secrets in the whole cluster, including the cluster admin's secret if created. Thus, this critical component can be used to carry out an excessive permission attack after being forced to run on the attacker-controlled worker node, which involves the second step of the strategy ❸ in Figure 6.

For the Anthos third-party app, its `istio-cni-node` critical DaemonSet can be used to delete pods across the whole cluster. This makes the critical components possibly rescheduled and run on the attacker-controlled worker node, which involves the first step

of strategy ❸ in Figure 6. Additionally, its `activator`, `autoscaler`, `controller`, `metrics`, and `webhook` critical component has the "get/list" verbs of the "secrets" resource, and they all involve the second step of strategy ❸ in Figure 6.

5.4.2 Excessive Permissions in Amazon EKS. Amazon EKS can enable 13 third-party apps during its startup, and three of them, namely Amazon VPC CNI, Tetrate Istio Distro (TID), and Upbound Universal Crossplane (UXP) are vulnerable to excessive permission attacks that can result in privilege escalations. The details are summarized in Table 4.

More specifically, the Amazon VPC CNI app has a `aws-node` critical DaemonSet that has the "update" verb of the "nodes" resource. This can be leveraged to update nodes and force the critical components of other apps to run on the attacker-controlled worker node, which involves the first step of strategy ❸ in Figure 6. The TID app

has a `istiod` critical component, and the UXP has a `crossplane` critical component. They both have the "get/list" verbs of the "secrets" resource. Thus, they both involve the second step of strategy ❸ in Figure 6.

5.4.3 Excessive Permissions in Azure AKS. Azure AKS can enable 7 third-party apps to enhance the Kubernetes cluster's capabilities. The Secret store CSI driver app and Azure Policy app suffer from excessive permission attacks. The Secret store CSI driver app has a `aks-secrets-store-csi-driver` critical DaemonSet. It has "get/list" verbs of the "secrets" resource, which can be abused to obtain all secrets inside the whole cluster directly, and it involves the strategy ❶ in Figure 6.

As for the Azure Policy app, it has the `gatekeeper-audit` and the `gatekeeper-controller` critical components. They both have "get/list" verbs of "*.*" resources. These excessive permissions can be abused to retrieve all secrets in the whole cluster, which involves the second step of strategy ❸ in Figure 6.

5.4.4 Excessive Permissions in Alibaba Cloud ACK. Alibaba Cloud ACK can enable 13 third-party apps during its startup. However, the ACK becomes susceptible to multiple excessive permission attacks with 9 of them. The results are summarized in Table 4.

The Prometheus Monitoring app and CSI Volume Plugin app can be leveraged to make excessive permission attacks. Specifically, the `node-exporter` critical DaemonSet of the Prometheus monitoring app and `csi-plugin` critical DaemonSet of the CSI Volume plugin app both have "get/list" verbs of the "secrets" resource, which can be abused to get all secrets in the whole cluster directly. And they both involve the strategy ❶ in Figure 6. The Prometheus Monitoring app also carries out two critical components `arms-prometheus-ack-arms-prometheus` and `kube-state-metrics`. They both have the "get/list" verbs of the "secrets" resource, and they involve the second step of strategy ❸ in Figure 6.

The node-problem-detector, Flannel, and Terway apps all have critical DaemonSets which have excessive permissions to modify the "nodes" or "nodes/status" resource. These enable them to force critical components of other third-party apps to run on the attacker-controlled worker node, involving the first step of strategy ❸ in Figure 6.

Additionally, the critical components of the Nginx Ingress, ALB Ingress, MSE Ingress, and CloudMonitor Agent apps have excessive permissions which can be used to retrieve all secrets in the cluster. And they all trigger the second step of strategy ❸ in Figure 6.

5.5 Responsible Disclosure

We report our findings to the relevant communities and cloud vendors through the proper channels. We obtain multiple CVEs and a bounty.

Report to communities. For the identified 51 projects from all 153 CNCF projects, we report all their issues to the relevant communities. As shown in Table 5, of the 51 identified projects, 32 of them respond to us, and the other 19 have no responses yet. Of the 32 responded projects, 10 of them confirm our reports and fix their source code, and 8 CVEs are assigned to us. 22 of 32 responded projects confirm our reports and we are still communicating with

Table 5: Result summary of Identified CNCF projects.

Identified projects		51	
No Response		19	
Responded	32	Fixed (CVE)	8
		Fixed (no CVE)	2
		Pending	22

them. For ethical considerations, we only post the confirmed vulnerabilities with CVE numbers and refrain from publicly disclosing other identified projects.

Report to vendors. We report the vulnerabilities of cloud environments to related vendors through the proper channels, and they all confirm our findings. Specifically, we report our findings about Google GKE through the Google Bug Hunters[10]. We write an e-mail to the AWS Security Team[11] to report the problems of Amazon EKS. For the Azure AKS issues, we submit a vulnerability report to the Microsoft Security Response Center (MSRC)[12]. For the Alibaba Cloud ACK problems, we submit a report to the Alibaba Security Response Center (ASRC)[13].

All four vendors confirm the issues we reported, and they promise to fix them. For the Google GKE, the Google Bug Hunters acknowledge our findings and give us a bounty. More specifically, they think the exploitation likelihood is medium, and the issue is qualified as an abuse-related methodology with medium impact. They say: *"Google Vulnerability Reward Program panel has decided to issue a reward of $1337.00 for your report."*. We have decided to donate this bounty. For the Amazon EKS, the Amazon Security Team confirms our findings and they are fixing them. They say: *"We are currently in the process of implementing a fix for the issue."* For the Azure AKS, the MSRC also confirms our reports and will fix these problems. They say: *"A fix for this issue will be considered in a future version of this product as a defense in depth."* For the Alibaba Cloud ACK, the ASRC confirm our findings and they are fixing these issues. They say: *"Thanks for your report. We have checked the settings and will fix them."*

6 MITIGATION DISCUSSION

This section presents several actionable suggestions to mitigate the risks based on our discussions with related communities. Note that these methods are specific to certain third-party apps, and what works for one may not work for another. Additionally, some of these methods may have side effects on the normal functionalities of the apps and require a trade-off between security and performance. It is a significant undertaking to resolve these problems fundamentally and it possibly requires collaboration between the vendors of third-party apps.

Removing unnecessary permissions. The third-party app vendors should review and remove unnecessary permissions of the related ClusterRoles and service accounts. If the existing service

---

[10]https://bughunters.google.com/
[11]aws-security@amazon.com
[12]https://www.microsoft.com/en-us/msrc
[13]https://asrc.alibaba.com/

account's ClusterRole has excessive permissions which are not required by the app's functionalities, those permissions should be removed. However, once the permissions required for the normal operation of third-party apps are removed, the application may crash. The maintainers of apps need to review the apps' source code carefully before applying this method.

For example, for those attacks presented in our paper, Open-Feature removed the unnecessary "update" verb of other "ClusterRoleBinding" resources besides the only required one. CubeFS, Fluid, Kubewarden, and Open Cluster Management (OCM) also fixed their problems by removing unnecessary permissions. Besides, Clusternet removed the `clusternet:hub` ClusterRole which has excessive permissions directly.

Using more complex designs for service accounts. Another suggestion to mitigate the risks is to use more complex service accounts. For third-party apps with multiple components that require different permissions, it is more secure to use multiple service accounts with varying permission levels than a single service account with excessive permissions. However, the service account which is given excessive permissions may be necessary for its operations. Besides, it may require additional work to use more complex service accounts for the third-party app's maintainers.

Using RoleBinding to remove cluster efforts. As the RoleBinding only gives permissions within a specific Kubernetes namespace whereas ClusterRoleBinding grants those permissions for the whole cluster, it can also restrain the related services account with RoleBinding to only access resources inside a specific namespace. Nevertheless, some third-party apps need to access resources in multiple Kubernetes namespaces where functionalities need to be consumed. This means locking down with Rolebinding would be difficult and cannot be generalized since the namespaces requiring resource consumption may not be known in advance.

For example, Kubewarden used rolebinding and restricted access of the `kubewarden-controller-manager-cluster-role` to only those secrets in the "kubewarden" namespace. The OCM also leveraged rolebinding to eliminate cluster efforts.

Using accurate resource names. Using accurate resource names is an effective way to mitigate the risks, particularly for critical resources such as secrets. For example, the Kubevirt app uses the secret name to restrain the secrets that can be accessed by the `kubevirt-operator` ClusterRole. However, similar to other methods of mitigating the risks, it may impact some features of specific apps by removing any access to resources via specific resource names.

7 RELATED WORK

In this section, we present the studies that are related to Kubernetes security and container security.

7.1 Kubernetes Security

Attacks in Kubernetes. There are multiple works on attacking Kubernetes. Sushring et al. confirm attackers can launch co-residency attacks on a victim application from inside state-of-the-art containers running in the same Kubernetes cluster [64]. Shamim et al. study attacks under the scenario of violating Kubernetes security best practices [62]. Ronen et al. propose a DDoS attack against

Kubernetes auto-scaling [26]. Pecka et al. research the attack methods in the DevOps pipeline with a Kubernetes environment [56]. Rahman et al. analyze the misconfigurations in Kubernetes manifests [57]. Zeng et al. propose a full-stack vulnerability analysis of the cloud-native platform, which includes Kubernetes [78].

Defenses in Kubernetes. There are also works on securing Kubernetes. Kong et al. propose a secure container deployment strategy to defend against co-resident attacks in container cloud [44]. Karn et al. realize the detection of malicious cryptomining software running in Kubernetes cluster by monitoring Linux system calls [43]. Baarzi et al. implement a system for defending application-layer DoS attacks against microservices [25]. Haque et al. propose a scheme to automate the security configuration of container orchestrator [42]. Blaise et al. propose a methodology for evaluating the security and extracting the risk mode of a Helm Chart deployment [27].

To the best of our knowledge, our work is the first to systematically exploit the excessive permission of Kubernetes' third-party apps. Our work further complements previous research efforts on understanding the security of container orchestration systems.

7.2 Container Security

Attacks in containers. The security of the container is always an important research area. Luo et al. study the convert container channels of Docker that cause information leakage [50]. Gao et al. study the risk of information leakage caused by the /proc and /sys file systems in the container [39]. Lin et al. prove that kernel vulnerabilities can break through the isolation mechanism of containers [49]. Duarte et al. study the vulnerability of Docker by static code analysis of Docker patch code [35]. Yang et al. systematically identify the abstract resource attack surface in the container [77]. Haq et al. give a security analysis of Docker containers for the ARM architecture [41]. However, they target co-host containers. The new attack surface that exists in Kubernetes third-party apps is out of their scope.

Defenses in containers. In addition, the security enhancement for containers also receives extensive researches. Lei et al. design a security mechanism that can minimize the available system calls in the container according to the container image [47]. Sun et al. design a security namespace that enables autonomous security control for containers [66]. Suneia et al. research the security implementation in container fusion scenario [67]. Nam et al. design a novel security-enforcement network stack for containers [52]. Brady et al. implement a system for detecting malicious container images [28]. However, these papers mainly focus on securing the native container, not the third-party apps of the Kubernetes cluster, and they can not defend against our attack.

Unlike those previous works, our work shows that the excessive permissions of third-party apps in Kubernetes can be leveraged to take over the whole cluster. We are the first to systematically research the new attack surface and measure the security impacts by exploiting these attack surfaces in multiple real deployments.

7.3 Android Permission Security

Android permission Attacks Excessive permissions given to apps can also lead to security vulnerabilities in Android. For example,

Xing et al. discover that a malicious app can escalate a set of privilege permissions via OS upgrading [76]. Aafer et al. show that a malicious app is able to leverage the gap carried out by the hanging attribute references to acquire critical system capabilities [1]. Yousra et al. identify and explore the inconsistent security configurations in custom Android ROMs [2]. Gorski et al. systematically identify permission re-delegation vulnerabilities within Android's system services [40]. Tuncay et al. exploit false transparency to deceive users into granting sensitive permissions to malicious apps [74]. Li et al. reveal the shortcoming of Android custom permissions that can be exploited to make privilege escalation [48]. Aldoseri et al. demonstrate two vulnerabilities of Andriod app components that can affect the security and privacy guarantees of recent Android versions [3]. However, the Android permission attacks target the application-level permission model, whereas our attacks exploit the cluster-level access control mechanism in Kubernetes to gain unauthorized privileges.

Securing Android permission attacks. There are multiple works on securing permission attacks in Android. For example, Sanz et al. systematically detect malware in Android by permission usage analysis [60]. Arp et al. make a broad static analysis to identify malicious applications directly on the smartphone [5]. Wijesekera et al. propose a machine-learning approach that exploits users' past permission decisions to predict future decisions when permissions are used [75]. Arora et al use permission pairs to detect Android malware [4]. Shen et al. explored what extra information that systems can provide to help users make more informed permission decisions [63]. However, these defense methods in Andriod are implemented in a single operation system through manifest declarations and runtime user consent, whereas mitigating our attacks in Kubernetes requires a security mechanism enforced throughout the entire cluster and targeting the Kubernetes APIs. Therefore, referring to the security mechanism of Android to defend against our attack requires more effort.

8 CONCLUSION

This paper reveals that multiple third-party apps in the Kubernetes cluster are granted excessive permissions. Such permissions can be exploited to make attacks with severe consequences (e.g., taking over the whole cluster and resulting in privilege escalation), which we termed as *excessive permission attacks*. To systematically explore the paths of excessive permission attacks, we design three strategies based on different ways to abuse excessive permissions. We evaluate the impact by analyzing the CNCF projects in the local cluster, and 51 CNCF projects are vulnerable to excessive permission attacks. Furthermore, We analyze the excessive permissions of third-party apps in Google GKE, Amazon EKS, Azure AKS, and Alibaba Cloud ACK, and all four cloud vendors have excessive permissions in the third-party apps used by their Kubernetes clusters. We report all our findings to related communities and cloud vendors, and eight CVEs and a bounty are given to us. Finally, we provide several actionable suggestions to mitigate the risks.

ACKNOWLEDGMENTS

The authors would like to thank our shepherd and reviewers for their insightful comments. Those comments helped to reshape this

REFERENCES

[1] Yousra Aafer, Nan Zhang, Zhongwen Zhang, Xiao Zhang, Kai Chen, XiaoFeng Wang, Xiaoyong Zhou, Wenliang Du, and Michael Grace. 2015. Hare hunting in the wild android: A study on the threat of hanging attribute references. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1248–1259.

[2] Yousra Aafer, Xiao Zhang, and Wenliang Du. 2016. Harvesting Inconsistent Security Configurations in Custom Android {ROMs} via Differential Analysis. In *25th USENIX Security Symposium (USENIX Security 16)*. 1153–1168.

[3] Abdulla Aldoseri, David Oswald, and Robert Chiper. 2022. A Tale of Four Gates: Privilege Escalation and Permission Bypasses on Android Through App Components. In *European Symposium on Research in Computer Security*. Springer, 233–251.

[4] Anshul Arora, Sateesh K Peddoju, and Mauro Conti. 2019. Permpair: Android malware detection using permission pairs. *IEEE Transactions on Information Forensics and Security* 15 (2019), 1968–1982.

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

[6] Kubernetes Authors. 2022. Kubernetes. https://kubernetes.io/.

[7] Kubernetes Authors. 2023. Case Studies. https://kubernetes.io/case-studies.

[8] Kubernetes Authors. 2023. Control Plane. https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane.

[9] Kubernetes Authors. 2023. DaemonSet. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/.

[10] Kubernetes Authors. 2023. Deployment. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[11] Kubernetes Authors. 2023. Namespaces. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/.

[12] Kubernetes Authors. 2023. Role and ClusterRole. https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole.

[13] Kubernetes Authors. 2023. Secrets. https://kubernetes.io/docs/concepts/configuration/secret/.

[14] Kubernetes Authors. 2023. Securing a Cluster. https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/.

[15] Kubernetes Authors. 2023. Service Accounts. https://kubernetes.io/docs/concepts/security/service-accounts/.

[16] Kubernetes Authors. 2023. StatefulSets. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/.

[17] Kubernetes Authors. 2023. Using RBAC Authorization. https://kubernetes.io/docs/reference/access-authn-authz/rbac/.

[18] Yuval Avrahami. 2021. Finding Azurescape – Cross-Account Container Takeover in Azure Container Instances. https://unit42.paloaltonetworks.com/azure-container-instances/.

[19] AWS. 2023. Amazon EKS add-ons. https://docs.aws.amazon.com/eks/latest/userguide/eks-add-ons.html.

[20] AWS. 2023. Isolating tenant workloads to specific nodes. https://aws.github.io/aws-eks-best-practices/security/docs/multitenancy/#isolating-tenant-workloads-to-specific-nodes.

[21] AWS. 2023. What is Amazon EKS? https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html.

[22] Azure. 2023. Add-ons, extensions, and other integrations with Azure Kubernetes Service. https://learn.microsoft.com/en-us/azure/aks/integrations.

[23] Azure. 2023. Azure Kubernetes Service (AKS). https://azure.microsoft.com/en-us/products/kubernetes-service.

[24] Azure. 2023. Best practices for advanced scheduler features in Azure Kubernetes Service (AKS). https://learn.microsoft.com/en-us/azure/aks/operator-best-practices-advanced-scheduler.

[25] Ataollah Fatahi Baarzi, George Kesidis, Dan Fleck, and Angelos Stavrou. 2020. Microservices made attack-resilient using unsupervised service fissioning. In *Proceedings of the 13th European workshop on Systems Security*. 31–36.

[26] Ronen Ben David and Anat Bremler Barr. 2021. Kubernetes Autoscaling: YoYo Attack Vulnerability and Mitigation. *arXiv e-prints* (2021), arXiv–2105.

[27] Agathe Blaise and Filippo Rebecchi. 2022. Stay at the Helm: secure Kubernetes deployments via graph generation and attack reconstruction. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 59–69.

[28] Kelly Brady, Seung Moon, Tuan Nguyen, and Joel Coffman. 2020. Docker container security in cloud computing. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 0975–0980.

[29] ALESSANDRO BRUCATO. 2023. Detecting and mitigating CVE-2022-42889 a.k.a. Text4shell. https://sysdig.com/blog/cve-2022-42889-text4shell/.

[30] Tencent Kubernetes Engine Distributed Cloud Center. 2023. Tencent Kubernetes Engine Distributed Cloud Center Operation Guide Product Documentation. https://main.qcloudimg.com/raw/document/intl/product/pdf/1144_45541_en.pdf.

[31] Tencent Kubernetes Engine Distributed Cloud Center. 2023. Tencent Kubernetes Engine Distributed Cloud Center Product Introduction Product Documentation. https://main.qcloudimg.com/raw/document/intl/product/pdf/1144_45535_en.pdf.

[32] Alibaba Cloud. 2023. Alibaba Cloud Container Service for Kubernetes (ACK). https://www.alibabacloud.com/product/kubernetes.

[33] Alibaba Cloud. 2023. App Marketplace. https://www.alibabacloud.com/help/en/container-service-for-kubernetes/latest/app-marketplace.

[34] CNCF. 2023. ANNUAL SURVEY 2022. https://www.cncf.io/reports/cncf-annual-survey-2022/.

[35] Ana Duarte and Nuno Antunes. 2018. An empirical study of docker vulnerabilities and of static code analysis applicability. In *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 27–36.

[36] Fluid. 2023. Adopters of Fluid. https://github.com/fluid-cloudnative/fluid/blob/master/ADOPTERS.md#adopters-of-fluid.

[37] Cloud Native Computing Fundation. 2023. GRADUATED AND INCUBATING PROJECTS. https://www.cncf.io/projects/.

[38] Cloud Native Computing Fundation. 2023. SANDBOX PROJECTS. https://www.cncf.io/sandbox-projects/.

[39] Xing Gao, Benjamin Steenkamer, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. 2018. A study on the security implications of information leakages in container clouds. *IEEE Transactions on Dependable and Secure Computing* 18, 1 (2018), 174–191.

[40] Sigmund Albert Gorski III and William Enck. 2019. Arf: identifying re-delegation vulnerabilities in android system services. In *Proceedings of the 12th conference on security and privacy in wireless and mobile networks*. 151–161.

[41] Md Sadun Haq, Ali Şaman Tosun, and Turgay Korkmaz. 2022. Security Analysis of Docker Containers for ARM Architecture. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE, 224–236.

[42] Mubin Ul Haque, M Mehdi Kholoosi, and M Ali Babar. 2022. KGSecConfig: A Knowledge Graph Based Approach for Secured Container Orchestrator Configuration. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 420–431.

[43] Rupesh Raj Karn, Prabhakar Kudva, Hai Huang, Sahil Suneja, and Ibrahim M Elfadel. 2020. Cryptomining detection in container clouds using system calls and explainable machine learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 674–691.

[44] Tong Kong, Liming Wang, Duohe Ma, Zhen Xu, Qian Yang, and Kai Chen. 2019. A secure container deployment strategy by genetic algorithm to defend against co-resident attacks in cloud computing. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 1825–1832.

[45] Kubevirt. 2023. Adopters. https://github.com/kubevirt/kubevirt/blob/main/ADOPTERS.md.

[46] Kubewarden. 2023. Adopters. https://github.com/kubewarden/kubewarden-controller/blob/main/ADOPTERS.md/.

[47] Lingguang Lei, Jianhua Sun, Kun Sun, Chris Shenefiel, Rui Ma, Yuewu Wang, and Qi Li. 2017. SPEAKER: Split-phase execution of application containers. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 230–251.

[48] Rui Li, Wenrui Diao, Zhou Li, Jianqi Du, and Shanqing Guo. 2021. Android custom permissions demystified: From privilege escalation to design shortcomings. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 70–86.

[49] Xin Lin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, and Quan Zhou. 2018. A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 418–429.

[50] Yang Luo, Wu Luo, Xiaoning Sun, Qingni Shen, Anbang Ruan, and Zhonghai Wu. 2016. Whispers between the containers: High-capacity covert channel attacks in docker. In *2016 IEEE trustcom/bigdatase/ispa*. IEEE, 630–637.

[51] Antony Martin, Simone Raponi, Théo Combe, and Roberto Di Pietro. 2018. Docker ecosystem–vulnerability analysis. *Computer Communications* 122 (2018), 30–43.

[52] Jaehyun Nam, Seungsoo Lee, Hyunmin Seo, Phil Porras, Vinod Yegneswaran, and Seungwon Shin. 2020. {BASTION}: A security enforcement network stack for container networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 81–95.

[53] OpenFeature. 2023. Adopters. https://github.com/open-feature/community/blob/main/ ADOPTERS.md/.

[54] OpenKruise. 2023. Users. https://github.com/openkruise/kruise#users.

[55] Inc O'Reilly Media. 2023. Chapter 1. Container Security Threats. https://www.oreilly.com/library/view/container-security/9781492056690/ch01.html.

[56] Nicholas Pecka, Lotfi Ben Othmane, and Altaz Valani. 2022. Privilege Escalation Attack Scenarios on the DevOps Pipeline Within a Kubernetes Environment. In *Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering*. 45–49.

[57] Akond Rahman, Shazibul Islam Shamim, Dibyendu Brinto Bose, and Rahul Pandita. 2023. Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. *ACM Transactions on Software Engineering and Methodology* (2023).

[58] Michael Reeves, Dave Jing Tian, Antonio Bianchi, and Z Berkay Celik. 2021. Towards improving container security by preventing runtime escapes. In *2021 IEEE Secure Development Conference (SecDev)*. IEEE, 38–46.

[59] Deb Richardson. 2021. What I learned about Kubernetes and Knative Serverless. https://www.redhat.com/en/blog/what-i-learned-about-kubernetes-and-knative-serverless.

[60] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. 2013. Puma: Permission usage to detect malware in android. In *International joint conference CISIS'12-ICEUTE 12-SOCO 12 special sessions*. Springer, 289–298.

[61] Alibaba Container Service. 2020. From Serverless Containers to Serverless Kubernetes. https://www.alibabacloud.com/blog/from-serverless-containers-to-serverless-kubernetes_596533.

[62] Shazibul Islam Shamim. 2021. Mitigating security attacks in kubernetes manifests for security best practices violation. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1689–1690.

[63] Bingyu Shen, Lili Wei, Chengcheng Xiang, Yudong Wu, Mingyao Shen, Yuanyuan Zhou, and Xinxin Jin. 2021. Can systems explain permissions better? understanding users' misperceptions under smartphone runtime permission model. In *30th USENIX Security Symposium (USENIX Security 21)*. 751–768.

[64] Sushrut Shringarputale, Patrick McDaniel, Kevin Butler, and Thomas La Porta. 2020. Co-residency attacks on containers are real. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*. 53–66.

[65] Container Security Site. 2023. Container Breakout Vulnerabilities. https://www.container-security.site/attackers/container_breakout_vulnerabilities.html.

[66] Yuqiong Sun, David Safford, Mimi Zohar, Dimitrios Pendarakis, Zhongshu Gu, and Trent Jaeger. 2018. Security namespace: making linux security frameworks available to containers. In *27th USENIX Security Symposium (USENIX Security 18)*. 1423–1439.

[67] Sahil Suneja, Ali Kanso, and Canturk Isci. 2019. Can container fusion be securely achieved?. In *Proceedings of the 5th International Workshop on Container Technologies and Container Clouds*. 31–36.

[68] Sysdig. 2022. Sysdig 2022 Cloud-Native Security and Usage Report. https://sysdig.com/2022-cloud-native-security-and-usage-report/.

[69] Oren Teich. 2019. Cloud Run, a managed Knative service, is GA. https://cloud.google.com/blog/products/serverless/knative-based-cloud-run-services-are-ga.

[70] Google Cloud terms. [n. d.]. Isolate your workloads in dedicated node pools. https://cloud.google.com/kubernetes-engine/docs/how-to/isolate-workloads-dedicated-nodes.

[71] Google Cloud terms. 2023. GKE Features. https://cloud.google.com/kubernetes-engine#section-2.

[72] Google Cloud terms. 2023. Google Cloud Marketplace. https://cloud.google.com/marketplace.

[73] Google Cloud terms. 2023. Google Kubernetes Engine(GKE). https://cloud.google.com/kubernetes-engine.

[74] Güliz Seray Tuncay, Jingyu Qian, and Carl A Gunter. 2020. See no evil: phishing for permissions with false transparency. In *29th USENIX Security Symposium (USENIX Security 20)*. 415–432.

[75] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. 2017. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1077–1093.

[76] Luyi Xing, Xiaorui Pan, Rui Wang, Kan Yuan, and XiaoFeng Wang. 2014. Upgrading your android, elevating my malware: Privilege escalation through mobile os updating. In *2014 IEEE symposium on security and privacy*. IEEE, 393–408.

[77] Nanzi Yang, Wenbo Shen, Jinku Li, Yutian Yang, Kangjie Lu, Jietao Xiao, Tianyu Zhou, Chenggang Qin, Wang Yu, Jianfeng Ma, et al. 2021. Demons in the shared kernel: Abstract resource attacks against os-level virtualization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 764–778.

[78] Qingyang Zeng, Mohammad Kavousi, Yinhong Luo, Ling Jin, and Yan Chen. 2023. Full-stack vulnerability analysis of the cloud-native platform. *Computers & Security* 129 (2023), 103173.